International Journal of Computing and Artificial Intelligence

E-ISSN: 2707-658X
P-ISSN: 2707-6571
IJCAI 2020; 1(1): 39-45
Received: 22-11-2019
Accepted: 28-12-2019

Manideep Yenugula
Kohls, Milpitas, California, 95035, USA

Raghunath Kodam
Apple, California, 95035, USA

David He
Apple, California, 95035, USA

Corresponding Author:
Manideep Yenugula
Kohls, Milpitas, California, 95035, USA

Multiple data centers intended for latency minimization using artificial intelligence algorithms

Manideep Yenugula, Raghunath Kodam and David He

DOI: https://doi.org/10.33545/27076571.2020.v1.i1a.79

Abstract
Lightweight deep learning data and algorithms center technology have recently advanced to the point where multiple inferences of models tasks can be executed simultaneously on limited data center resources. This allows us to work together towards a common goal instead of focusing on achieving high quality in each individual task. On the other hand, real-time applications are never good with multi-model inferences because of the high total operating latency. For multi-model deployment, algorithms should be fine-tuned to reduce latency as much as possible without jeopardizing safety-critical scenarios. In this study, we employ an open neural networks exchange (ONNX) execution engine to investigate model inference and develop a real-time job scheduling technique for deploying multiple models. Afterwards, a container-based application deployment approach is suggested, and inference jobs are allocated to various containers according to the scheduling techniques. The suggested technique may drastically cut down on total running delay in real-time applications, according to the experimental findings.

Keywords: Latency optimization, multi-model, task scheduling, autonomous driving, AI

1. Introduction
Geographically dispersed data centers linked by the Internet hold vast amounts of data. A large portion of cloud service providers' operating budgets go into covering the energy costs of maintaining the data storage servers and the costs of transporting data between data centers so that all of the many copies of data may be updated. Under the limits of user access latency requirements, very little research has placed data replicates in data centers by simultaneously considering power usage and network transit [1]. Distributed storage systems are notoriously prone to failure, which has led to the widespread use of majority quorum-based consistency of data algorithms [2]. The constant generation of massive amounts of data by enterprises necessitating analysis across geographically distributed sites is a direct result of the globalization of services. Conventional wisdom holds that, for reasons like low latency data processing and the lack of wide-area bandwidth, it is either not practical or wasteful to move all information to a single cluster. The use of geographically dispersed datacenters for processing large data has been more common over the past decade [3]. Online data-intensive services, of which web searches are a subset, are housed in data centers. Hundreds of thousand of index nodes aid the nodes in a distributed architecture search engine. In a partition-aggregate method, these nodes provide search results to an aggregator over numerous interdependent retrieval steps. But it's not simple to optimize these systems for low power consumption, fast reaction, and good search quality [4]. Vehicular networks are vital to intelligent transportation systems because they provide a reliable link between cars and users, which is becoming more important due to the rising demand for mobile multimedia services and the fast development of related technology. On the other hand, there may be a lot of idle computing, communication, and storage capacity in a large number of parked cars [5]. Distributed storage systems often use erasure codes as a means of data loss prevention due to their space-optimal data redundancy. There has been no effort on delivering differentiated latency across several tenants that may have varying latency needs, despite recent advancements on measuring average service delay when erasure codes are applied [6]. Now more than ever, in the age of cloud computing, there is a pressing need to make cloud applications more reliable and elastic. The conventional wisdom is that these objectives may be best accomplished by separating the lifecycle of critical application states

from those of specific application instances; in this approach, data and states are often stored in and retrieved from cloud databases that are physically located near the application code. There are stringent latency constraints on these storage options for state access due to the application's high performance needs. In order to guarantee localization of data for all functions, cloud database instance are deployed on various servers. But since certain states are shared and because application workload patterns are inherently unpredictable, data access across hosts inside the data center (Or between data centers if the application demands it) is unavoidable [7].

Rather of focusing on completing each job to a high standard independently, the activities often have a common purpose and work together to achieve it. Reducing total running latency to fulfill safety requirements is, hence, a major concern in autonomous systems.

