An Autonomous Resource Management Model towards Cloud Morphing

Kenjiro Cho
IIJ Research Laboratory

Jean-François Baffier
IIJ Research Laboratory

ABSTRACT

This paper proposes an autonomous resource management model for future edge clouds with abundant and diverse computing resources. The model utilizes a pseudo cost function to allocate resources based on load, and a convex cost function to enable load balancing and idle-resource pooling. Stakeholders can manipulate the cost function or resource weights for micro-jobs to control resource utilization. We explore various cost tuning methods and present their performance through simulations.

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**; • **Networks** → *Network resources allocation.*

KEYWORDS

edge computing, autonomous resource management, convex cost function, cloud morphing

ACM Reference Format:
Kenjiro Cho and Jean-François Baffier. 2023. An Autonomous Resource Management Model towards Cloud Morphing. In *6th International Workshop on Edge Systems, Analytics and Networking (EdgeSys '23), May 8, 2023, Rome, Italy*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3578354.3592864

1 INTRODUCTION

A possible future direction of edge computing is distributed cloud computing, leveraging diverse and geographically scattered computing resources at the edge as part of the cloud. While most edge devices today are designed for specific purposes due to their limited resources, future environments may have abundant computing resources through utilization of various available resources. For example, idle computing resources at home could be exploited for a cloud, similar to how in-house solar power systems are connected to an electrical grid. Once there are enough edge resources, they can be used for general purposes without tight management.

We believe that microservices [11] will be a driving force behind the exploitation of smaller computing resources. With microservices, a cloud service is composed of a collection of loosely-coupled lightweight services; each microservice is ephemeral and short-lived. This enables the use of much smaller resource units than virtual machines or containers, facilitating flexible and efficient use of underlying cloud resources, including smaller edge resources. Furthermore, decomposing current long-lived web services into microservices could have an impact similar to packet switching that decomposes communication channels into packets.

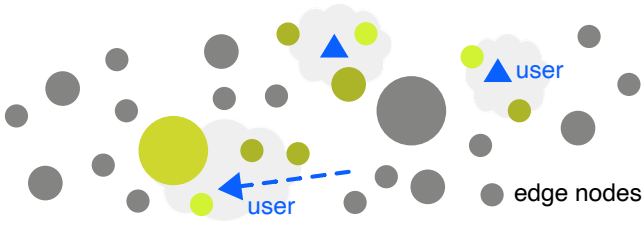

Figure 1: Cloud morphing: the shape of a cloud dynamically follows the usage pattern

In an edge cloud, local edge nodes should handle local requests without relying on a central controller. As a result, cloud management must also shift from current central control to a distributed autonomous model, necessitating a fundamental change to the cloud management model.

Cloud Morphing is our vision for future edge clouds, where a cloud service instance adapts to usage patterns, transforming the locations of the resources and their connections by dynamically allocating microservices across distributed and heterogeneous resources (Figure 1). Microservice jobs are assigned to minimize the overall execution cost that includes computing, communication with the user, and database access. For instance, interactive tasks follow the user's location while data-intensive tasks remain close to the data, regardless of user location. Edge computing is formed automatically by allocating resources close to the users. Furthermore, services are inherently fault-tolerant and resilient against outages or disasters since faulty resources are automatically evicted from the resource pool.

A larger pool of small resources simplifies physical resource management from an operational perspective. Physical nodes can be easily attached to or detached from the resource pool. When there is a consistent hotspot, new resources can be placed close to the hotspot and attached to the resource pool at a convenient time.

Diverse resources would be owned and managed by different parties. This requires loose management of resources as small parties cannot afford dedicated skilled operators. The utilization of each resource must be easily manipulable without compromising system stability. Furthermore, energy savings could be crucial for edge resources.

