Performance Optimization of IoT Based Biological Systems Using Deep Learning

Omer Irshad (*Affiliation:* Department of Computer Science and Engineering, The University of Engineering and Technology Lahore, Pakistan, *Email:* umar_irshad@hotmail.com, *Tel/Fax:* +92-42-99029450-300, *Cell:* +923008870760, *Address:* The University of Engineering and Technology, G.T Road, Staff Houses Engineering University Lahore, Punjab P.O. 54890, ***Formal Principal/Corresponding author***),

Muhammad Usman Ghani Khan (*Affiliation:* National Center of Artificial Intelligence, Al-Khawarizmi Institute of Computer Science, The University of Engineering and Technology Lahore, Pakistan, *Email:* usman.ghani@uet.edu.pk),

Razi Iqbal (*Affiliation:* Al-Khawarizmi Institute of Computer Science, The University of Engineering and Technology Lahore, *Email*: razi.iqbal@ieee.org),

Shakila Basheer (*Affiliation:* Department of Information System, College of Computer and Information Sciences, Princess Nora Bint Abdul Rahman University, Saudi Arabia, *Email:* sbbasheer@pnu.edu.sa),

Ali Kashif Bashir (*Affiliation:* (1) Manchester Metropolitan University, UK, *Email:* dr.alikashif.b@ieee.org (2) School of Electrical Engineering and Computer Science, National University of Science and Technology, Islamabad (NUST) , Pakistan.)

***Abstract*- The advent of sensors and high-throughput technologies has resulted in an exponential growth of big biological data. Various distributed biological systems have been deployed for big biological data analytics and providing consolidated information to its end users. Performance optimization plays a significant role while making these systems interactive and responsive. Current performance optimization techniques consider no or fewer history data of the system's functional context while optimizing performance, especially at cache, persistence and computation levels. In this paper, an intelligent multi-agent-based performance optimization approach is proposed that addresses the performance issues at these three levels. Based on the internet of things (IoT) and deep learning paradigm, the proposed approach blends state-of-the-art probabilistic, recurrent neural network and long short term memory models to intelligently predict the upcoming behavior and optimization needs of the system. It intelligently persists and migrates biological data objects among different distributed system nodes. We deployed the proposed performance optimization approach and showed significant performance gain in comparison with existing approaches.**

***Index Terms*- Object Caching, Distributed Biological System, Intelligent Multi-agents, Probabilistic Analysis, Response Time.**

1. INTRODUCTION

BIOLOGICAL cell system is one of the yet studied complex systems. For improving living and health standards of life, researchers, scientists, and biologists are continuously striving for discovering the hidden aspects of a cellular system. To infer and discover valuable knowledge, it is required to computationally understand this complex system through its data and functional processes. Due to the advent of emerging sensors and high-throughput technologies, biological data or data is growing at an exponential rate [1]. Currently, experimental biological data is heterogeneous and available in different types and formats through various data access interfaces. Dozens of computational systems are available to provide a well-aggregated and consolidated view of such geographically dispersed and heterogeneous data [2]. Among these computational systems, distributed biological systems are famous due to enabling distributed computing and providing an integrated view of such huge geographically dispersed and heterogeneous data with better online accessibility, availability, and usability. In such systems, better request turnaround time mainly depends on reasonable system performance at various computational levels like data caching, persisting, processing, routing, etc [3]. Among these computation levels, caching, persisting and processing are especially famous when network or data packet routing issues are not subject to discussion.

Currently, many approaches are available for optimizing system performance at these three levels [4]. Caching techniques share a common objective of improving cache hits while ensuring cache consistency in place. These techniques take current state information of the system, identify cache patterns, and accordingly let the data to reside in cache. These techniques are not proactive enough and do not plan a caching

mechanism based on the history of system states [5]. Computational load balancing algorithms also use current state information of the system for balancing the computational load on distributed computational nodes. These algorithms are activated after the happening of the worst performance situation. Storage devices use a data layout plan for persisting permanent data. These storage layout plans are made according to the current needs of data demand. With the passage of time when data access demand changes, the system re-computes the data layout plan for fulfilling the new requirements of data access demand. Re-computation of data layout plan takes too much computation cost and time in addition to other network resources. Resultantly, until the implementation of a new storage plan, the system keeps showing poor performance.

In this paper, we propose a performance optimization approach for IoT based distributed biological systems that are specialized for data analytics. The proposed approach is based on a probabilistic model that uses state and temporal information management constructs from Recurrent Neural Network (RNN) and Long Short Term Memory (LSTM) models of deep learning paradigm. RNN is a type of neural network, which is used to hold the temporal information for the time-specific problems. LSTM is also the type and the extension of the RNN, which has the special as well as the standard memory gates. Both RNN and LSTM are built into the network and output the memory which has a direct impact of the spatial as well as the temporal information. Our proposed approach continuously and autonomously takes past and current state of the system and probabilistically analyzes it for providing intelligence to the system or its designer for updating the current caching, persisting and computing mechanisms. The proposed approach can add value to the system in a couple of ways (1) system designer can get optimization suggestions to manually redesign caching, persisting and computing mechanism (2) it can be implemented as part of the system. In this case, it will continuously check the need for data reorganization and will autonomously take optimization measures to keep the system's performance up all the time.

Outline of remaining sections is as follows, *Section 2* provides the current state of the work in the cache, storage, and computation performance optimization areas. *Section 3* describes the proposed methodology to optimize the system performance at three levels mentioned in *Section 2*. *Section 4* validates the proposed methodology. *Section 5* shows performance gain after deploying the proposed methodology. *Section 6* concludes the findings, usefulness of our work and future directions.

2. RELATED WORK

In 2018, The Nucleic Acids Research journal reported around 1613 molecular biology databases that are publically available [6]. This number is obtained just after adding around 66 new databases that were reported in 2018 [7]. Among these databases, few comprehensive databases like DNA Data Bank of Japan [8], GenBank [9] and European Nucleotide Archive [10] reported petabytes of the data increase in last few years. In addition to the exponential growth, biological data is heterogeneous and geographically dispersed too. To address the volume, heterogeneity, geographical dispersion, online accessibility and availability related issues, distributed computing paradigm well suits for providing consolidated and unified access of data to its end users.