2. Related Work

Assuming all data has K copies, the goal of minimizing operational cost while meeting user accessibility latency requirements is addressed in [8]. Authors provide a practical approach that is both operational and latency-aware. Least Cost Data Placement (LCDP) is a method that, based on data access rates, divides data into many groups and then, greedily, chooses K data centers that incur the lowest cost per data group. They demonstrate that the LCDP method approaches the data placement issue with an accuracy of \(\frac{1}{2} \ln |\mathrm {U}|\), where \(|\mathrm {U}|\) is the total amount of users. The results of their simulations show that the suggested method may successfully lower data center operating costs, power consumption costs, and network transport costs.

The MeteorShower architecture was suggested in [9] by researchers; it facilitates sequentially consistent key-value storage across several data centers and is fault-tolerant for read/write operations. The fact that the majority of read operations are carried out locally inside a single data center is a notable feature. Read latency is reduced from dozens of milliseconds to tens of millisecond as a consequence of this. To supplement techniques based on a majority quorum, MeteorShower has a data consistency algorithm. Thus, it retains tolerance for faults, balanced load, and all the other desired qualities of majority quorums. A MeteorShower solution built on top of Cassandra is tested and deployed across several Google Cloud Platform data centers. Despite replicas being stored in different data centers, the MeteorShower framework has been shown to reliably handle read requests without incurring communication delays. As a consequence, read queries now have latency in the tens of milliseconds rather than the hundreds of millisecond that was previously guaranteed, but write requests still have the same latency and fault tolerance. Therefore, MeteorShower is most effective on read-intensive workloads.

To overcome these obstacles, the authors of the aforementioned research propose a two-phase Map-Reduce strategy that takes into account datacenter-to-datacenter cost balancing between bandwidth, storage, processing, migration, and latency. Through simultaneous reduction of all five cost factors, they formulate a combined stochastic integer nonlinear optimization problem, which streamlines the optimization of data transit, resource provisioning, with reducer selection. Additional work is put into designing an effective online algorithm that can decrease the time-averaged operating cost over the long run.

Learn more about how content delivery networks that employ a central data center to service several customers via a common wireless medium fare in terms of energy efficiency in [11]. With a focus on applications that can withstand delay, researchers provide an optimization and design approach to precoding that minimizes overall energy usage while ensuring a certain level of service. An energy-buffering period trade-off is derived in a closed-form equation for single-user scenarios, demonstrating the influence of the major system features on the total energy consumption. Then, for cases involving numerous users, they define an energy minimization issue using a precoding scheme based on minimal mean square error. They provide an iterative approach that approximates the non-convex constraint linearly in order to solve the issue suboptimally, allowing us to get around the non-convexity of the defined problem. At last, numerical results are given to show how successful the solution is.

In order to identify the source of the high tail delay problem in cloud CDNs, the authors of the aforementioned article [12] examine a massive dataset acquired from a well-known cloud CDN supplier and compare it to benchmark data collected on Amazon's Web Services and Microsoft Azure. The efficacy of cloud CDNs can be drastically diminished due to this issue. One major idea is to reduce tail latency by sending requests in parallel to different clouds in cloud CDNs. Nevertheless, in their specific case, application developers frequently have limited funds for cloud services, so there are still unanswered questions about how to efficiently decide how many pieces to download from all of them as well as when to download them. Here, they introduce TailCutter, a framework for scheduling workloads with the stated goal of minimizing tail latency within the budgetary restrictions imposed by application providers. To solve the problem of tail delay minimization in cloud CDNs, they create an algorithm called RHC-based MTMA, which stands for receding horizon control driven maximal tail reduction. Many data centers on AWS and Azure are going to get TailCutter. Extensive testing using a large-scale actual-world traces (gathered from a major ISP) demonstrates that TailCutter may minimize user-perceived delay through up to 58.9% of the 100th percentile, when compared to competing solutions within the same budget.

Find out two crucial features that could influence the system's performance in [13] by conducting experiments: (1) the energy usage and response time are significantly affected by a small number of queries that take a long time to process; (2) the ISN's quality contribution is unrelated to how long it takes to answer a query. Their findings inform the proposal of TailCut, an algorithm that allows ISN-aggregator coordination to reduce energy usage while maintaining quality and latency requirements by carefully discarding lengthy query executions. In addition to meeting the requirements for tail latency and quality, their testing findings demonstrate that TailCut can yield power savings of up to 39%.