Realizing such systems requires technical advances across many fields. In particular, a new autonomous resource management model

EdgeSys '23, May 8, 2023, Rome, Italy

ACM ISBN 979-8-4007-0082-8/23/05…$15.00
https://doi.org/10.1145/3578354.3592864

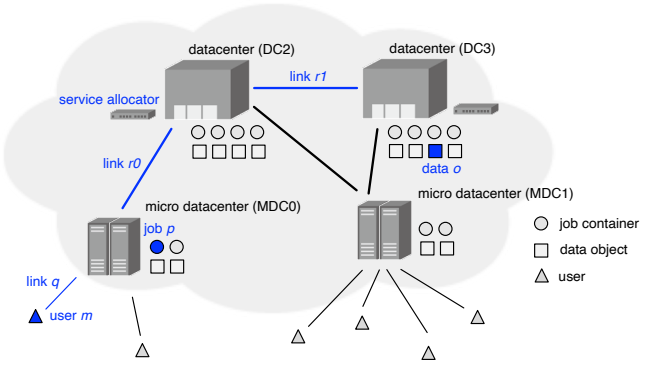

Figure 2: Simple system model for job assignment

is needed for distributed diverse resources, which is the focus of this paper.

We utilize dynamic pricing for decentralized resource allocation suitable for microservices, which acts as backpressure against congestion. The system is designed to avoid serious congestion, provided that some resources remain available in the resource pool. This mechanism also reserves a sufficient margin essential for the performance of statistically multiplexing services. To manipulate resource allocation, we use pseudo cost, which should not be confused with service fees for customers. Pseudo cost is not an actual monetary charge; rather, it is used to control resource utilization.

Our paper draws on the extensive literature that formulates resource allocation as optimization problems. As such, we utilize terminology and expressions from optimization theory. However, our aim is not to pursue theoretical optimum allocations, but rather to present a practical feedback control model for decentralized resource management.

2 RESOURCE MANAGEMENT MODEL

Our resource management model can be explained through a simplified system configuration consisting of two datacenters (DCs) and two micro-datacenters (MDCs) [6], as depicted in Figure 2. When a user initiates a service request to a nearby service allocation server, the server creates a series of microservice 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 resource agents for available resources and their current costs. The job is then allocated to the node that minimizes the overall cost for the service. The area for cost inquiry could be within proximity along the path between the user and the data object. The load of a resource can be measured in various ways, and most resources have built-in load report functions. If the load of a resource fluctuates too fast, a smoothing filter (e.g., EWMA) should be used to stabilize the behavior. A precise load value is not necessary and a rough estimate suffices for loose resource management. Still, if the allocation is not sufficiently small in both size and duration, the load may not converge as expected. Therefore, microservice is a key enabler for our approach.

2.1 Micro-job Assignment

In this paper, we employ a simple micro-job model that defines the required resources for a micro-job as $J(p, q, r, s)$. Here, p denotes the number of micro containers for computation, q denotes frontend communication with the user, r denotes backend communication with data objects (such as databases), and s is the number of time slots. The communication costs 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 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.

To instantiate a micro-job requested by a user, the service server identifies the optimal node to allocate the required resources for J: p, q, and r for a duration s. The pseudo cost E to host micro-job j for a unit of time at node i for user m and data object o is calculated as:

$$E(j, i) = H(j, i) + G(j, i, m, o)$$

here, $H(j, i)$ represents the computing cost to run j at i, and $G(j, i, m, o)$ denotes the communication cost to run j at i between m and o.

$$H(j, i) = p \cdot f(\rho_i)$$
$$G(j, i, m, o) = q \cdot \sum_{l \in path(m,i)} f(\rho_l) + r \cdot \sum_{l \in path(i,o)} f(\rho_l)$$

$f(\rho)$ is the cost function of a resource load ρ, and $path(m, i)$ is a set of links from m to i (e.g., the shortest path weighted by cost). To assign micro-job j, the server selects the node that minimizes the cost, as expressed by:

$$argmin_i \; E(j, i)$$

2.2 Pseudo Cost Functions

Our model employs a parametric representation of pseudo cost, as a function of load, and uses it for load control and also as backpressure against congestion. Micro-jobs are naturally gravitated to the most cost-efficient location.

The proposed model is based on congestion pricing in which the cost of a resource dynamically changes according to the load of the resource. It works as a barrier function for optimization; the capacity constraint is enforced by a penalizing cost when approaching the full capacity.

Another key idea is **idle-resource pooling** that tries to put resources into an idle state for energy saving. We propose a convex cost function that enables idle-resource pooling as part of the congestion pricing mechanism.

A pseudo cost function in our model maps the load of a resource $\rho \in [0, 1]$ to the corresponding cost. The capacity limit is enforced by the cost function that rapidly grows as the load approaches 1.0, which is known as a barrier function in optimization theory.