In the past few decades, several distributed biological systems or distributed systems have been deployed based on different data integration paradigms like view or linked-data systems, workflow-based systems, mashup or smashup based systems, semantic web-based systems, etc. Systems based on these paradigms contain several similarities with respect to their architectural components (e.g., distributed caching scheme, distributed computation and persistence mechanism, load-balancing technique, data migrators, etc.). In these systems, performance plays a significant role in improving the overall system's usability and availability. In these systems, performance can be discussed in different application areas at multiple levels of design complexity. For data caching, there are around seven application areas in various distributed biological systems [11-23] in which caching plays a significant role in the optimization of performance. These areas are a central processing unit, shared memory multiprocessor, distributed shared memory, distributed file management, distributed proxy cache, world wide web, and internet application and integration [24]. In these areas, it is required to temporarily store data for improving the turned around time of the request. Communicate the decision matter.

In the last few decades, a number of caching techniques have been devised to address performance-related issues. For example, some caching strategies give an additional role to their clients [5]. Each client also caches all those objects which were requested and received by it. If some client demands a particular object and its neighboring client holds, then that object is provided by the owner client. Cache hints technique treats information as hints for maintaining the cache consistency [25]. It focuses on the consistency of cache data rather than increasing the cache hits. These hints provide knowledge about the accuracy of the information in the cache. The system gains performance improvement by improving the cache accuracy instead of improving the cache hit ratio. Acceptable accuracy is calculated from calculating the ratio of lookup costs to the costs of identifying and recovering from wrong entries of the cache. For reducing network traffic, a cache strategy that uses multiple distributed cache servers performs well under the limited network span and specific network topology [26].

All distributed cache servers are controlled by a central cache control server. The central cache control server maintains a list for all contents which are cached by the multiple cache servers. If some request needs content, then it is looked in the central cache control server's content list. Caching strategies that follow a linear division of caches at multiple levels usually perform well for the average scale centralized and distributed database systems [27]. In this case, the cache is linearly divided into multiple levels instead of a single level. Each level is further divided into two parts. One part contains a query result and the other part contains a sub-query execution plan. This sub-query execution plan is obtained by dividing the main

query execution plan. These sub-query execution plans are independent but interrelated with each other.

Peer-to-peer (P2P) caching techniques place popular objects at different locations [28]. The algorithm identifies the importance of the object and makes a decision about the generation of its number of copies for disseminating to multiple locations. It does not do over caching because, with the passage of time, less popular objects are replaced by popular ones. For P2P video-on-demand systems, sometimes peer gets selfish and takes service from others but do not provide service to others. To overcome this problem, an incentive-based approach is used. Each peer gets some incentive if it is holding any video that is highly popular [29]. Due to this, the load on the main server is shared by the peers.

Poor performance at the computation level too sometimes causes a bottleneck. Sometimes processing load at the particular processing node gets too high so that some objects are required to be migrated from the overloaded node to the less loaded one. Sometimes a task that may involve multiple objects needs timely availability of required objects to meet its completion deadline. In such a situation, the system's resource manager migrates demanded objects from one location to another one [30]. Various algorithms have been devised to reduce the processing load of a particular node to cope with the complexity of meeting task completion deadlines [31]. In the case of the overloaded processing node, these algorithms first identify the overloaded node and then migrate a few objects to the less loaded node. In the case of highly demanded objects, these algorithms first calculate the expected completion time of the task and then compare it with the assigned deadline. If the task completion deadline seems not to be met, a reasonable migration of demanded objects is scheduled to meet the deadline [32].

Optimization at the storage level too plays a significant role in reducing the speed mismatch between persistent and cache storage. Managing data on the storage-intensive servers or devices comes with lots of data management problems. Such types of servers are usually used as web servers or multimedia servers for handling the high demand for data. These servers usually consist of several storage disks that are internally connected with a dedicated network to form a larger storage cluster. These storage disks are further constrained by the storage size and number of simultaneous access to data [33]. For efficient data utilization, an initial data layout plan is made according to the initial requirements. But with the passage of time, these requirements may change and the system has to re-compute the initial data layout plan to accommodate the new requirements [34]. When the need for a re-computing data layout plan arises, the problem of mapping the initial layout to the target layout also arises. Sometimes the system needs addition, removal or replacements of entire disks. In such cases, entire data from the source disk is required to be reasonably migrated to the destination disk.

All the caching, persisting and processing techniques have a common objective of improving system performance in their respective areas [35]. These techniques use no or less contextual information of user requests and the history of system states. Due to this, the intelligent decision for performance optimization cannot be taken. In addition to this, most of the optimization techniques get activated or responsive after the happening of worst-case situations. In this case, frequent data migrations are performed at the expense of extensive system downtime and additional resources [36]. Furthermore, most of the techniques do not fully support for computing and rendering the object at its constituent level. So the system has to process the whole object even the user does not demand its all constituents.

Significant performance can be obtained, if these techniques are supplemented by the behavioral intelligence of upcoming user requests and system states. Behavioral intelligence can be obtained by statistical and probabilistic analysis of historical data of the system states [37, 38]. Our deep learning-based probabilistic approach provides behavioral intelligence for existing optimization techniques by analyzing the history and current data of user requests and system states. In IoT based distributed biological systems, these states are request size, count, type, origin, scope, date, time, available and required resources, priority, importance, residual affinity, etc. [39, 40]. Our proposed model uses these states as model parameters and works in a feedback-loop manner to autonomously optimize the system performance. So, these are some of the characteristics that make our proposed approach novel, better and distinct among existing approaches.

3. PROPOSED SOLUTION

In a distributed system, performance can be optimized at various levels of the system design. *Fig. 1* pictorially shows the idea of performance optimization in the distributed system at three levels like cache, computation, and storage.