With the vehicular edge computing cache strategy suggested in [14], content providers work together to store frequently accessed files in the storage of parked automobiles spread across several parking lots. The suggested VEC caching architecture takes data center capabilities all the way out to the network's periphery. Consequently, overall transmission

delay may be drastically cut in half and duplicate transmissions from distant servers can be eliminated. Here, they provide an iterative ascending pricing auction–based content placement method that aims to reduce average latency for mobile consumers. Comparing the suggested caching technique to the most popular and commonly used one, numerical findings reveal a performance boost of up to 24% with regard to of average latency.

This research explores two scheduling policies-weighted queue and priority queue-to provide and optimize differential latency in erasure-coded storage. It then provides a unique framework for this purpose [15]. Using random file placement as well as service time distributions, they measure service latency for various tenant classes for both policies. In order to reduce differential delay across three distinct choice spaces, they create an optimization framework: 1) Scheduling requests; 2) data insertion; and 3) managing resources. The suggested optimization is solved using efficient algorithms that use bipartite matching as well as convex optimization methods. Elastic service-level agreements are made possible by their technology, allowing them to satisfy the needs of diverse applications. They go on to demonstrate their approach in action by implementing both queuing models in an open-source cloud storage deployment. This configuration mimics three data centers spread out across different locations by reserving bandwidth. In erasure-coded storage systems, experimental findings provide light on service differentiation as well as elastic quality of service by validating their conceptual delay analysis as well as demonstrating a substantial joint decrease in latency for various file classes.

3. Proposed Work

Here we present a common situation when the suggested approach might be useful. Following that, the two types of scheduling techniques that were created are shown. The goal is to reduce the average data fusion delay as much as possible while operating several model inference workloads on a single Data Center.

3.1 DCN Model

Figure 1 depicts the storage system's architecture. In a distributed storage system, a group of storage server or nodes N (where |N|= N) is set up. Storage servers receive all data elements. There are computational capabilities for data analytics included into each storage server as well. It may be necessary to transfer data across storage servers for analytical applications that include several data items. A DCN connects all of the servers. We don't depend on any particular DCN architecture to provide a generic data organization solution. Keep in mind that the end-to-end data flow measurements are the only foundation of our design. From the tree-based Clos with Fat-tree to the recursive DCell with BCube, as well as the adaptable Helios and cThrough, our approach is capable of supporting any conceivable DCN topology.

3.2 Scheduling Models

The first allocation strategy we have designed is the Optimal Solution Selection Method. By the method, the optimal allocation solution is chosen each time, so that the overall latency can be guaranteed to be minimal. The method is described in detail as follows:

The first thing to notice is that one embedded edge devices has n sensor inputs.

$$S = \{s_1, s_2, \cdots, s_i, \cdots, s_n\} \quad (1)$$

s_i is the i-th sensor input, and n is the sensor count.

On top of that, every edge device has n apps installed.

$$A = \{a_1, a_2, \cdots, a_i, \cdots, a_n\} \quad (2)$$

where a_i denotes the i th App.

For n various inputs, the processing and inference delay needs for each input job.

$$L = \{l_1, l_2, \cdots, l_i, \cdots, l_n\} \quad (3)$$

where l_i displays the time it takes for s_i to make an inference. The time it takes for a particular device to finish a single inference job is a constant.

Since the n There are! Assignment options since tasks needs to be given to n apps. Equation (4) displays the assignment matrix.

$$C = \{c_1, c_2, \cdots, c_i, \cdots, c_{n!}\} \quad (4)$$

where c_i denotes the i th distribution policy.

$$\begin{array}{cccccc} & a_1 & a_2 & \cdots & a_n & \\ c_1 & s_1 & s_2 & \cdots & s_n & T_{c_1} \\ c_2 & s_2 & s_1 & \cdots & s_n & T_{c_2} \\ \vdots & \cdots & \cdots & \cdots & \cdots & \vdots \\ c_{n!} & s_{n!} & s_{n-1} & \cdots & s_1 & T_{c_{n!}} \end{array} \quad (5)$$

Equation (6) shows that various applications will be allotted the n jobs in each request round. First, we may test out all of the possible assignment options and get the $T_{a_i c_k}$ through Equation (6).

$$T_{a_i c_k} = T_{a_i}(r-1) + l_j,\ i,j = 1,2,\cdots,n; k = 1,2,\cdots,n! \quad (6)$$

is the total amount of time that App a_i has spent running according to the distribution policy c_k

Then, using Equation (7), we can get the total latency of all the apps for a single option. Using Equation (), we can determine the data fusing latency for each assignment option, which is dependent on the app with the largest delay.

$$T_{c_k} = \max\{T_{a_1}, T_{a_2}, \cdots, T_{a_n} \mid c_k\} \quad (7)$$

in which T_{c_k} stands for the total amount of time spent running in the rth round of requests under the kth allocation policy.

