





Balancing Computing and Networking in Autonomous Edge Clouds

Shinnosuke Masuda 
Kyoto University

Tsuyoshi Hasegawa 
Kyoto University

Kazuyuki Shudo 
Kyoto University

Kenjiro Cho 
III

Abstract—In the ever-changing environment of edge cloud systems, it is necessary to allocate diverse cloud resources in an efficient manner, at the same time, reducing energy consumption. A simple distributed resource allocation model is proposed in a previous study that utilizes a pseudo cost function of resource load, and it is shown that a convex function allows energy saving by pooling idle edge servers. In this paper, we extend this cost model beyond simple load, and present a method to balance computation and communication by merging multiple constraints into the cost function. We show trade-offs among soft constraints along with energy saving effects by simulation, using specific scenarios with computational and communication constraints. Our simulation results show that the combination of a convex function with a polynomial penalty function works well for balancing energy saving and other soft constraints.

Index Terms—edge computing, autonomous resource management, convex cost function, cloud morphing

I. INTRODUCTION

In the future edge computing, diverse and geographically scattered computing resources such as small PC servers and micro-datacenters can be utilized as part of larger cloud services. To make full use of such diverse resources in an ever-changing edge cloud environment, it requires a flexible and efficient resource management model that dynamically optimizes the resource usage considering multiple constraints such as edge server load and network latency. At the same time, the system needs to be resilient in coping with unexpected events and easy to operate so that it should be decentralized and autonomous.

The cloud morphing model [1] proposes a method for autonomously allocating resources using pseudo cost functions. It is a distributed algorithm to find the best computing node for a given job, by a simple cost-minimizing job allocation method. The cost of a resource dynamically changes as a function of resource load, which works to avoid over-concentration, also known as congestion pricing [2], [3].

A cost function can be used for both hard constraints and soft constraints. Hard constraints are those that the system must always adhere to, while soft constraints are those that the system should strive to meet as much as possible. System designers determine the priorities of these constraints based on the objectives they wish the system to achieve. A cost function that monotonically increases with load can implement congestion avoidance as a soft constraint as well as capacity limit as a hard constraint.

In addition, the use of a convex cost function is proposed in [1] to maintain the load of active servers in a target range,

and to conserve energy by placing idle servers into a standby mode. The paper primarily explains the behavior of convex functions on server load, and does not examine interactions with other constraints such as communication requirements.

In this paper, we investigate how to balance computation and communication costs, by incorporating different constraints into cost functions, and validate their effects by simulation. In our pseudo cost function, hard constraints are enforced by a barrier function that increases to infinity as the load approaches the capacity limit. On the other hand, soft constraints are implemented as some form of penalty function that raises the cost as the system deviates from the constraints. For example, a convex function is used to keep servers' load in a certain range, and a monotonic polynomial function is used for communication distance soft constraint. Our simulation results show that the combination of convex functions and polynomial penalty functions works well for balancing different soft constraints such as energy saving and the communication distance constraint.

II. RELATED WORK

The penalty and barrier function methods approximate a constrained optimization problem with an unconstrained one [4], and are used in robotics for safety critical control [5]. The cloud morphing model [1] proposed by Cho *et al.* can be viewed as an attempt to apply penalty and barrier functions into cloud resource management, assuming that micro-services would evolve to enable flexible utilization of diverse edge resources. They propose a simple allocation model that minimizes pseudo costs by applying congestion pricing [2], [3] to resource allocation in edge clouds. In this model, the pseudo cost sharply increases as the server load approaches its capacity, reflecting the capacity hard constraint in the function. At the same time, the pseudo cost is also used to avoid congestion, which works as a penalty function when the load exceeds the target value. Moreover, by using a convex function for the pseudo cost, they propose a method called **idle-resource pooling** that reduces energy consumption by maintaining the load of active servers near the bottom of the convex curve and keeping unnecessary servers in a standby mode.

Although the proposed model consists of computational and communication costs, the paper [1] primarily focuses on explaining the behavior of idle-resource pooling using convex

functions. It does not delve into how to balance computation and communication or their interaction. Furthermore, the pseudo cost is simply a function of load and is not used for constraints other than capacity. In this paper, we extend the pseudo cost function model to incorporate multiple constraints, using communication distance as an example, and show specific methods for balancing computation and communication.