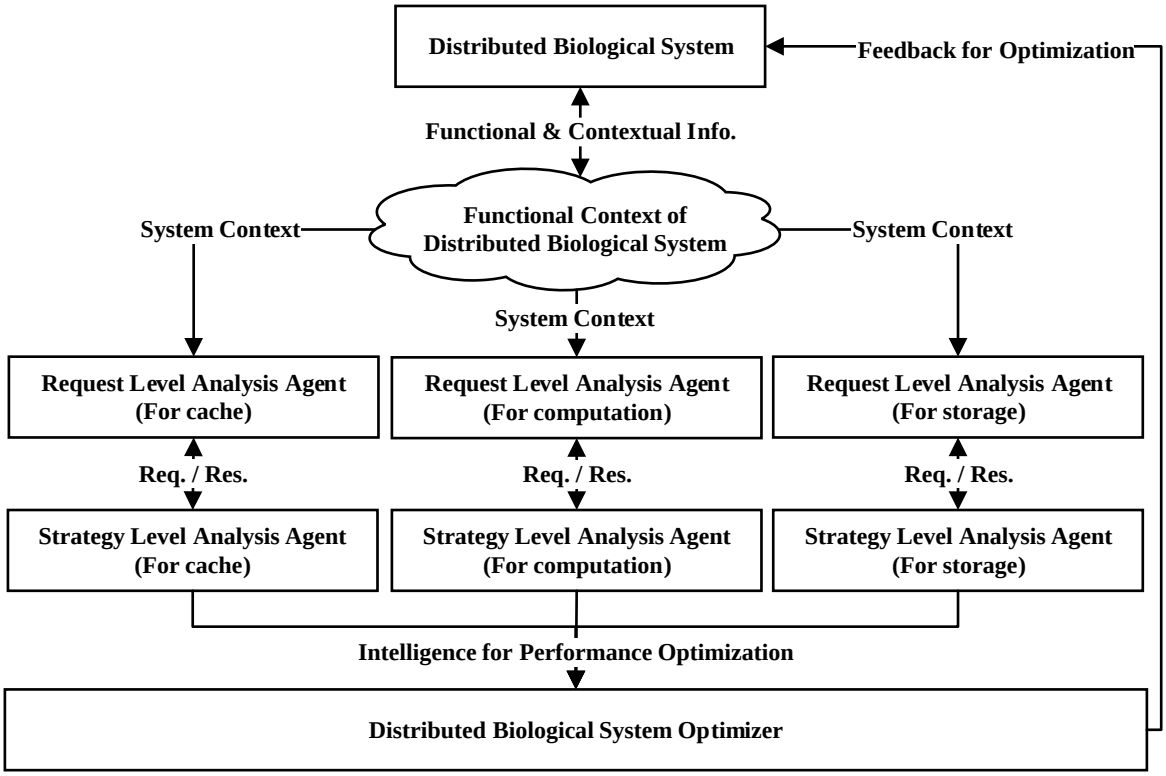

Fig. 1. Performance optimization through multiple intelligent agents.

In *Fig. 1*, *Distributed Biological System* or *System* has a *Functional Context* and six intelligent software agents. *The system* has different Requesting Terminals (RTs) through which *Users* can send requests to the *System* for required data resources. *The system* accepts the requests and processes them to send the response to its end users. The *system* has other channels too through which it can take inputs (e.g., sensors, external systems, web services, etc.), but for the sake of simplicity, we only consider RTs as the request source.

Users from any RT can request Biological Objects (BO) like deoxyribonucleic acid (DNA) sequence, codding or non-codding gene annotation, chromosome region, medical image or video, gene map, graph, pattern, etc. BO may be an aggregation of more than one smaller object (e.g. a DNA has multiple regions marking codding genes, exons, and introns in DNA regions, etc.). This aggregation is primarily established if either some chemical bondings exist among objects or user demands custom aggregation. For example, a long sequence of DNA contains several regions and among them, there are some protein-coding regions that are associated with different proteins and phenotypes. Based on this association, an aggregated BO can be defined that can comprehensively define a concept like a gene involved in protein production, a gene that can cause cancer, etc. *Fig. 2* pictorially shows the aggregation structure as follows.

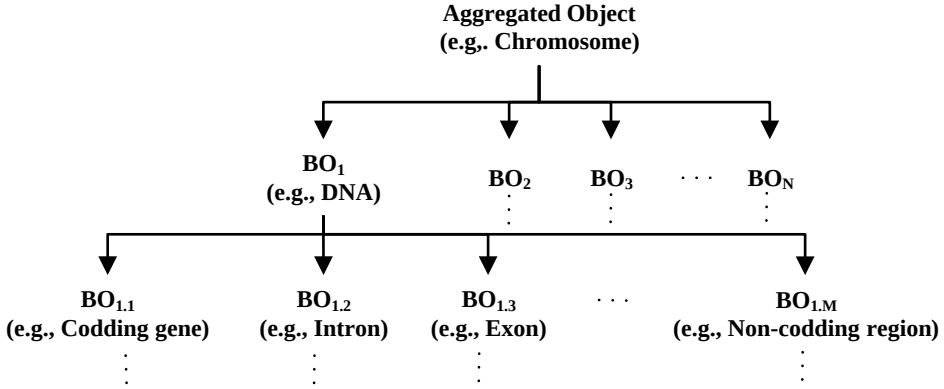

Fig. 2. Aggregation of biological objects.

In *Fig. 2*, BO_1, BO_2, BO_3, …, BO_N are the constituents of *Aggregated Object* and $BO_{1.1}$, $BO_{1.2}$, $BO_{1.3}$, …, $BO_{1.M}$ are the constituents of BO_1. This de-aggregation can be continued at a greater level of the hierarchy. At the bottom of the hierarchal structure, those objects reside that have high cohesion among their constituents and cannot be further de-aggregated.

Most of the time when an aggregated object is demanded, its user just partly consumes it as per the need. Other constituents get wasted at the expense of computational and caching cost. In our proposed approach, we copped this issue by learning from history data of the system state parameters like RT name or geographical location, requested object identifier, request date/time, list of object's constituents used and not with respect to requester and list of supporting objects (i.e., those objects that are required during the execution of object being processed.). *Table 1* shows sample access history of objects requested from different users from different or same RTs.

TABLE 1
SAMPLE ACCESS HISTORY OF BIOLOGICAL OBJECTS

RT	Req. Object	Req. Time	Object's Constituents		Supporting Objects
			Total	Used	
T1	BO_1	10-Jan-2019, 10:00AM	$BO_{1.1}$, …, $BO_{1.M}$	$BO_{1.1}$, $BO_{1.2}$	BO_2, BO_3
T2	BO_1	11-Jan-2019, 11:00AM	$BO_{1.1}$, …, $BO_{1.M}$	$BO_{1.1}$, $BO_{1.2}$	BO_4, BO_3
T2	BO_1	11-Jan-2019, 11:10AM	$BO_{1.1}$, …, $BO_{1.M}$	$BO_{1.1}$, $BO_{1.2}$, $BO_{1.3}$	BO_4, BO_3
T1	BO_2	10-Jan-2019, 09:00AM	$BO_{1.1}$, …, $BO_{1.L}$	$BO_{1.1}$, $BO_{1.3}$	BO_4, BO_1
T2	BO_2	10-Jan-2019, 09:10AM	$BO_{1.1}$, …, $BO_{1.L}$	$BO_{1.1}$, $BO_{1.2}$	BO_4, BO_1
T2	BO_2	11-Jan-2019, 09:10AM	$BO_{1.1}$, …, $BO_{1.L}$	$BO_{1.1}$	BO_4, BO_1, BO_3

The first entry of *Table 1* shows that *T1* requested BO_1 on *10-Jan-2019* at *10:00 AM*. Second, its two constituents, $BO_{1.1}$ and $BO_{1.2}$, were used by its *User* out of its total constituents *(i.e.* $BO_{1.1}$, $BO_{1.2}$… $BO_{1.M}$*)*. Third, this object consumed some services from BO_2 and BO_3 during its lifetime. The rest of the entries can be interpreted in a similar way.