Equation (8) allows us to choose the assignment c_x with the lowest latency, which in turn minimizes the running latency.

$$c_x = \min\{T_{c_1}, T_{c_2}, \cdots, T_{c_{n!}}\} \quad (8)$$

The assignment option c_x may be used to divide up all n jobs across n applications, and Equation (9) can be used to compute the cumulative latency of each app.

$$T_{a_i}(r) = T_{a_i}(r-1) + l_{k|c_x,a_i} \quad (9)$$

$$T_{a_i}(r) = \sum_{m=1}^{r} l_{k|m} \quad (10)$$

T_(a_i) (r) is the total amount of time that has passed since the i-th application ran during the r-th request cycle.
Here, the intricacy of the time required is:

$$T(n) = O(n!) \quad (11)$$

The complexity of the space is:

$$S(n) = O(1) \quad (12)$$

Obtaining the best solution and minimizing running delay of multi-model inference jobs are guaranteed by this technique, since it compares all allocation options and picks the one resulting in the least amount of delay. One drawback of this approach is the time it takes to execute allocation strategy-making while running several model inferences at once. To find a happy medium between allocation strategy runtime and multi-model inference job runtime.

To start, the n inputs to sensors are arranged in descending order of importance:

$$S_{\text{rank}} = \text{Rank}\{s_1, s_2, \cdots, s_i, \cdots, s_n \mid \text{descending}\} \quad (13)$$

Every iteration involves sorting the total operating latency of n applications from smallest to largest:

$$T_{\text{rank}} = \text{Rank}\{T_{a_1}, T_{a_2}, \cdots, T_{a_n} \mid \text{ascending}\} \quad (14)$$

At last, the i th job in the S_{rank} is allocated to the i th app in the T_{rank}:

$$T_{\text{rank}}(r) = T_{\text{rank}}(r-1) + S_{\text{rank}} \quad (15)$$

Here, the intricacy of the time required is:

$$T(n) = O(n \log n) \quad (16)$$

The space complexity is:

$$S(n) = O(1) \quad (17)$$

The strategy's mathematical modeling is shown in Figure 1. At each allocation, the application that is now idle gets the job with the longest execution time, as indicated in the figure. This process continues until the application that is currently active gets the work with the least execution time. By using this method, the total running latency may be significantly decreased.

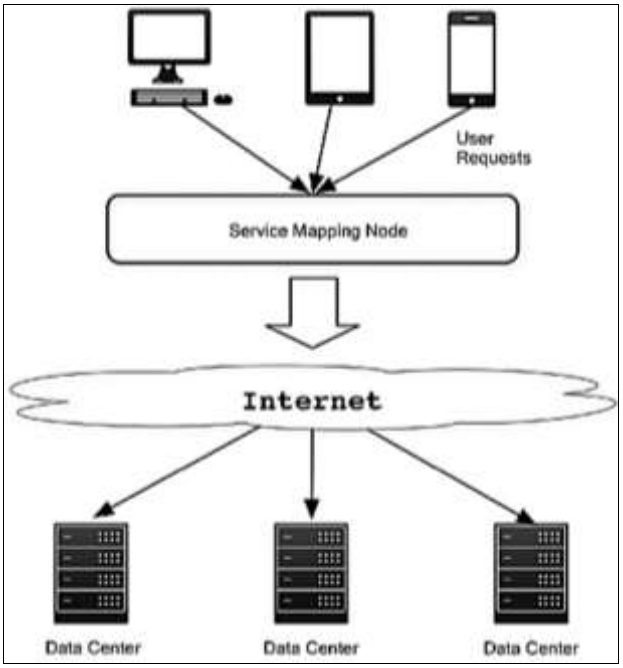

Fig 1: MultiData Center Model

3.3 Latency Minimization using DRL

Cloud companies run several datacenters worldwide to host their cloud-based services, due to regulatory restrictions and the need to minimize latency to end users. Request allocation, the process of allocating user requests to datacenters that offer the best combination of characteristics valued by cloud providers (Such as low bandwidth cost) and end-users (Such as low latency), is a new challenge that is arising under such geo distributed architecture. But previous approaches to request allocation have major flaws: they either optimize benefits for providers or users exclusively, or they optimize benefits for both providers and users while ignoring important but practical considerations, such as users' diverse latency requirements and datacenters' diverse per-unit bandwidth costs.