As for other prior studies, there exist rich collections of work in optimizing resource allocation in edge cloud systems [6]–[8], in congestion pricing or smart data pricing for networking [9]–[12], and in carbon-aware networking [13]–[15]. In one of the early studies that used cost functions for networking, Murphy *et al.* [16] applied cost functions to network bandwidth allocation. Xu *et al.* [6] proposed an auction model for the edge computing infrastructure layer. Wagner *et al.* [17] used congestion pricing for resilient job allocation in a distributed military cloud. Our approach differs from these in using convex functions and considering different types of soft constraints, in the context of micro-service based edge resource allocation.

III. MODELS

We follow the system model proposed in [1] and use the same but slightly simplified notations as in Fig.1.

A. System Model

When a user (who could be a drone) initiates a service request to a nearby service allocation server (not shown in Fig.1), the server creates a series of micro-service jobs to fulfill the request. The server obtains the necessary resource information for each job, identifies the locations of the user and the required data object, and asks nearby edge servers for available resources and their current costs. The micro-job is then allocated to the edge that minimizes the overall cost for the service. Note that the pseudo cost is a metric for the resource allocation purpose and should not be confused with monetary charge.

We assume that each micro-job is ephemeral, short-lived and independent of each other, which allows stateless micro-job allocation. Although it may not be true for the currently available micro-services, we believe it would be feasible in the future. We also assume that a user finds a nearby allocation server by a discovery protocol, and the allocation server knows nearby edge servers a priori.

B. Micro-job Assignment

In our model, a micro-job is defined as $J(p, q, r, s)$. Here, p denotes required computational units, q denotes frontend communication amount with the user, r denotes backend communication amount with data object, and s is the number of time slots. The communication costs incurred with q and r are distance-dependent so that an interactive job having $q \gg r$ will be placed close to the user, and a data-intensive job with $q \ll r$ will be placed closer to the data. For the sake of simplicity, we do not differentiate the directions of

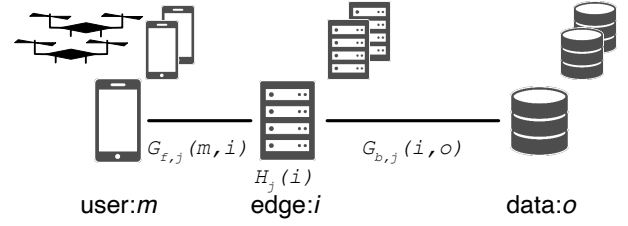


Fig. 1: System model: finding the cost minimizing edge: i for executing a requested micro-job: j from user: m using required data: o .

communication for q and r , and assume only one user and one data object per micro-job in this paper.

A micro-job is specified by a service provider, who can optionally associate weights with it. For instance, a service provider may assign a higher weight to the frontend communication of a delay-sensitive job to ensure the job is placed close to the user. In this manner, cloud service providers can prioritize which type of resources should be used for a specific service.

When a user requests a micro-job, the allocation server identifies the optimal edge to allocate the required resources for J : p , q , and r for duration s . The pseudo cost E to host micro-job j for a unit time at edge i for user m and data object o is calculated as the sum of the computing cost and the communication cost:

$$E_j(i) = H_j(i) + G_j(i, m, o)$$

here, $H_j(i)$ represents the computing cost to run micro-job j at edge i , and $G_j(i, m, o)$ denotes the communication cost to run micro-job j at i between m and o .

$H_j(i)$ is dynamically calculated using $f(\rho)$, a cost function of resource load ρ . Similarly, $G_j(i, m, o)$ is the sum of the frontend communication cost $G_{f,j}$ and the backend communication cost $G_{b,j}$ and calculated with $g(\rho, d)$, a cost function of both load ρ and distance d :

$$\begin{aligned} H_j(i) &= p \cdot f(\rho_i) \\ G_j(i, m, o) &= G_{f,j}(m, i) + G_{b,j}(i, o) \\ &= q \cdot g(\rho_{mi}, d_{mi}) + r \cdot g(\rho_{io}, d_{io}) \end{aligned}$$

To assign micro-job j , the server simply selects the edge that minimize the cost: $\text{argmin}_i E_j(i)$.

C. Pseudo Cost Functions for Computing

The pseudo cost of a resource, $f(\rho)$, is a function of load, designed as a barrier function for optimization [4]; the capacity constraint is enforced by a penalizing cost when approaching the full capacity.

The unique idea in [1] is the use of a convex function for **idle-resource pooling** that tries to keep the load of active resources in the appropriate range, resulting in placing unused resources to a standby mode for energy saving.