Statistical analysis for data shown in *Table 1* shows that BO_1 is requested 3 times from different requesting terminals between *10-Jan-2019* at 10:00 AM to *11-Jan-2019 at* 11:10 AM. It does not need all of its constituents. $BO_{1.1}$ and $BO_{1.2}$ are used three times and $BO_{1.3}$ is used one time. Among those objects that are providing services to BO_1, BO_3 is used three times, BO_4 is used two times and BO_2 is used one time. Similarly, BO_2 is requested 3 times from different requesting terminals between *10-Jan-2019* at *09:00 AM* to *11-Jan-2019* at *09:10 AM*. It does not need all of its constituents. $BO_{1.1}$ is used three-time, $BO_{1.2}$ is used one time and $BO_{1.3}$ is used one time. Among those objects that are providing services to BO_2, BO_1 and BO_4 are used three times and BO_3 used one time. *Table 2* shows these statistics as

TABLE 2
STATISTICAL ANALYSIS FOR ACCESS HISTORY OF EACH BIOLOGICAL OBJECT

Object	Req. Time From	Req. Time To	Object's Constituents	Required Objects
BO_1	10 - Jan-2019 at 10:00AM	11 - Jan-2019 at 11:10AM	$BO_{1.1}$, $BO_{1.2}$, $BO_{1.3}$	BO_3, BO_4, BO_2
BO_2	10 - Jan-2019 at 09:00AM	11 - Jan-2019 at 09:10AM	$BO_{1.1}$, $BO_{1.2}$, $BO_{1.3}$	BO_4, BO_1, BO_3

The first entry of *Table 2* shows that during the date/time from *10-Jan-2019* at *10:00 AM* to *11-Jan-2019* at *11:10 AM*, BO_1 requested services from BO_3, BO_4, and BO_2 (in descending priority order) and also demanded its constituents $BO_{1.1}$, $BO_{1.2}$, and $BO_{1.3}$ (in descending priority order). The rest of the table entries can be interpreted in a similar way. Based on these statistics, the demand for all objects can be prioritized for performance optimization.

3.1 Request Level Analysis Agent (For Cache)

This software agent is responsible for statistical analysis of the history of requested objects, their constituents and supporting objects requested from different requesting terminals in different time intervals (e.g., *10:00 AM* to *11:00 AM* is a time interval in a day of 24 hours.). With the help of such statistical analysis, the agent recommends the future placement of objects on the cache. This placement is usually done with respect to the specific time intervals. Most of the objects are highly demanded in specific time intervals and get less demanded in other intervals. For example, if statistical analysis shows that BO_1 is demanded *50* times in the time interval from *10:00 AM* to *11:00 AM* and only 2 times in the time interval from *02:00 PM* to *03:00 PM*, then BO_1 needs to be present in cache every day just before *10:00 AM* to *11:00 AM*. Based on this strategy, cache hits and system performance show significant improvement.

3.2 Strategy Level Analysis Agent (For Cache)

The input of this software agent is the output of the *Request Level Analysis Agent (For Cache)* agent. This agent has sufficient knowledge of different caching and server replication schemes. It contains information regarding the characteristics, behaviors, limitations, benefits, etc. of different caching and server replication techniques. This agent perceives the given statistical analysis of the *Request Level Analysis Agent (For Cache)* agent and recommends the best suitable caching or server replication technique. Keeping this agent's function separate from *Request Level Analysis Agent (For Cache)* agent gives localized design level change effect. That is, if in future any new caching or server replication scheme is required to be implemented, then the only design of *Strategy Level Analysis Agent (For Cache)* will be affected.

3.3 Request Level Analysis Agent (For computation)

This agent receives the incoming request information and statistically analyzes the history of requested objects with respect to specific time intervals. The perspective of this agent is to know (1) which object is demanded from which requesting terminal (2) what type of computation objects are required (3) how much time an object needs for completing

its computation (4) what other constituents and supporting objects are required to the object which is being processed (5) in which time interval objects are highly demanded (6) which object is requested before or after the object which is being processed. This agent understands such characteristics of the objects and proposes an execution plan for fulfilling upcoming needs. For example, if the request for BO_1 receives and BO_1 needs its constituents $BO_{1.1}$, $BO_{1.2}$ and BO_2 and BO_3 as its support objects, then the execution plan places all these required objects in some reasonable execution order so that constituents and supporting objects of BO_1 get available before BO_1's execution time. The prime objective of the execution plan is to avoid deadlock situations while ensuring that all required objects are available before the upcoming demands.

3.4 Strategy Level Analysis Agent (For computation)

The input of this software agent is the output of the *Request Level Analysis Agent (For computation)* agent. This agent has sufficient knowledge of different computation load balancing techniques. It contains information regarding characteristics, behaviors, limitations, benefits, etc. of different computation load balancing techniques. This agent understands the given statistical analysis of the *Request Level Analysis Agent (For computation)* agent and recommends the best suitable computation load balancing scheme. Keeping this agent's function separate from *Request Level Analysis Agent (For computation)* agent gives a localized effect of system design change. That is, if in future, some new computation load balancing scheme is required to be implemented, then the only design of *Strategy Level Analysis Agent (For computation)* will be affected.

3.5 Request Level Analysis Agent (For storage)

On storage intensive servers, data is stored according to a data layout plan. This data layout plan is developed according to the need of data access patterns. This agent statistically analyzes the history of requests and identifies the best suitable data access patterns. It identifies the priority of objects with respect to the specific time interval and place so that they can efficiently be retrieved later. It also identifies access relationships among objects with respect to the specific time intervals. For example, in some specific time interval, if BO_1 needs its constituents $BO_{1.1}$, $BO_{1.2}$ and its supporting objects BO_2, BO_3, then data access pattern considers them in a tight access relationship and rest of its constituents and supporting objects in loose access relationship for that specific time interval. Based on this, whenever a specific time interval arrives, the system considers those objects that are highly demanded in that particular time interval and accordingly loads them before their demand.