This proposal proposes to use DeepRL agents in lieu of domain-specific rule-based heuristics. As its topology changes, the DeepRL agent takes action-choosing which connections to activate-receives rewards-depending on link use and flow duration-and updates its policy-through state-action mapping. To be more precise,

- State space: the structure of the network (shown as an sparse matrix with entries representing the connections that are active).
- Action space: various permutations of links (shown as a vectors where each element denotes the likelihood of the matching link to be collected).
- Reward: reduce average flow completion time (FCT) and optimize link usage; that is,

$$R = \sum_{f \in F} \sum_{l \in f} \frac{b_f}{d_f} \quad (18)$$

where F stands for all finished flows and l for all utilized linkages represented by f. The whole time of flow f is denoted by d_f, while the total amount of transferred bytes is represented by b_f.

The agents that embody the data center's functionality are trained offline using a network simulator. A convolutional neural networks (CNN) is used by the learning model. The input state of the CNN is the network topology as well as traffic matrix, and the output state is the policy that specifies which connections in the topology should be activated. To calculate the final probability vectors (policy vector), the convolutional layers first collect spatial information from the network architecture and traffic matrices. Then, they combine these data with those from fully connected layers with softmax. With a decent number of episodes with the capacity to train a solution near to the optimum one across multiple data center topologies, such an ML-based approach established its usefulness.

Figure 1 shows the overall design of. Maintenance of the Q-function encapsulates the essence of Q-learning-based job scheduling:

$$Q: \text{State}(\mathcal{S}) \times \text{Action}(\mathcal{A}) \rightarrow \text{Reward}(\mathcal{R}). \quad (19)$$

Through intense data flows, the storage system's dynamic information (State) may be learnt, allowing one to understand which information item should be stored on the appropriate node (Action A) in order to decrease associated service latencies. After that, the analysis and read/write latencies that were discovered are used as the Reward $\mathcal{R}$ that the recurrent model may be trained. So, DataBot+ is able to provide more effective data placement policies in the long run.

To be more precise, the data arrangement policy and the present state determine the storage sites for data item m before its writing into the storage device at time t $\pi, s \in \mathcal{S}$. After that, the data m is moved to that location by executing the action a, $a \in \mathcal{A}$. All read and analysis operations throughout t and t^', as well as the delay of the last write at t, may be monitored until the data point m is updated at t^'. The right away reward r_t of the action an is calculated as the weighted average of the read/write as well as analytical latencies. Following the action to update at t^', the entire system advances to a different state s^'. As stated in, the present reward r_t at the time t continues to influence the times to follow. A function that maximizes the expectation of the long-term reward is the optimum Q-value function Q^* (s,a).

$$Q^*(s,a) = \max_{\pi} \mathbb{E}[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi], \quad (20)$$

where $\gamma \in (0,1)$ acts as a price reduction $Q^*(s,a)$ is possible to do using the Bellman equation in the following way:

$$Q^*(\boldsymbol{s},a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a') \mid \boldsymbol{s},a\right]. \quad (21)$$

Fig. 2 shows that in a dynamic surroundings, several variables, including request patterns and network circumstances, may impact future rewards. Convergence to the best solution is not guaranteed by standard RL approaches based on temporal differences. This problem is addressed by using the NN to efficiently and accurately estimate the Q-function.

4. Results & Discussion

Here we show the benefits of our methods with regard to latency and provide a detailed description of the experimental setting. Here we lay up the groundwork for comparing performance by introducing the metrics and benchmarks. After that, we'll test our algorithm extensively to see how it performs and what features it has.

In our model, a cloud provider with forty datacenters is considered. We remove the unit from every setting in our simulation for simplicity's sake. In particular, we assigned 1000 as the maximum bandwidth capacity for each datacenter's upstream connection. The range of values for each datacenter's per-unit bandwidth cost is chosen at random from [0.03, 0.3]. Assuming a delay requirement of 50–500 for each request, our simulation takes into account 1000, 1500, as well as 2000 concurrent user inquiries, respectively. Furthermore, the bandwidth required to process each request is randomly selected from the interval [5, 15]. Each datacenter's reaction time, Pi(•), is defined as the remaining capacity times a coefficient, first chosen at random from 1 to 100. Here are two ways that we compared using our algorithm. To start, there's the latency-only method, which avariciously sends all requests to the datacenter having the lowest total delay. The second kind is an algorithm that prioritizes minimizing bandwidth costs; it routes all requests to the datacenter that offers the best deal. For simplicity's sake, we'll refer to our proposed method as "LC," different cost-only and latency-only variants as "LO" and "CO," respectively.