The **standard convex cost function** is shown in Fig.2 and defined as: $f(\rho) = (2\rho - 1)^2 / (1 - \rho) + 1$. This function is

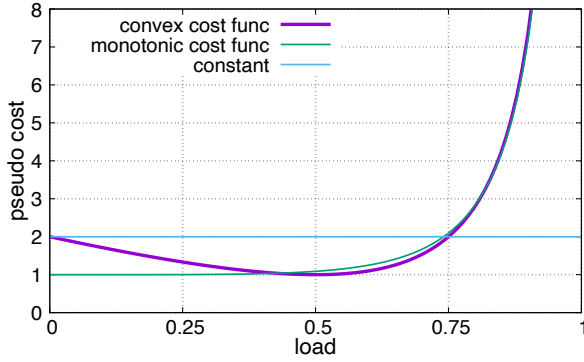


Fig. 2: Cost functions for computing: convex, monotonic and constant.

designed to have the properties: $\min f(\rho) = f(.5) = 1$, and $f(0) = f(.75) = 2$. The cost grows rapidly when $\rho > .75$, which would activate an idle resource in the pool. Therefore, the system automatically tries to keep $\rho \leq .75$, aiming at $\rho = .5$.

On the other hand, the **standard monotonic cost function** tries to distribute the load equally, and is defined as: $f(\rho) = \rho^{4.5}/(1-\rho) + 1$, to roughly match the standard convex cost function in $[.5, .75]$, the working load range. We also use the **constant cost function** that is not affected by load for purposes of comparison: $f(\rho) = 2$.

D. Hard Constraints vs. Soft Constraints

In our pseudo cost model, a hard constraint is enforced by a barrier function and a soft constraint is realized by a penalty function. Our monotonic function of a resource load works as both hard constraint and soft constraint; it is a hard constraint that the load cannot exceed the capacity, at the same time, it is a soft constraint that avoids congestion as the load increases. Our convex function realizes another soft constraint that tries to keep the load at the target working range.

A cost function can have multiple constraints, by adding a different term for each constraint. Also, the variable for a constraint is not limited to the load of a resource; for example, latency or CO₂ emissions could be a constraint variable.

The behaviors for hard constraints are straightforward because they rarely interfere with each other, but the interactions among different soft constraints require trade-offs in compromising each constraint even though they are automatically adjusted by cost functions. We will illustrate interactions of soft constraints using specific examples.

E. Simulation Model

In our simulation, we use a scenario in which a flock of drones are assigned to nearby edge servers in order to show the interactions of constraints. The computing constraints are that each edge server can handle up to a limited number of drones at a time, and it is preferable to make idle servers as many as possible. The communication constraint is that the distance between a drone and the assigned edge should be less than a predefined threshold.

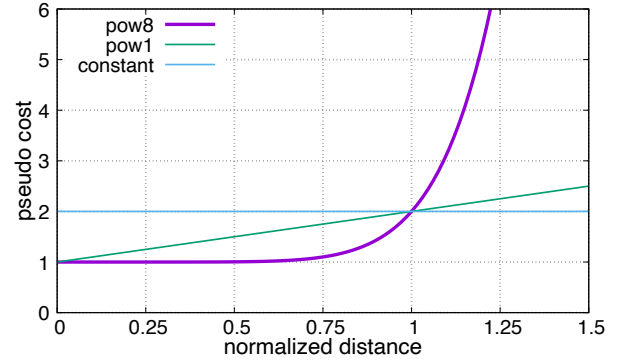


Fig. 3: Network distance cost functions: pow8, pow1 and constant.

For simplicity, we assume the network capacity is abundant so that the frontend and backend communication loads can be ignored (e.g., $J(p = 1, q = 1, r = 0, s = 1)$ and $g(\rho, d) \approx g(d)$). Note that we use only 2 soft constraints for the simulation to illustrate interactions between different soft constraints, since interactions are similar with 3 or more constraints. Also, the use of drone distance for the communication cost is not very general but it is used as a simple example to illustrate the location-dependent nature of the communication cost.

The distance constraint is a soft constraint and expressed by a polynomial function: $G = g(d) = (d/D_{th})^n + 1$. Here, d is the distance between a drone and the assigned edge server, and D_{th} is the soft distance threshold. We use $n = 1$ and $n = 8$ to illustrate the difference, shown as *pow1* and *pow8* in Fig.3. In general, a threshold-based soft constraint can be represented with this type of functions. As n becomes larger, the slope of the function becomes steeper, which has 2 effects: when $d < D_{th}$, a minor difference becomes more negligible and when $d > D_{th}$, the penalty becomes stronger. We also use $n = 0$ as *constant* in Fig.3 that ignores distance.