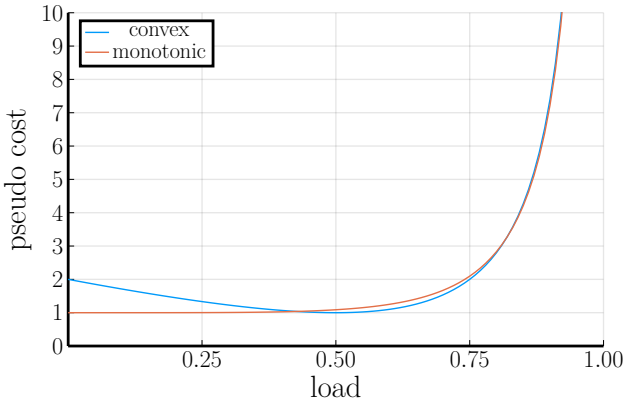

Figure 3: Standard cost functions

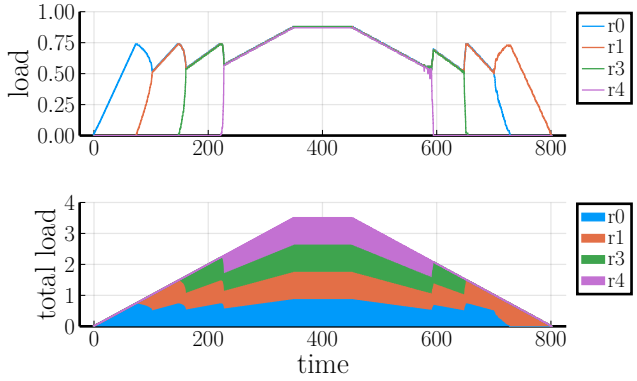

Figure 4: Load distribution among 4 equivalent cost nodes: the load of each resource (top) and the total load (bottom)

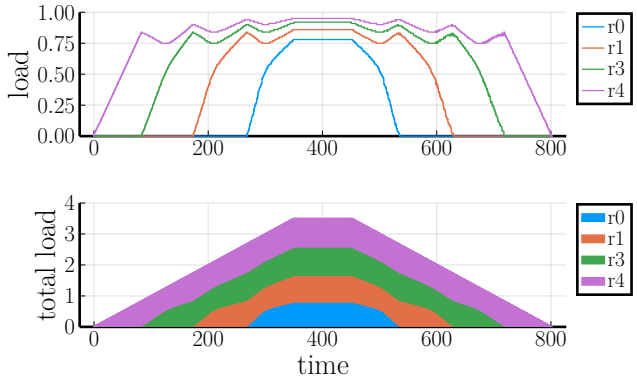

Figure 5: Load distribution: 4 proportional cost nodes

We use two types of pseudo cost functions: one is the monotonic cost function and the other is the convex cost function. The monotonic cost function is a simple barrier function that monotonically grows with load, up to infinity as $\rho \rightarrow 1$. The convex cost function is also a barrier function but also for idle-resource pooling. We use the convex form for computing but the monotonic form for network links as energy-saving-by-idling is not common for network links.

The standard forms that have the minimum cost of 1.0 are shown as bold lines in Figure 3. We will show how to manipulate the cost functions in Sec. 2.4.

The standard convex cost function is defined as:

$$f(\rho) = \frac{(2\rho - 1)^2}{1 - \rho} + 1$$

This function has the properties: $min\ f(\rho) = f(.5) = 1$, and $f(0) = f(.75) = 2$. The cost grows rapidly when $\rho > .75$. The system automatically tries to keep $\rho \leq .75$, aiming at $\rho = .5$. The standard monotonic cost function is defined as:

$$f(\rho) = \frac{\rho^{4.5}}{1 - \rho} + 1$$

to roughly match the standard convex cost function in $[.5, .75]$, the *working load range* explained in the next subsection. Note that the standard forms are defined for convenience, and other functions with a similar shape also work for our purposes.

2.3 Idle-Resource Pooling in Action

The behavior of idle-resource pooling by the convex cost function is illustrated by the following example in Figure 4.