4.1 Single Data Center Scenario

The effect of latency requirement is then assessed. Just so we're clear, we choose the latency requirements lj at random from the intervals [50, 100], [50, 250], and [50, 500], respectively.

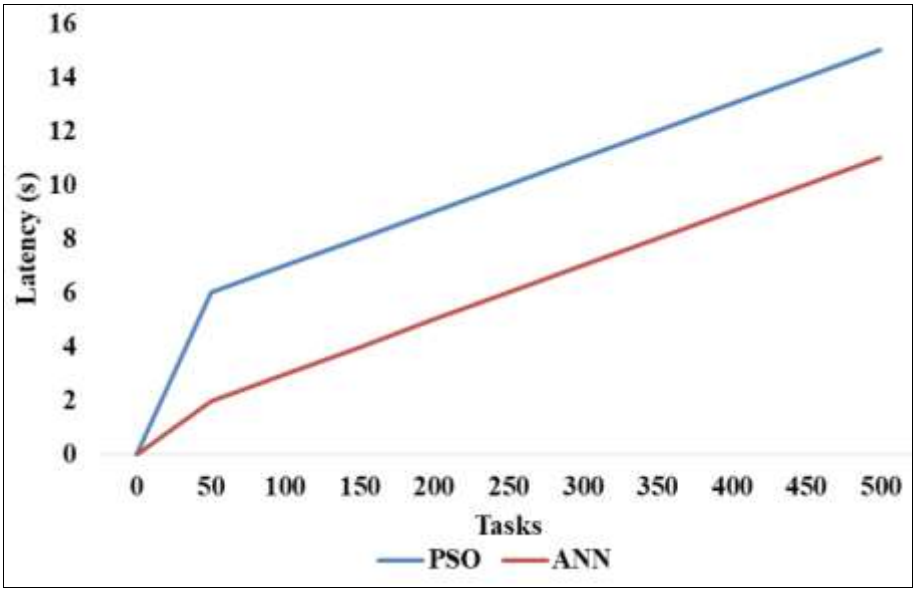

Fig 2: Latency Analysis for Single DCN

4.2 Multiple Data Centers Scenario

The effect of latency requirement is then assessed. Just so we're clear, we choose the latency requirements lj at random from the intervals [50, 100], [50, 250], and [50, 500], respectively.