3.6 Strategy Level Analysis Agent (For storage)

The input of this software agent is the output of the *Request Level Analysis Agent (For storage)* agent. This agent contains sufficient knowledge of different storage management techniques. It contains information regarding characteristics, behaviors, limitations, benefits, etc. of different storage management techniques. This agent perceives the given statistical analysis of the *Request Level Analysis Agent (For storage)* agent and recommends the best suitable storage management technique. Keeping this agent's function separate from *Request Level Analysis Agent (For storage)* agent gives a localized effect of system design change. That is, if in future some new storage management scheme is required to be implemented, then the only design of *Strategy Level Analysis Agent (For storage)* will be affected.

All discussed software agents can be implemented in a multimodular way so that system extensibility can be ensured while minimizing the overhead of system maintenance and change effect. These agents are autonomous and their output is helpful in the system design, development and maintenance activities. Designers and developers learn from agents' outcome and deploy the suggested changes accordingly. This cycle continues until the refinement of the system's performance and cache, storage, and computation load balancing techniques get aligned with the context of the system. Finally, system design level changes get minimized and performance remains up all the time.

4. VALIDATION

For validating the proposed solution, an intelligence of the *Request Level Analysis Agent (For storage)* agent is improved at the storage level of a prototype biological distributed system. This intelligence helps in improving the way of storing data on storage devices and ultimately in improving the performance gain of the whole system. Here only *Request Level Analysis Agent (For storage)* agent is providing intelligence to the system, but this validation mechanism can similarly be implemented for cache and computation levels too.

Intelligence can be gained by various probability calculation approaches (e.g., Frequentist, Bayes, etc.). For example, in the case of Bayes, we can introduce a new data object with some reasonable initial prior and likelihood. With this methodology, the system can calculate the initial posterior. After some reasonable time period, the system can again calculate new posterior with the feedback of the old posterior. That is, this process will work in a feedback loop as shown in *Fig. 1*. In every iteration, the process will get a better understanding of the system's behavior. After some iterations, this iterative process will return a convergence point for the system's maximum performance.

In the frequentist approach, probabilities are calculated on the basis of frequencies of the events. Here event means a data object stored in some storage is requested from some requesting terminal on a specific time interval and processed at some processing terminal. After the happening of each event, the probability of the respective data object will be recalculated with respect to the time interval and storage location. According to the calculated probability value, the object will be migrated to the more suitable storage location

either before the time interval or on the demand of the object. As we do not have any prior information about the occurrences of events, so we will use the frequentist approach to calculate the probabilities.

Suppose we have a biological distributed system, say *"BioDistSys"*, with *n* requesting terminals say RT_1, RT_2 … RT_n, *n* processing nodes say P_1, P_2 … P_n (i.e., one processing node for one requesting terminal.) and *n* storage locations say S_1, S_2 … S_n (i.e., one storage location at one processing node.). Further suppose that we have *m* data objects say D_1, D_2 … D_m, that can be requested from any requesting terminal (s). That is, any number of data objects can be repeatedly requested from any requesting terminal in some specific time interval. With the non-probabilistic approach, any data object can be stored in any storage location. But with the probabilistic approach, data objects will be stored in their respective storage locations with respect to some specific time interval. That is, if RT_1 made a maximum number of requests for D_2 in some specific time interval then it is more suitable to keep D_2 in S_1 for that specific time interval. Similarly, with the passage of time, if RT_2 exceeds from RT_1 in making requests for D_2 in another specific time interval, then it is more suitable to migrate D_2 from S_1 to S_2. In case, when an object is equally demanding for more than one storage location in the same time interval, then we can temporarily replicate the object on all demanding storage locations.

Before going into the algorithmic details, let's formulate the method of computing probability function for calculating the probability of each data object. Let's assume we have three random variables (1) data object *D*, (2) specific time interval *T* and (3) storage location *S*. Variable *D* may take any value from the set of *m* data objects like D_1, D_2 … D_m, variable *T* may take any value from the set of *t* time intervals like T_1, T_2 … T_t and finally variable S may take any value from the set of *n* storage locations like S_1, S_2 … S_n. Each time interval will have a start interval time and an end interval time (e.g. *1:00 AM to 2:00 AM* where *1:00 AM* is a start interval time and *2:00 AM* is an end interval time). We may concatenate day, week or month information with each time interval to make it more informative for further analysis. The value of *t* will depend on the number of defined time intervals. Each time interval may have variable length but the sum of all time intervals must be equal to 24 hours of a day.

In order to calculate the probability of demand for some data object D_i (here i = 1 … m) during time interval T_j (here j = 1 … t) from storage location S_k (here k = 1 … n), we assume that there are total *N* events and a three-dimensional array corresponding to three random variables *D*, *T* and *S*. Each cell c_{ijk} (here i = 1 … m, j = 1 … t and k = 1 … n) of the array contains the number of instances of respective events. For example, if c_{135} contains value equal to 10, it means there are 10 events occurred in the system in which data object D_1 was requested during the time interval T_3 from storage location S_5.

So the maximum probability P_{max} of any data object D_i to be in S_{max} during some specific time interval T_j can be computed as given in the probability equations (1a) and (1b).

$$P(S_k \mid D_i, T_j) = c_{ijk} / \sum_{x=1}^{n} c_{ijx} \quad (1a)$$

$$P_{max}(S_{max} \mid D_i, T_j) = \max \{P(S_1 \mid D_i, T_j), P(S_2 \mid D_i, T_j) \dots P(S_n \mid D_i, T_j)\} \quad (1b)$$

For some $n \geq max \geq 1$ where $S_{max} \in \{S_1, S_2 \dots S_n\}$

Here $D_i \in \{D_1, D_2 \dots D_m\}$, $T_j \in \{T_1, T_2 \dots T_t\}$ and $S_k \in \{S_1, S_2 \dots S_n\}$

The maximum probability P_{max} of some specific time interval T_{max}, in which D_i is in some storage location S_k, can be calculated as given in the probability equations (2a) and (2b)

$$P(T_j \mid D_i, S_k) = c_{ijk} / \sum_{x=1}^{t} c_{ixk} \quad (2a)$$

$$P_{max}(T_{max} \mid D_i, S_k) = \max \{P(T_1 \mid D_i, S_k), P(T_2 \mid D_i, S_k) \dots P(T_t \mid D_i, S_k)\} \quad (2b)$$

For some $n \geq max \geq 1$ where $T_{max} \in \{T_1, T_2 \dots T_t\}$

Here $D_i \in \{D_1, D_2 \dots D_m\}$ $T_j \in \{T_1, T_2 \dots T_t\}$ and $S_k \in \{S_1, S_2 \dots S_n\}$