First, assume a pool of 4 equivalent resources with the standard convex cost function. Also, assume that micro-jobs are continuously assigned to the system; each micro-job is much smaller than the capacity of a resource. The initial system load $\sum \rho$ is 0, and gradually increased up to 3.5 until time 350. After time 450, the system load is gradually decreased back to 0 until time 800. Here, load 1.0 is the capacity of a single resource. Initially, all resources in the pool are idle, and their costs are all $f(0) = 2$. For the first job, one resource r_0 is randomly selected for allocation, and its cost becomes lower: $f(0+) < 2$. As a result, subsequent jobs are assigned to r_0, with lowering cost towards $\rho = .5$ and then rising again until $\rho = .75$ where $f(.75) = 2 = f(0)$. At this point, another resource r_1 is selected for allocation. r_1 is preferred over r_0 as its cost becomes lower with new allocations so that both loads move towards $\rho_0 = \rho_1 = .5$, where both are balanced. Both loads rise again until $\rho_{r0} = \rho_{r1} = .75$, where the third resource r_2 kicks in. It repeats for r_3, but no more idle resource is available when $\sum \rho$ reaches 3.0 so that the loads grow beyond .75 up to $\rho = .875$ and $\sum \rho = 3.5$.

When the system load decreases, the process is reversed. After reaching $\rho = .5$ for all, one resource with the lowest load $\rho < .5$ becomes more expensive than the others. This one is less preferred for subsequent assignments, and quickly loses the load until it becomes idle again, while the other 3 keep ρ in $[.5, .75]$. It repeats for the remaining ones.

It is easy to see how the number of active resources changes when there are more resources. When the number of active resources is increasing, the load of each active resource stays at around $\rho = .75$. On the other hand, when the number of active resources is decreasing, the load of each active resource stays at around $\rho = .5$. In short, when all resources are equal, the system tries to maintain the load of active resources in the *working load range* $[.5, .75]$, while keeping idle resources as much as possible.

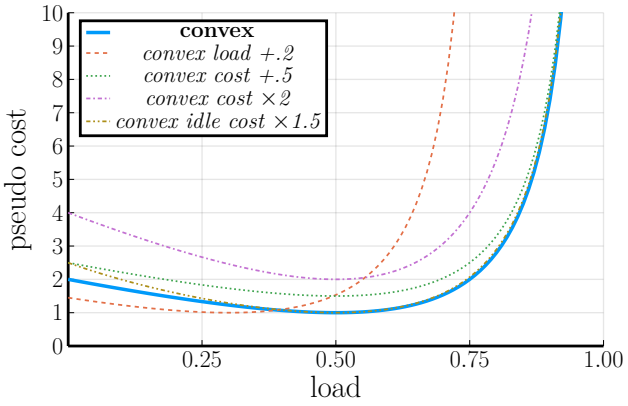

Figure 6: Manipulating convex cost functions

The usage of a resource can be controlled by manipulating the cost function. Let's change the costs of the 4 resources with the ratio $1:2:4:8$, that is $8f_{r0} = 4f_{r1} = 2f_{r2} = f_{r3}$. Figure 5 shows the load distribution. Here, we focus on the interaction between r_0 and r_1 since the other interactions are similar. To activate r_1, the load of r_0 goes up to .84 to satisfy: $f_{r1}(0) = 4 = f_{r0}(.84)$. When r_1 is moving towards idle after time 700, the load of r_0 is .75 to satisfy: $f_{r1}(.5) = 2 = f_{r0}(.75)$

For unequal resources in general, the required load to trigger a new allocation is higher than $\rho = .75$ for the already active ones to match the cost $f(0)$ for the new one, but the load will not go much further as the slope of the cost function is steep. Similarly, when the most expensive one among active resources becomes idle, the load of the remaining ones stay at the matching cost $f(.5)$ for the deactivating one.

2.4 Manipulating the Cost Function

The utilization of a resource can be manipulated by modifying the cost function of the resource as shown as the variants in Figure 6. One can *lower or raise the load level* of a resource by shifting the load in the cost function and adjusting the target load, $f'(\rho) = f(\rho + \Delta)$. To *change the activation order* in the idle-resource pooling, one can raise or lower the cost, e.g., by making the cost n times more expensive, $f'(\rho) = nf(\rho)$. For minor adjustment, one can use an additive form, $f'(\rho) = f(\rho) + \Delta$. To make *idle-resource pooling more aggressive*, one can raise the cost at $\rho = 0$: e.g., to raise the cost at $\rho = 0$ by a factor of $(n+1)/2$, $f'(\rho) = n(2\rho - 1)^2/(1 - \rho^n) + 1$.