Thus, the total cost E for assigning drone m to edge server i is a function of the load of i and the distance d between m and i :

$$E(i) = H(i) + G(i) = f(\rho_i) + g(d_{mi})$$

IV. EVALUATION

To illustrate the interactions of different soft constraints in balancing computation and communication costs, we conduct experiments through simulations. In the simulation, a flock of drones move around on a square area and are dynamically assigned to edge servers, based on the server's load and the distance to the server. When a drone requests a micro-job, it is assigned to the edge server having the lowest cost, and the drone requests another micro-job when the micro-job is finished. The simulator plays a role of the allocator who knows the loads of the edge servers and the position of a job-requesting drone.

For the computational cost, we compare 3 functions in Fig.2: the *convex* function, the *monotonic* function, and

TABLE I: Simulation settings

Number of servers	10
Map size	300×300
Distance threshold	150
Max number of allocatable drones per server	25
Number of drones (total Load=0.2)	50
Number of drones (total Load=0.4)	100
Number of drones (total Load=0.6)	150
Micro-job duration (steps)	8-24
Simulation length (steps)	10,000

constant. The *convex* function is our main target that tries to keep the load of an active server between the range $[0.5, 0.75]$ and, as a result, enables idle-resource pooling by shifting loads to non-idle servers. The *monotonic* function tries to distribute load equally among servers. The behaviors of the *convex* and *monotonic* functions are similar when the load is higher than 0.5 so that their differences can be observed only when the load is light. The *constant* ignores the server load and used as a baseline.

For the communication cost, we compare 3 functions in Fig.3: *pow8*, *pow1* and *constant*. The *pow8* function is our main target to keep the distance roughly within the distance threshold. With *pow8*, the impact of distance is negligible when $d < 0.75 \times D_{th}$ and the penalty quickly increases when d exceeds D_{th} . With *pow1*, the cost is a linear function of distance which is a weaker constraint than *pow8*, and used to illustrate differences from *pow8*. With *constant*, the distance is ignored and used as a baseline.

A. Simulation Settings

The parameter settings used in the simulation are shown in Table I. The number of servers is set to 10, and placed non-uniformly. Each server can handle up to 25 drones at a time, allowing for a total of 250 drones to be processed simultaneously. The drones move around as flocks using the boids algorithm [18].

B. Comparing different cost functions

1) *Overview by snapshots*: To illustrate the behaviors of the different cost functions, we use snapshots of the simulation outputs shown in Fig.4; Fig.4a and Fig.4b consider only server load, Fig.4c considers only communication distance, and Fig.4d combines both constraints and automatically balances the different soft constraints. The snapshots are taken at a specific time in the same simulation scenario so that the positions of the drones are the same in the snapshots. This scenario uses 100 drones, and the overall load on the servers is kept at 0.4 (100/250). Later, we vary the total load by increasing the number of drones.

Each snapshot figure has 3 panels: the drone assignment map on the left, the server loads on the upper right and the drone distance on the lower right. The drone assignment map shows the positions of the edge servers with circles and the drones with black points, as well as their assignments with dotted lines. The color of an edge server shows the load of

the server. The plots on the right show the load of each server and the distance to the assigned server for each drone.

Fig.4a shows a simple load balancing among the server by means of the *monotonic* cost function, without considering the distance cost (*constant*). With the *monotonic* cost function, a new micro-job is assigned to the server having the lowest load so that all the servers have similar load. In the figure, all the servers have the load at around 0.4, while the distance is ignored.

In Fig.4b, the cost function for the server is replaced by the *convex* cost function for idle-resource pooling that tries to keep the load of active servers in the target range $[.5, 0.75]$ and make the other servers inactive. In the figure, 6 active servers have the load at around 0.65 and 4 servers are idle. Again, the distance is ignored.