Similarly, the maximum probability P_{max}, that D_{max} should be in some storage location S_k during some specific time interval T_j, can be calculated as given in the probability equations (3a) and (3b)

$$P(D_i \mid T_j, S_k) = c_{ijk} / \sum_{x=1}^{m} c_{xjk} \quad (3a)$$

$$P_{max}(D_{max} \mid T_j, S_k) = \max \{P(D_1 \mid T_j, S_k), P(D_2 \mid T_j, S_k) \dots P(D_m \mid T_j, S_k)\} \quad (3b)$$

For some $m \geq max \geq 1$ where $S_{max} \in \{S_1, S_2 \dots S_n\}$

Here $D_i \in \{D_1, D_2 \dots D_m\}$, $T = \{T_1, T_2 \dots T_t\}$ and $S = \{S_1, S_2 \dots S_n\}$

Above mentioned probability equations (1a), (1b), (2a), (2b), (3a) and (3b) are collectively forming a probability computing method for calculating the probability of each data object with respect to time and location. These equations help in deciding the migration of an object, from one storage location to another in a certain time interval. Similarly, for computation and cache levels these equations can compute probabilities for migrating objects from one processing terminal or cache to another processing terminal or cache with respect to time, location, and processing load (for computation level only.).

5. Results and Discussions

Based on the proposed probability calculation technique, an algorithm for migration of objects is devised for our supposed system *"BioDistSys"*. This algorithm takes data object set, time interval set and storage locations set as input and returns migration plan as an output. The pseudo-code of the algorithm is online available on our website [41].

Devised algorithm partially uses equations (1a) and (1b) to compute a migration plan for all objects in the system. Each entry of this migration plan identifies the best possible storage location for an object with respect to a particular time interval. This algorithm also supports if more than one storage locations are equally favorable for an object during a specific time interval. In this situation, we make multiple copies of the object accordingly and make multiple entries in

the migration plan for the same object and time interval with different storage locations.

For the concrete understanding of devised algorithm and probability equations, the devised algorithm is executed on the sample data set to elaborate on our proposed technique with the help of a use-case. The sample data set which is presented in *Table 3*, is randomly generated by assuming uniform distribution for data object D, time interval T and storage location S. The Matlab code for the algorithm is online available on our website [41].

TABLE 3
SAMPLE DATASET

Sr. #	Object	Time Int.	Storage Location
1	3	1	2
2	3	3	1
3	1	3	2
4	3	2	2
5	2	3	2
6	1	1	2
7	1	2	2
8	2	3	1
9	3	3	2
10	3	3	1

Here the sample size is 10, the number of distinct objects is 3, the number of time intervals is 3 and the number of storage locations is 2. For the verification and validation of the proposed methodology, we took a small amount of sample data but it can similarly be verified and validated on any real and larger sample size. Each row in *Table 3* corresponds to an instance of an event. That is, the first event depicts that object number 3 was demanded during the time interval number 1 from storage location 2. On the sample dataset, the algorithm calculates the frequency matrix for storage locations as shown in *Table 4* and *Table 5*.

TABLE 4
FOR STORAGE LOCATION 1

	Time Int. 1	Time Int. 2	Time Int. 3
BO_1	0	0	0
BO_2	0	0	1
BO_3	0	0	2

TABLE 5
FOR STORAGE LOCATION 2

	Time Int. 1	Time Int. 2	Time Int. 3
BO_1	1	1	1
BO_2	0	0	1
BO_3	1	1	1

Each cell of *Table 4* shows, the number of times that an object came in the storage location 1 in the respective time intervals. A similar explanation holds for each cell of *Table 5*. On the basis of these statistics, probabilities are calculated as given in *Table 6* and *Table 7*.

TABLE 6
FOR STORAGE LOCATION 1

	Time Int. 1	Time Int. 2	Time Int. 3
BO_1	0	0	0
BO_2	0	0	0.5
BO_3	0	0	.6667

TABLE 7
FOR STORAGE LOCATION 2

	Time Int. 1	Time Int. 2	Time Int. 3
BO_1	1	1	1
BO_2	0	0	0.5
BO_3	1	1	0.3333

Each cell of *Table 6* shows, the probability that an object can come in storage location 1 in the respective time intervals. If any cell contains 0 then it means that in respective time interval corresponding object has 0 probability to come in storage location 1. A similar explanation holds for each cell of *Table 7*.
On the basis of these statistics, a migration plan is calculated and shown in *Table 8*.

TABLE 8
OBJECT MIGRATION PLAN

Object	Time Int. 1	Time Int. 2	Time Int. 3
BO_1	2	2	2
BO_2	0	0	1
BO_3	2	2	1

Each cell of *Table 8* shows the suitable storage location number for a particular object in a specific time interval.

The final output of the proposed algorithm is a data object migration plan as shown in *Table 8*. This migration plan tells that which data object should be at which storage location during a specific time interval. With the help of this migration plan, the system can migrate the data objects before time to the storage location(s) where they should be. Resultantly, requests from any requesting terminal can be processed from its nearest associated processor and storage location. Based on this, there will be no need for data object migration during the run time or request time and hence the system's performance will remain up all the time.

For verifying and validating the whole concept, a concept simulator is developed. The code for this simulator is online available on our website [41]. This simulator executes two types of approaches 1) the common approach: an approach based on common characteristics of existing approaches and 2) the proposed approach.

The simulator gives the same input to both approaches and computes results for each approach. After that, the simulator analysis the results of both approaches and gives the results in terms of performance gain. *Table 9* gives algorithmic details of both approaches in terms of assumptions, input, output and normal course of action.

TABLE 9
ALGORITHMIC DETAILS OF PROPOSED AND COMMON (I.E. BASED ON COMMON CHARACTERISTICS OF EXISTING APPROACHES) APPROACH.

	Common Approach	Proposed Approach
Assumptions	Initially, all objects will be in the same storage location against all the time intervals	Initially, all objects will be initialized according to the migration plan
Input	There will be a sample dataset	
	No migration plan	Migration plan
Output	The total number of migrations happened for each object with respect to time interval and storage location.	
Course of action	Place all objects in the same storage location against all the time intervals 1. Compute migration need for each object. 2. First, find an object in its expected storage location 3. If an object is not present in its expected location then find it in all locations and migrate it from its present location to the expected location. 4. Count this mismatch as one migration.	Place all objects in storage locations according to the plan

5. Do nothing if an object is already in its expected location

Compute the total number of migrations happened in the common approach	Compute total number of migrations happened in the intelligent approach

After getting the results of both approaches, equation (4) computes the performance gain in terms of percent gain.

$$\text{Percent Gain} = ((TMCA - TMPA) / TMCA) * 100 \quad (4)$$

Here *TMCA* stands for Total Migrations with Common Approach and *TMPA* stands for Total Migrations with Proposed Approach (i.e., *TMCA* and *TMPA* show the total number of required migration for both approaches.). *Table 10* shows the migration plan that is computed by the *Percent Gain* formula on the sample dataset given in *Table 3*.