A *premium service* can be realized by shifting the load in the same way as lowering the load level but for specific users or jobs (not for a specific resource) so as to have premium jobs always being assigned to less loaded resources. Similarly, an *economy service* that allows to be assigned to more loaded resources can be made by a negative shift. It would be appealing as a business model to enable multiple classes using a single resource pool with a single shift parameter.

Other than manipulating the cost function, cloud service providers can adjust the required resources and their weights for a job. Also, it is effective to place data objects close to the users, and both service providers and their users should have some control over where to store the data. Thus, the system allows stakeholders to loosely control the resource utilization.

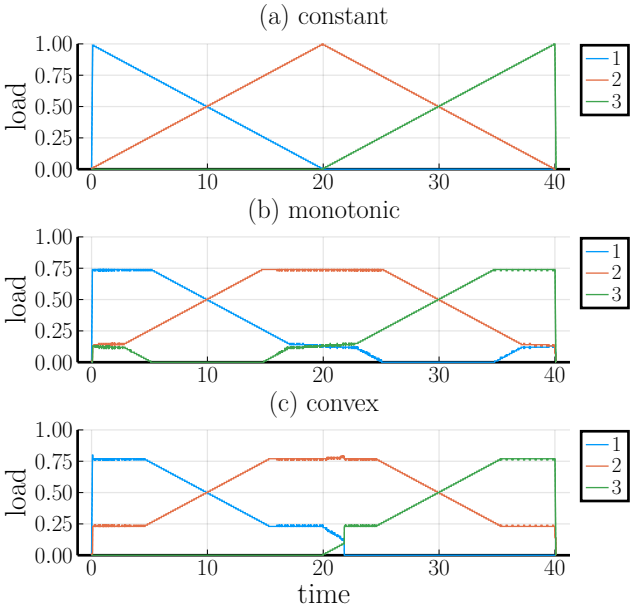

Figure 7: Comparing (a) constant, (b) monotonic and (c) convex behaviors with a flock of drone scenario

3 SIMULATION RESULTS

We have developed a simple simulation tool to evaluate the model that is publicly available[1]. We omit the details of the simulation settings in this paper but all the settings are described in the simulation tool.

The first scenario, depicted in Figure 7, involves a group of drones that move cyclically through a network of interconnected nodes. The drones move through the cycle, gradually transitioning from one node to the next. The total load is kept constant (1.0) with interactive jobs. The different cost models are used for the nodes to illustrate their behaviors while the same monotonic cost is used for the links. In the baseline case with the constant costs, each drone is always assigned to the nearest node. In the other cases, the load of each node does not go up beyond .75, with the difference in the assignment of the remaining load. The monotonic costs act so as to make the remaining load equally shared, while the convex costs act for shifting the remaining load to one node. At time 20 in (c), the remaining load starts to shift from node 1 to node 3 and, once it hits the balancing point, all the remaining load moves to node 3. This behavior of convex costs leads to the stability in case of turbulence.

The effects of manipulating the cost function and the resource weight for a job is illustrated in the second scenario in Figure 8 using the topology in Figure 2. The capacities of MDC and DC are set to 100 and 1,000 respectively. Note that the capacity of DCs is 10 times larger than that of MDCs so that DC's load in the top plot looks

[1] https://github.com/iijlab/KuMo.jl

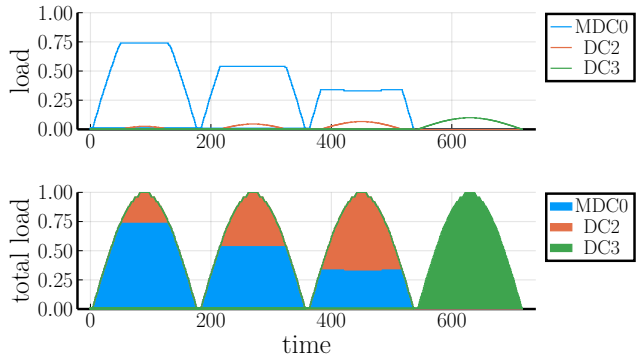

Figure 8: Cost manipulations: shifting the load +.2, +.4 and raising the weight for data access