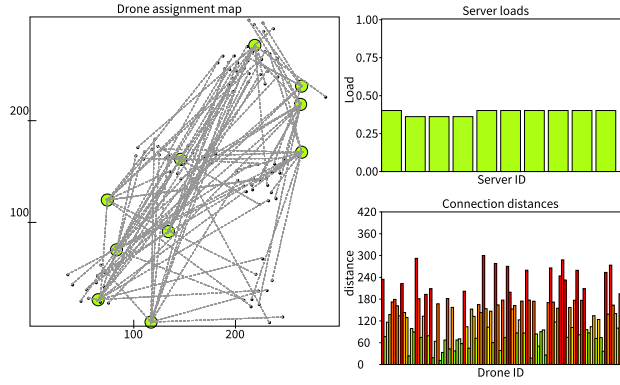
On the other hand, Fig.4c considers only the distance and ignores the server load. Each drone is assigned to the closest server by means of the polynomial cost function. If the closest server is full, the drone is assigned to the next closest server. In the figure, most of the drones are assigned to the closest server, but some are overflowed to the next closest server.

In Fig.4d, both server load and distance are considered by combining the *convex* cost function for the server load and the polynomial cost function for the distance. We can observe that the load of the active servers are less than 0.7 and 3 servers are idle, and the drone distances are less than the threshold of 150. In this snapshot, both constraints are fully satisfied but it is not always the case during the simulation. When it is not possible to satisfy both load constraint and distance constraint, the trade-off is made so as to minimize the sum of the penalty costs.

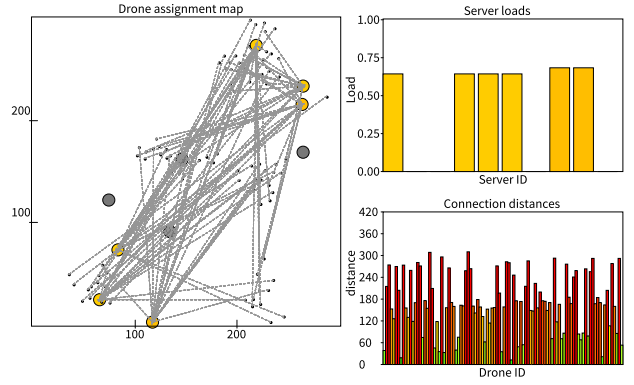
2) *Comparing metric distributions*: The snapshots in Fig.4 are taken at a specific time during the simulation. Next, we examine the distributions of 3 metrics over the entire duration of the simulation in Fig.5: the non-idle server load, the number of idle servers, and the drone distance. The distributions are shown by violin-plots with quartile bars, grouped by the 3 server cost functions, and each group has the 3 distance cost functions.

In Fig.5a, the median load values of the *convex* cost group are around 0.7, much higher than the *monotonic* and *constant* cost group with the median load values at around 0.4. The *convex* cost group is able to maintain the load roughly within the target range $[.5, .75]$, where *constant* that ignores the distance has the shortest tail while *pow1* and *pow8* have slightly longer tails. The *monotonic* cost group, distributing the load evenly among the servers, results in lower load values than the *convex*. The *constant* cost group ignores the load so that some servers experience capacity overflow.

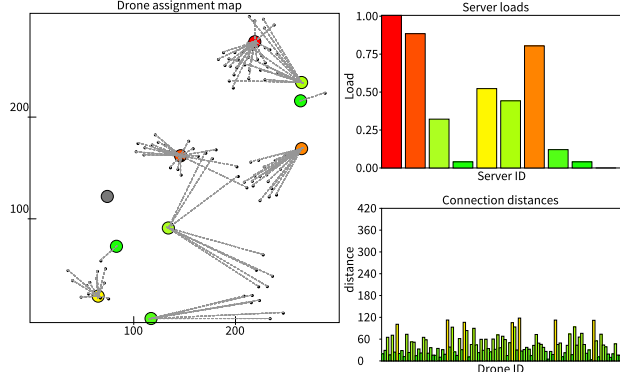
In Fig.5b, the median numbers of idle servers are around 3.5 for the *convex* cost group, around 1.0 for the *monotonic* cost group, and about 2.0 for the *constant* cost group. The *convex* cost group, by keeping the higher load for the active servers, is able to make more servers into the idle state for energy saving.



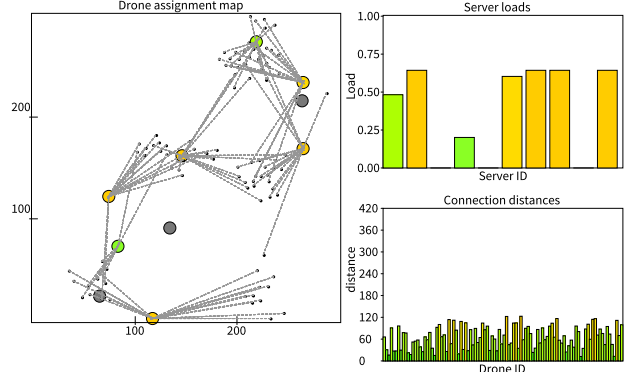
(a) monotonic-constant: even load balancing, ignoring the distance.