TABLE 10
MIGRATION PLAN

Object	Time int. 1	Time int. 2	Time int. 3
BO_1	2	2	2
BO_2	1	1	1
BO_3	2	2	1

Table 11 and *Table 12* differentiate migration needs for both approaches based on the migration plan given in *Table 10*.

TABLE 11
MIGRATION NEED FOR THE COMMON APPROACH

Object	Number of Migrations Happened
BO_1	11
BO_2	4
BO_3	12

TABLE 12
MIGRATION NEED FOR THE PROPOSED APPROACH

Object	Number of Migrations Happened
BO_1	0
BO_2	4
BO_3	4

So percent gain in performance can be shown in the following equation.

Here TMCA = 27, TMPA = 8
Percent Gain = ((TMCA–TMPA)/TMCA)*100 = ((27– 8)/27)*100 = 70.4%

Percent gain shows that how much percent, proposed approach reduces the need for object migration in a system that uses a common approach. In addition to this, there is a negligible performance and implementation overhead if the proposed technique is integrated with some external system. It is because the proposed technique works with negligible computation and storage complexity. It migrates objects before time and so the user will not see any delay due to the unavailability of the resources and demanded objects.

CONCLUSION AND FUTURE DIRECTIONS

It is a widely accepted fact that biological data is heterogeneously growing and geographically dispersing at an astronomical rate from the last few decades. Distributed computing is one of the paradigms that are suitable for storing and processing such big data. Providing interactive and responsive experience to the end system users is tightly coupled with the reasonable system performance. In this paper an IoT based system performance optimization approach is proposed, that uses state-of-the-art probabilistic models from deep learning paradigm for optimizing the cache, persistence and processing nodes to achieve better performance. Few improved design features make our proposed approach distinct and efficient than other approaches like optimization based on contextual and functional history data, autonomous and loopback mechanism for continuous optimization, IoT based device communication, etc. Furthermore, we are working on enhancing the optimization technique for warehouse and workflow-based biological systems. These systems are friendlier for those users which require freestanding and off-line processing of biological data.

ACKNOWLEDGMENT

This research was funded by the Deanship of Scientific Research at Princess Nourah bint Abdulrahman University through the Fast-track Research Funding Program.

REFERENCES

[1] C.E. Cook, M.T. Bergman, R.D. Finn, G. Cochrane, E. Birney, R. Apweiler. (2016, Jan). The European Bioinformatics Institute in 2016: Data growth and integration. *Nucleic Acids Research*. 44(Database issue). pp. D20-D26. doi:10.1093/nar/gkv1352.

[2] V. Lapatas, M. Stefanidakis, R.C. Jimenez, A. Via, M.V. Schneider. (2015, Dec). Data integration in biological research: an overview. *Journal of Biological Research.* 22(1). doi:10.1186/s40709-015-0032-5.

[3] V. Gligorievi ć, N. Pržulj. Methods for biological data integration: perspectives and challenges. (2015, Nov). *Journal of the Royal Society Interface*. 12(112): doi:10.1098/rsif.2015.0571.

[4] X. Tang, M.T. Kandemir, M. Karakoy, M. Arunachalam. Co-optimizing memory-level parallelism and cache-level parallelism. Proc. of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2019. pp. 935-949.

[5] A. Waterland, E. Angelino, E.D. Cubuk, E. Kaxiras, R.P. Adams, J. Appavoo, M. Seltzer. Computational caches. Proc. of the 6th International Systems and Storage Conference, 2013.

[6] D.J. Rigden, X.M. Fernández. (2019, Dec.). The 26th annual Nucleic Acids Research database issue and Molecular Biology Database Collection. *Nucleic Acids Research. 47(D1)*, pp. D1-D7. Available: doi:10.1093/nar/gky1267.

[7] D.J. Rigden, X.M. Fernández. (2018). The 2018 Nucleic Acids Research database issue and the online molecular biology database collection. *Nucleic Acids Research.* 46(Database issue), pp. D1-D7. Available: doi:10.1093/nar/gkx1235.

[8] J. Mashima, Y. Kodama, T. Fujisawa, *et al*. (2017, Jan.). DNA Data Bank of Japan. *Nucleic Acids Research. 45(D1)*, pp. D25–D31. Available: https://doi.org/10.1093/nar/gkw1001

[9] D.A. Benson, M. Cavanaugh, K. Clark, *et al*. (2018, Jan.). GenBank. *Nucleic Acids Research. 46(D1)*, pp. D41-D47. Available: doi:10.1093/nar/gkx1094

[10] A.L. Toribio, B. Alako, C. Amid, *et al*. (2017, Jan.). European Nucleotide Archive in 2016. *Nucleic Acids Research. 45(D1)*, pp. D32–D36. Available: https://doi.org/10.1093/nar/gkw1106

[11] T. Etzold, A. Ulyanov, Argos P. (2004, Jan.). SRS: Information retrieval system for molecular biology data banks. *Methods in Enzymology.266*, pp. 114–128. Available: https://doi.org/10.1016/S0076-6879(96)66010-8

[12] P. Kersey, L. Bower, L. Morris, A. Horne, R. Petryszak, C. Kanz, *et al*. (2005, Jan.). Integr8 and genome reviews: integrated views of complete genomes and proteomes. *Nucleic Acids Research. 33(Suppl. 1),* pp D297–302. Available: doi: 10.1093/nar/gki039

[13] *The Entrez Search and Retrieval System*, 2nd ed., Bethesda (MD): National Center for Biotechnology Information (US)., In The NCBI Handbook., 2007.

[14] D. Smedley, S. Haider, B .Ballester, *et al*.(2009, Jan.). BioMart - biological queries made easy. *BMC Genomics*. *10(22),* pp. 1–12.

[15] E. Cadag, B. Louie, P. J. Myler, P. Tarczy-Hornoch, "Biomediator data integration and inference for functional annotation of anonymous sequences," in *Pacific Symposium on Biocomputing* 2007, pp. 343–354.

[16] D. Blankenberg, N. Coraor, G.V. Kuster, J. Taylor, A. Nekrutenko. (2011, April). Integrating diverse databases into a unified analysis framework: *A Galaxy approach. Database (Oxford). (bar011),* pp. 1–9.

[17] K. Wolstencroft, R. Haines, D. Fellows, *et al*. (2013, July). The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Research. 41,* pp. W557-W561. Available: doi:10.1093/nar/gkt328.