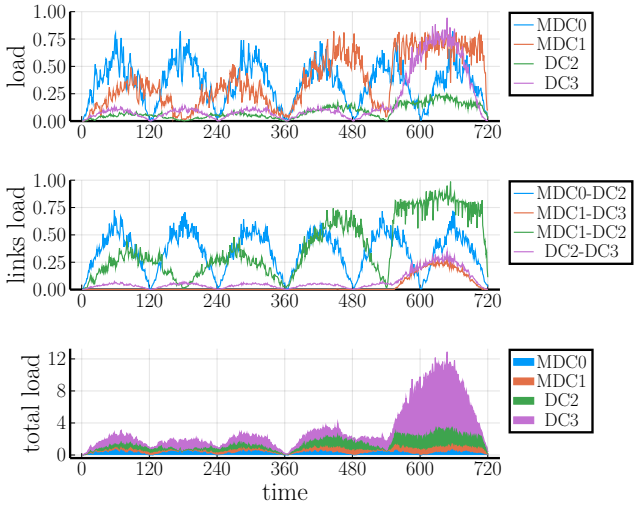

Figure 9: Mixed load with 2 DCs and 2 MDCs

much smaller for the same volume of jobs. In the normalized total load in the bottom plot, the volume is normalized to the capacity C of MDC to show the total volume of jobs: $\sum \tilde{\rho},\ \tilde{\rho} = \rho \cdot C/C_{MDC}$.

Here, a series of interactive jobs of the same type are generated in 4 waves between a user at MDC0 and an object at DC3. In the first wave, most of the jobs are assigned to the node closest to the user (MDC0) but some overflowing jobs are offloaded to DC2, the upstream of MDC0. The peak load of MDC0 is .74 in the first wave. In the second and the third waves, the cost function of MDC0 is manipulated to reduce the load by shifting the load by .2 and .4 respectively. As a result, the peak load of MDC0 is reduced from .74 to .54 in the second wave, and then, to .33 in the third wave. In the fourth wave, the weight for the backend communication is raised, and all jobs are assigned to DC3 that has the object.

A more complex scenario is shown in Figure 9. Again, the topology in Figure 2 is used, and random fluctuation is added to the interval and duration of jobs. A series of jobs are generated between user0 at MDC0 and an object at DC3, and between user1 at MDC1 and an object at DC2. Both have the ratio of 1 : 2 for interactive vs. data-intensive jobs. To observe offloading behaviors, user1's jobs are increased by a factor of 2 in the third wave (time 360-540), and by a factor of 10 in the fourth wave (time 540-720). Before time 360, interactive jobs are assigned closer to the users, and data intensive jobs are assigned closer to the data. In the third wave, MDC1 reaches the upper limit and overflowing jobs are assigned to DC2. In the fourth wave, the link MDC1-DC2 reaches the upper limit so that overflowing jobs are assigned to DC3. (To absorb the surge in the fourth wave in this scenario, enough capacity is provided to the link MDC1-DC3.) This scenario illustrates the responsive behavior of the system, and how jobs for MDCs can be offloaded to upstream DCs.

Overall, the simulation results illustrate the behaviors of the idle-resource pooling, quick responses to changing load by offloading excess jobs to upstream DCs, and converging to stable states once the load is settled while maintaining the load for each active resource at around the target load.

4 RELATED WORK

Resource allocation for edge computing and micro data centers has a rich collection of work; most approaches are based on some form of optimization and many employ auction models. The topics are ranging from VM auctions in IaaS [18, 19], load balancing within a data center [4, 12], to distributed resource management [2] for microservices [15]. Xu *et al.* [17] proposed an auction model for the edge computing infrastructure layer that is similar to our system model in separating the infrastructure layer.

Congestion pricing and smart data pricing for networking have been extensively studied [13]. Early work includes congestion pricing by Murphy *et al.* [10], and auction-based resource allocation by MacKie-Mason *et al.* [9], both in 1994. Kelly *et al.* [7] framed congestion pricing in an optimization framework for fair allocation, which inspired a large number of the following work [3, 5, 13]. Congestion pricing are also applied to cloud computing [8, 14].