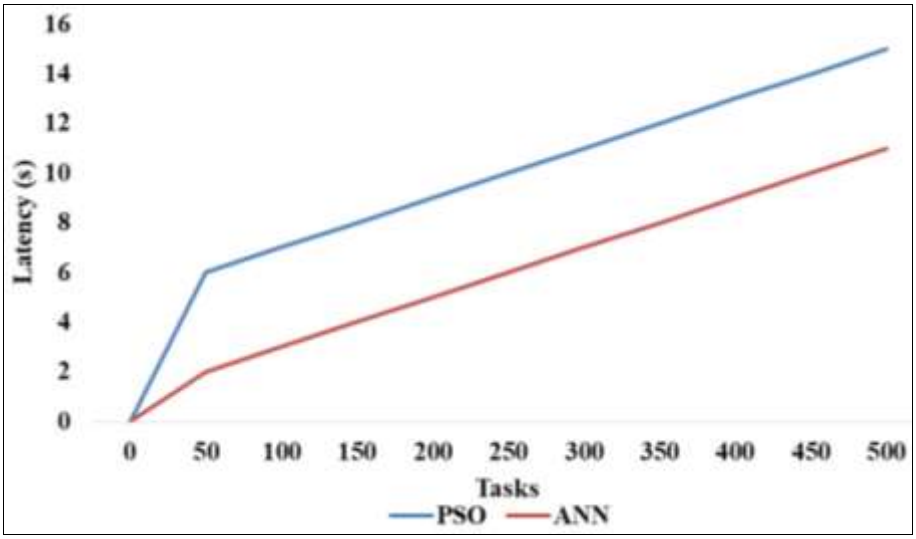

Fig 3: Latency Analysis for Multiple DCN

The delay that users encounter greatly affects their queries. Consequently, the total delay fulfillment of user requests is also assessed in this work. According to Fig.2, Fig 3, LC dosage may completely satisfy the latency needs of all requests, even if it cannot lower latency on its own. The frequency of user requests that meet the delay criterion for various algorithm types is shown in Table 2. From this, we may deduce that CO is unable to meet the latency demands of any user request, while LO and LC are. The rationale for this is that CO completely disregards the latency needs of user requests in favor of optimizing the bandwidth cost.

5. Conclusion

To ensure that end customers' latency needs are met while lowering the overall bandwidth cost for suppliers of cloud services, this article examines an emerging topic of how to allocate each user requests to an appropriate information center. To make it easier to solve, we first transform the integer programming issue with a continuous convex optimization issue. Next, we develop a random-sample request allocation procedure to guarantee that the original optimization problem's solution is viable, and we get the request allocation decision appropriately. A tight upper limit for the overall bandwidth cost may be obtained using our technique, as we have shown. Lastly, we do thorough simulations. When compared to more traditional methods, our suggested algorithm guarantees end users' latency needs at a lower cost to cloud service providers.

6. Future Work

This is a challenging task because automatically suggesting a single solution that satisfies the subjective preference of each stakeholder has remained an open problem for decades

7. References

1. Liu Y, Yu FR, Li X, Ji H, Leung VC. Distributed resource allocation and computation offloading in fog and cloud networks with non-orthogonal multiple

access. IEEE Transactions on Vehicular Technology. 2018;67:12137-12151.
2. Wang P, Yao C, Zheng Z, Sun G, Song L. Joint task assignment, transmission, and computing resource allocation in multilayer mobile edge computing systems. IEEE Internet of Things Journal. 2019;6:2872-2884.
3. Szalay M, Mátray P, Toka L. Minimizing state access delay for cloud-native network functions. 2019 IEEE 8th International Conference on Cloud Networking (CloudNet); c2019. p. 1-6.
4. Chappell SP, Beaton KH, Miller M, Lim DS, Abercromby AF. BASALT 1: Extravehicular activity science operations concepts under communication latency and bandwidth constraints at Craters of the Moon, Idaho.
5. Boviz D, Chen CS, Yang S. Cost-aware fronthaul rate allocation to maximize benefit of multi-user reception in C-RAN. 2017 IEEE Wireless Communications and Networking Conference (WCNC); c2017. p. 1-6.
6. Mane TP, Kanade S. Congestion control mechanism for TCP in data centered networks using multithreading.
7. Nagavalli M. Highlights some of the major issues in cloud services and analysis of various data center selection algorithms.
8. Fan Y, Wang C, Zhang B, Hu D, Wu W, Du DZ. Latency-aware data placements for operational cost minimization of distributed data centers. International Conference on Database Systems for Advanced Applications; c2020.
9. Liu Y, Guan X, Vlassov V, Haridi S. MeteorShower: Minimizing request latency for majority quorum-based data consistency algorithms in multiple data centers. 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS); c2017. p. 57-67.
10. Ramya SR, Prasad TV. Large scale data processing from multiple data centers.
11. Vu TX, Lei L, Vuppala S, Chatzinotas S, Ottersten BE. Energy-efficient design for latency-tolerant content delivery networks. 2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW); c2018. p. 89-94.
12. Cui Y, Dai N, Lai Z, Li M, Li Z, Hu Y, *et al*. TailCutter: Wisely cutting tail latency in cloud CDNs under cost constraints. IEEE/ACM Transactions on Networking. 2019;27:1612-1628.
13. Chou C, Bhuyan LN, Ren S. TailCut: Power reduction under quality and latency constraints in distributed search systems. 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS); c2017. p. 1465-1475.
14. Wang S, Zhang Z, Yu R, Zhang Y. Low-latency caching with auction game in vehicular edge computing. 2017 IEEE/CIC International Conference on Communications in China (ICCC); c2017. p. 1-6.
15. Xiang Y, Lan T, Aggarwal V, Chen YR. Optimizing differentiated latency in multi-tenant, erasure-coded storage. IEEE Transactions on Network and Service Management. 2017;14:204-216.