[18] R. Dowell , R. Jokerst, A. Day, S. Eddy, L. Stein. (2001, Oct.). The distributed annotation system. *BMC Bioinformatics*. *2(1),* pp.7.

[19] M. Wilkinson, H. Schoof, R. Ernst, D. Haase. (2005, May). BioMOBY successfully integrates distributed heterogeneous bioinformatics web services. *The PlaNet exemplar case. Plant Physiol. 138(1),* pp. 5–17.

[20] K.H. Cheung, K.Y. Yip, A. Smith, R. deKnikker, A. Masiar, M. Gerstein. (2005, July). YeastHub: a semantic web use case for integrating data in the life sciences domain. *Bioinformatics. 21(Suppl. 1),* pp. i85–96. Available: doi:10.1093/bioinformatics/bti1026

[21] E.K. Neumann, D. Quan. (2006). Biodash: a semantic web dashboard for drug development. *Pac Symp Biocomput.* pp. 176–87. Available: doi:10.1142/9789812701626_0017.

[22] F. Belleau, M.A. Nolin, N. Tourigny, P. Rigault, J. Morissette. (2008, Oct.). Bio2RDF: towards a mashup to build bioinformatics knowledge system health care and life sciences data integration for the semantic web. *Banff, Canada. 41(5),* pp. 706-716. Available: 10.1016/j.jbi.2008.03.004

[23] O. Irshad, M.U.G Khan. (2019, Feb.). Integration and Querying of Heterogeneous Omics Semantic Annotations for Biomedical and Biomolecular Knowledge Discovery. Current Bioinformatics; 14(8): doi: 10.2174/1574893614666190409112025.

[24] V. Milutinovic. (2000, September). Caching in Distributed Systems. IEEE Concurrency. 8(3). pp. 2-3.

[25] Hasslinger G, Ntougias K, Hasslinger F, Hohlfeld O. (2017). Performance Evaluation for New Web Caching Strategies Combining LRU with Score Based Object Selection. Computer Networks. Available: 10.1109/ITC-28.2016.150

[26] J. Gomez-Vilardebo. (2018, June). A Novel Centralized Coded Caching Scheme with Coded Prefetching. IEEE Journal on Selected Areas in Communications, Special Issue on Caching for Communication Systems and Networks. 35(8), pp. 1904 -1904. Available: 10.1109/JSAC.2017.2721578

[27] J. Hachem, N. Karamchandani, S. Moharir, S. Diggavi. (2018, June). Caching with Partial Adaptive Matching. *IEEE Journal on Selected Areas in Communications, Special Issue on Caching for Communication Systems and Networks. 36(8),* pp.1831 – 1842. Available: 10.1109/JSAC.2018.2845018

[28] Gao G, Li R, Xiao Z, Xu Z. Distributed Caching Strategies in Peer-to-Peer Systems. IEEE International Conference on High Performance Computing and Communications 2011. Available: 10.1109/HPCC.2011.11

[29] Wu. Weijie, T.B. Ma Richard, C.S. John Lui. (2014, Mar.). Distributed Caching via Rewarding: An Incentive Scheme Design in P2P-VoD Systems. IEEE Trans. Parallel Distrib. Syst. 25(3), pp. 612-621. Available: 10.1109/TPDS.2013.94

[30] E. Anderson, J. Hall, J. Hartline, J. Hobbes, A. Karlin, J. Saia, R. Swaminathan, J. Wilkes. (2010, June). Algorithms for data migration. Algorithmica. 57 (2), pp. 349-380. Available: 10.1007/s00453-008-9214-y

[31] Y.B. Zikria, S.W. Kim, M.K. Afzal, H.W., M.H. Rehmani. (2018, October). 5G Mobile Services and Scenarios: Challenges and Solutions. *MDPI Sustainability*. 10(10), pp. 1-9.

[32] Thalheim B, Wang Q. (2013, Sep.). Editorial: Data migration: A theoretical perspective. *Data Knowl. Eng. 87,* pp. 260-278. Available: 10.1016/j.datak.2012.12.003

[33] Roberts G, Chen S, Kari C, Pallipuram V. Data migration algorithms in heterogeneous storage systems: A comparative performance evaluation. IEEE 16th International Symposium on Network Computing and Applications (NCA), Cambridge, MA, 2017. Available: 10.1109/NCA.2017.8171340

[34] Phyu M.P, Phyu M.P, Thein N.L. Efficient storage management for distributed storage system. Proc. SPIE 8350, Fourth International Conference on Machine Vision (ICMV 2011): Computer Vision and Image Analysis; Pattern Recognition and Basic Technologies 2012. Available: 10.1117/12.920129

[35] R. Abbasi, H. Hassan, W. Ahmed, G. Rehman, N.M. F. Qureshi, B. Luo, A. K. Bashir. (2019). Generalized PVO based Dynamic Block Reversible Data Hiding for Secure Transmission Using Firefly Algorithm. Transactions on Emerging Telecommunications Technologies, Wiley.

[36] S Yaseen, S.M.A Abbas, A. Anjum, T. Saba, A. Khan, S.U.R Malik, N. Ahmed, B. Shahzad, A.K Bashir. (2018). Improved Generalization for Secure Data Publishing. IEEE Access. 6, pp. 27156-27165.

[37] M. Z. Khan, M. U. Ghani, O. Irshad, R. Iqbal. (2019, June). Deep Learning and Blockchain Fusion for Detecting Driver's Behavior in Smart Vehicles. Internet Technology Letters. doi: https://doi.org/10.1002/itl2.119.

[38] A. Musaddiq, Y.B. Zikriya, O. Hahm, H.J. Yu, A. K. Bashir, S.W. Kim. (2018, February). A Survey on Resource Management in IoT Operating Systems. IEEE Access. 6, pp. 8459-8482.

[39] Y.B. Zikria, S.W. Kim, O. Hahm, M.K. Afzal, M.Y. Aalsalem. (2019, April). Internet of Things (IoT) Operating Systems Management: Opportunities, Challenges, and Solution. *Sensors*. 8 (19), pp. 1-10. doi: 10.3390/s19081793.

[40] G. Rathee, A. Sharma , H. Saini, R. Kumar, R. Iqbal. (2019, June). A Hybrid Framework for Multimedia Data Processing in IoT-HealthCare using Blockchain Technology. *Multimedia Tools and* Applications.1(23). doi:10.1007/s11042-019-07835-3.

[41] Link: https://ncai.kics.edu.pk/brl-downloads/. (**Note.** The link will publically be available after the acceptance of this paper. However software code has already been uploaded to the submission portal during the paper submission process.)