Exponential cost growth as a function of load is well known in packet switching networks (e.g., M/M/1 queue and CSMA/CD Ethernet). The idea to use such cost functions for resource management was in [10] where Murphy *et al.* used a cost function for distributed bandwidth allocation in ATM networks in 1994. Their cost function is a barrier function for utility optimization and their simple cost minimizing allocation algorithm is also somewhat similar to our allocation model.

A system model similar to ours is found in [16] where Wagner *et al.* used congestion pricing for resilient job allocation in a distributed military cloud. They used a game-theoretic resource allocation method based on Nash Bargaining, and developed a Hadoop-based prototype system.

We were inspired by the concept of microservices, and have applied packet switching techniques including congestion pricing to distributed heterogeneous cloud resource management. To the best of our knowledge, our model is unique in exploiting microservices and using a convex cost function not just for a barrier function but also for idle-resource pooling.

5 SUMMARY AND FUTURE WORK

We have presented the cloud morphing vision for future distributed cloud services, proposed a resource allocation model based on cost functions, and presented how the idle-resource pooling works. We are planning to refine the proposed model and develop a working prototype system.

For the model refinement, the current model is simplistic so that we will add bidirectional communication costs, multiple users per job, interactions among jobs, and other features. We need deeper studies on cost functions to understand interactions among different cost functions with load-balancing and idle-resource pooling, and how to better control the overall system behavior. Also, we did not consider dependency among microservice jobs, but it would be necessary to investigate the impact of the interplay of microservice jobs [15].

For the prototype development, we need realistic future microservice workload models. Also, it is necessary to develop mechanisms and protocols for discovering available resources [1] and exchanging cost information. A path selection mechanism is also needed when assigning a job, probably using a source routing mechanism.

There exists a rich area for further research: one topic is *hierarchical configurations*. A hierarchical system model would be preferred for scalability and for management purposes. For example, a distributed datacenter model can consist of the inter-DC layer with DC-level resources and the intra-DC layer with rack-level resources. The hierarchical model naturally extends to an *inter-cloud model* in which different cloud systems are federated. The advantage is that the idle-resource pooling mechanism allows utilizing external clouds only when needed.

Further, it would be possible to *crowdsource the resource supply* at the edge. Then, we need a monetary charging, authentication and authorization mechanisms to add externally provided resources to the resource pool.

The location of data is critical for the pseudo cost so that *data placement* would play an important role for cloud morphing. A possible approach is to migrate data at a coarser timescale, based on usage history. Another possibility is to design a new distributed storage system suitable for the cloud morphing model in which (cached) data can be easily moved or replicated.

The idle-resource pooling is not used for network links in this paper, but it is possible to use it for dynamically establishing *Layer-2 paths* (e.g., lightpath switching over WDM networks).

We believe that future distributed heterogeneous clouds need a new paradigm for resource management, and hope this work will stimulate other research in the field.

REFERENCES

[1] Jeannie Albrecht, David Oppenheimer, Amin Vahdat, and David A. Patterson. 2008. Design and Implementation Trade-Offs for Wide-Area Resource Discovery. *ACM Trans. Internet Technol.* 8, 4, Article 18 (oct 2008). https://doi.org/10.1145/1391949.1391952

[2] Mansoor Alicherry and T.V. Lakshman. 2012. Network aware resource allocation in distributed clouds. In *INFOCOM*. 963–971. https://doi.org/10.1109/INFCOM.2012.6195847

[3] Bob Briscoe, Vasilios Darlagiannis, Oliver Heckman, Huw Oliver, Vasilios Siris, David Songhurst, and Burkhard Stiller. 2003. A Market Managed Multi-Service Internet (M3I). *Comput. Commun.* 26, 4 (mar 2003), 404–414. https://doi.org/10.1016/S0140-3664(02)00158-5

[4] Liuhua Chen, Haiying Shen, and Karan Sapra. 2014. Distributed Autonomous Virtual Resource Management in Datacenters Using Finite-Markov Decision Process. In *SOCC*. ACM. https://doi.org/10.1145/2670979.2671003

[5] Richard J Gibbens and Frank P Kelly. 1999. Resource pricing and the evolution of congestion control. *Automatica* 35, 12 (1999), 1969–1985.