(b) convex-constant: targeted load, ignoring the distance.



(c) constant-pow8: shortest distance, ignoring the server load.



(d) convex-pow8: targeted server load with distance threshold.

Fig. 4: Simulation snapshots with 4 combinations of cost functions: each snapshot consists of the drone assignment map, the server load panel and the drone distance panel.

Regarding the distance distributions in Fig.5c, the green-colored *constant* in each group ignores the distance, leading to random drone assignments. In the *convex* and *monotonic* cost groups, the median distance of *pow8* is larger than *pow1* but its upper tail of the distribution is shorter. This is because, with *pow8*, the distant cost difference is smaller when under the threshold but the penalty becomes much larger when exceeding the threshold. Note that *convex* with *pow8* has a shorter tail than *constant* with *pow8* that ignores the server load, for the reason that the latter often fills up the closest edge and needs to overflow to the potentially-distant next edge while the former never fills up edges and always has allocatable space at the closest edge.

We can observe the interaction between the server load constraint and the distance constraint; when making a compromise with the two constraints, the penalty function plays a role in balancing the degree of compromises. For the distance constraint, *pow8* imposes a stronger penalty than *pow1*, and thus, *pow8* has a shorter tail for the distance in exchange for the slightly longer tail for the server load.

The simulation results show that with the *convex* server function and the polynomial distance function, both server load and distance are mostly maintained within the target range. In addition, it is possible to prioritize one of the soft constraints by using a stronger penalty function.

C. Varying Load

So far, the total load is fixed to 0.4. In order to illustrate the interactions between the computational and communication constraints under different total load, we compare the tail of the distribution with the 95th-percentile (p95), by varying the overall server load at 0.2, 0.4 and 0.6 in Fig.6,

In Fig.6a, we compare the p95 load of non-idle servers against the p95 distance, with 5 major combinations of the server load functions and the distance functions. The soft constraint for the distance on the X-axis is the threshold 150 (blue area in the figure), and that for the server load on the Y-axis is the working load range $[0.5, 0.75]$ (yellow area in the figure), with the overlapping area representing the target area (green area in the figure).

The *convex* with *constant* is a baseline that ignores the distance, and thus, plotted on the far right. Similarly, the *constant* with *pow8* is another baseline that ignores the server load. Again, our focus is the *convex* with *pow8*, and it is compared with the *convex* with *pow1*.

In Fig.6a, the p95 load increases with the increased overall load, and the *convex* with *pow8* stays within the target area with load 0.2, but slightly above the area with load 0.4 and 0.6. Compared with the *convex* with *pow1*, the p95 distance is smaller with the *convex* with *pow8* because the *pow8* works