[6] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. 2009. The Cost of a Cloud: Research Problems in Data Center Networks. *SIGCOMM CCR* 39, 1 (dec 2009), 68–73. https://doi.org/10.1145/1496091.1496103

[7] F.P. Kelly, A.K. Maulloo, and D. Tan. 1998. Rate Control for Communication Networks:Shadow Prices, Proportional Fairness and Stability. *Journal of the Operational Research Society* 49 (02 1998). https://doi.org/10.1057/palgrave.jors.2600523

[8] Cinar Kilcioglu and Costis Maglaras. 2015. Revenue Maximization for Cloud Computing Services. *SIGMETRICS Perform. Eval. Rev.* 43, 3 (nov 2015), 76. https://doi.org/10.1145/2847220.2847245

[9] Jeffrey K. MacKie-Mason and Hal R. Varian. 1994. *Pricing the Internet.* Computational Economics 9401002. University Library of Munich, Germany. https://ideas.repec.org/p/wpa/wuwpco/9401002.html

[10] J. Murphy, L. Murphy, and E.C. Posner. 1994. Distributed Pricing For Embedded ATM Networks. In *The Fundamental Role of Teletraffic in the Evolution of Telecommunications Networks*. Teletraffic Science and Engineering, Vol. 1. Elsevier, 1053–1063. https://doi.org/10.1016/B978-0-444-82031-0.50108-6

[11] Irakli Nadareishvili, Ronnie Mitra, Matt McLarty, and Mike Amundsen. 2016. *Microservice architecture: aligning principles, practices, and culture.* O'Reilly Media, Inc.

[12] Negar Rikhtegar, Manijeh Keshtgari, Omid Bushehrian, and Guy Pujolle. 2021. BiTE: a dynamic bi-level traffic engineering model for load balancing and energy efficiency in data center networks. *Appl. Intell.* 51 (2021), 4623–4648.

[13] Soumya Sen, Carlee Joe-Wong, Sangtae Ha, and Mung Chiang. 2013. A Survey of Smart Data Pricing: Past Proposals, Current Plans, and Future Trends. *ACM Comp. Surv.* 46, 2 (nov 2013). https://doi.org/10.1145/2543581.2543582

[14] Jiayi Song and Roch Guerin. 2017. Pricing and bidding strategies for cloud computing spot instances. In *IEEE INFOCOM WKSHPS*. IEEE, 647–653. https://doi.org/10.1109/INFCOMW.2017.8116453

[15] Lalith Suresh, Peter Bodik, Ishai Menache, Marco Canini, and Florin Ciucu. 2017. Distributed Resource Management across Process Boundaries. In *SoCC*. ACM, 611–623. https://doi.org/10.1145/3127479.3132020

[16] Stuart Wagner, Eric Van Den Berg, Jim Giacopelli, Andrei Ghetie, Jim Burns, Miriam Tauil, Soumya Sen, Michael Wang, Mung Chiang, Tian Lan, Robert Laddaga, Paul Robertson, and Prakash Manghwani. 2012. Autonomous, Collaborative Control for Resilient Cyber Defense (ACCORD). In *SASOW*. https://doi.org/10.1109/SASOW.2012.16

[17] Jinlai Xu, Balaji Palanisamy, Heiko Ludwig, and Qingyang Wang. 2017. Zenith: Utility-Aware Resource Allocation for Edge Computing. In *EDGE*. IEEE, 47–54. https://doi.org/10.1109/IEEE.EDGE.2017.15

[18] Sharrukh Zaman and Daniel Grosu. 2013. A Combinatorial Auction-Based Mechanism for Dynamic VM Provisioning and Allocation in Clouds. *IEEE Transactions on Cloud Computing* 1, 2 (2013), 129–141. https://doi.org/10.1109/TCC.2013.9

[19] Xiaoxi Zhang, Zhiyi Huang, Chuan Wu, Zongpeng Li, and Francis C. M. Lau. 2017. Online Auctions in IaaS Clouds: Welfare and Profit Maximization With Server Costs. *IEEE/ACM Transactions on Networking* 25, 2 (2017), 1034–1047. https://doi.org/10.1109/TNET.2016.2619743