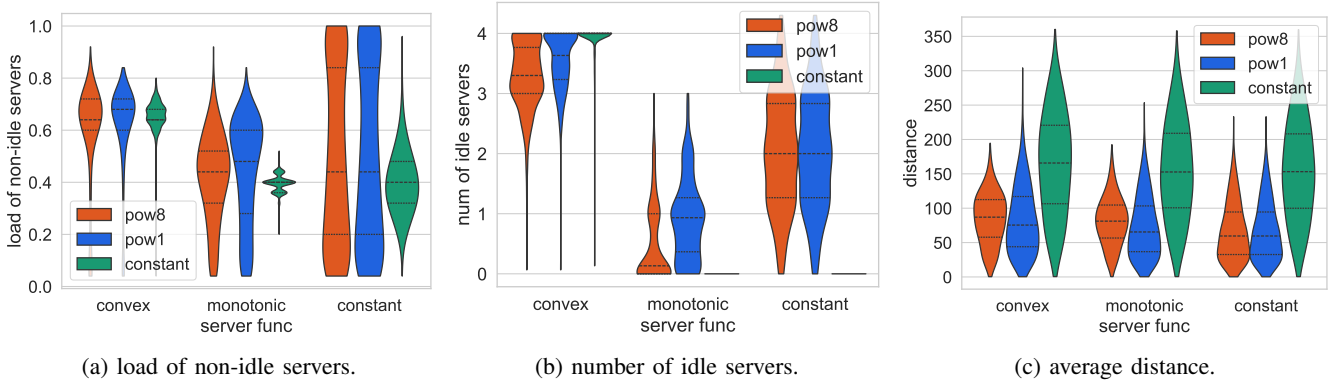
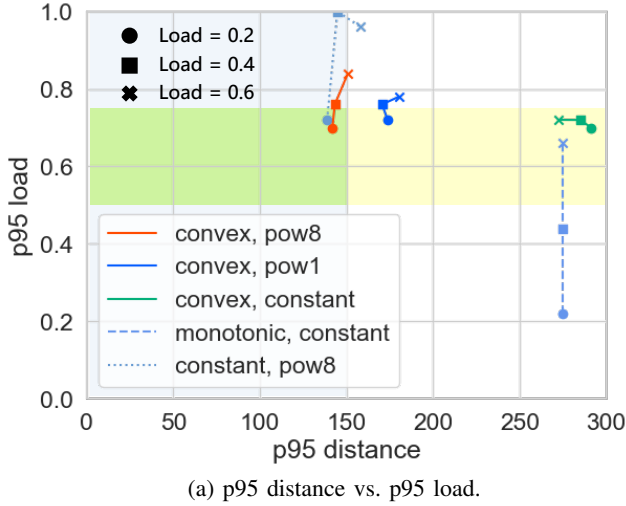
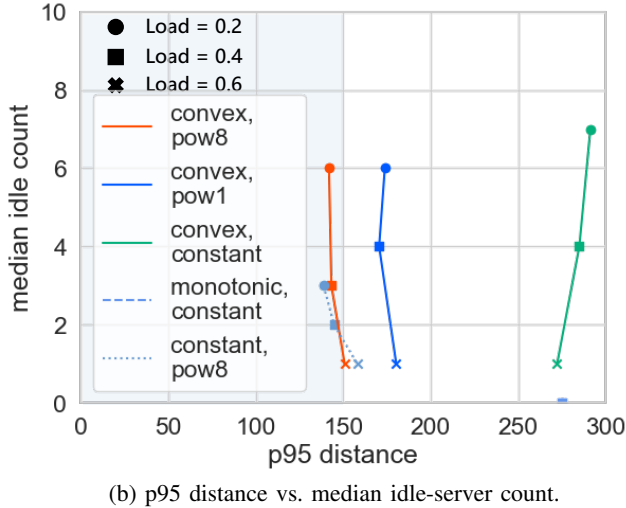


Fig. 5: Distributions of 3 metrics over the entire duration; 3 distant cost functions in 3 server cost function groups.



(a) p95 distance vs. p95 load.



(b) p95 distance vs. median idle-server count.

Fig. 6: Comparing the tail of distributions with varying load.

as a stronger distance penalty, as we observed in Fig.5c. We can confirm in Fig.6b that the number of idle servers decreases as the total load increases.

D. Summary

In the simulation, we use a specific scenario with a convex function for energy saving and a polynomial function for the distant constraint. However, in general, we can use a convex function for a soft constraint aiming at a specific target value, and a monotonic polynomial function for a threshold-based soft constraint.

Our pseudo cost model allows multiple hard and soft constraints embedded into a pseudo cost function, and enables automatically balancing different constraints in a distributed manner. Although there are interactions among soft constraints, different constraints are automatically balanced with appropriate penalty functions. If fine-tuning is needed, a stronger or weaker penalty function can be used to prioritize a certain soft constraint.

V. CONCLUSION

Edge computing in the near future would utilize flexible micro-services, leveraging diverse and geographically scattered computing resources. A possible solution for optimizing resource allocation in edge is a simple cost-based allocation model.

In this paper, we have identified the roles of hard constraints and soft constraints in the cost functions in the resource allocation model, and showed that soft constraints could interfere with each other. To illustrate the trade-offs in balancing computational and communication constraints, we have conducted simulations with the cost functions with specific scenarios. The simulation results showed that the combination of a convex function with a polynomial penalty function works well for balancing different requirements such as energy saving and the communication distance constraint.

Future work includes exploring more realistic simulation scenarios, considering communication loads with the user and required data, with different soft constraints. We are also interested in investigating methods for edge cloud operators to specify conflicting constraints in an intuitive manner.

ACKNOWLEDGMENTS

This work was partially supported by JSPS KAKENHI Grant Number JP21H04872 and JP24H00691.

REFERENCES

- [1] K. Cho and J. Baffier, "An Autonomous Resource Management Model towards Cloud Morphing," in *ACM EdgeSys'23*, 2023, p. 7–12. [Online]. Available: <https://doi.org/10.1145/3578354.3592864>
- [2] J. MacKie-Mason and H. Varian, *Pricing the Internet*. Cambridge, MA, USA: MIT Press, 1995, p. 269–314.
- [3] T. Henderson, J. Crowcroft, and S. Bhatti, "Congestion pricing. paying your way in communication networks," *IEEE Internet Computing*, vol. 5, no. 5, pp. 85–89, 2001.
- [4] K. Lange, "Penalty and barrier methods," in *Optimization*. Springer, 2013, ch. 13, pp. 313–339. [Online]. Available: https://doi.org/10.1007/978-1-4614-5838-8_13
- [5] A. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *European Control Conference*, 2019, pp. 3420–3431.
- [6] J. Xu, B. Palanisamy, H. Ludwig, and Q. Wang, "Zenith: Utility-aware resource allocation for edge computing," in *IEEE EDGE'17*, 2017, pp. 47–54.
- [7] M. Alicherry and T. Lakshman, "Network aware resource allocation in distributed clouds," in *INFOCOM*, 2012, pp. 963–971.
- [8] L. Suresh, P. Bodik, I. Menache, M. Canini, and F. Ciucu, "Distributed resource management across process boundaries," in *ACM SoCC*, 2017, p. 611–623. [Online]. Available: <https://doi.org/10.1145/3127479.3132020>
- [9] F. Kelly, A. Maulloo, and D. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research Society*, vol. 49, 02 1998.
- [10] S. Sen, C. Joe-Wong, S. Ha, and M. Chiang, "A survey of smart data pricing: Past proposals, current plans, and future trends," *ACM Comp. Surv.*, vol. 46, no. 2, nov 2013. [Online]. Available: <https://doi.org/10.1145/2543581.2543582>
- [11] R. Gibbens and F. Kelly, "Resource pricing and the evolution of congestion control," *Automatica*, vol. 35, no. 12, pp. 1969–1985, 1999.
- [12] B. Briscoe, V. Darlagiannis, O. Heckman, H. Oliver, V. Siris, D. Songhurst, and B. Stiller, "A Market Managed Multi-Service Internet (M3I)," *Comput. Commun.*, vol. 26, no. 4, p. 404–414, mar 2003. [Online]. Available: [https://doi.org/10.1016/S0140-3664\(02\)00158-5](https://doi.org/10.1016/S0140-3664(02)00158-5)
- [13] S. El-Zahr, P. Gunning, and N. Zilberman, "Exploring the benefits of carbon-aware routing," *Proc. ACM Netw.*, vol. 1, no. CoNEXT3, Nov. 2023. [Online]. Available: <https://doi.org/10.1145/3629165>
- [14] R. Jacob and L. Vanbever, "The Internet of Tomorrow Must Sleep More and Grow Old," *SIGENERGY Energy Inform. Rev.*, vol. 3, no. 3, p. 27–32, Oct. 2023. [Online]. Available: <https://doi.org/10.1145/3630614.3630620>
- [15] N. Zilberman, E. M. Schooler, U. Cummings, R. Manohar, D. Nafus, R. Soulé, and R. Taylor, "Toward carbon-aware networking," *SIGENERGY Energy Inform. Rev.*, vol. 3, no. 3, p. 15–20, Oct. 2023. [Online]. Available: <https://doi.org/10.1145/3630614.3630618>
- [16] J. Murphy, L. Murphy, and E. Posner, "Distributed pricing for embedded atm networks," in *The Fundamental Role of Teletraffic in the Evolution of Telecommunications Networks*. Elsevier, 1994, vol. 1, pp. 1053–1063. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780444820310501086>
- [17] S. Wagner, E. Berg, J. Giacopelli, A. Ghetie, J. Burns, M. Tauil, S. Sen, M. Wang, M. Chiang, T. Lan, R. Laddaga, P. Robertson, and P. Manghwani, "Autonomous, Collaborative Control for Resilient Cyber Defense (ACCORD)," in *SASOW*, 2012.
- [18] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *ACM SIGGRAPH '87*, 1987, p. 25–34. [Online]. Available: <https://doi.org/10.1145/37401.37406>