New Distributed Constraint Satisfaction Algorithms for Load Balancing in Edge Computing: A Feasibility Study

Khoi D. Hoang¹, Christabel Wayllace¹, William Yeoh¹, Jacob Beal², Soura Dasgupta³, Yuanqiu Mo³, Aaron Paulos², and Jon Schewe²

¹ Department of Computer Science and Engineering, Washington University in St. Louis {khoi.hoang, cwayllace, wyeoh}@wustl.edu

² Raytheon BBN Technologies

{jake.beal,aaron.paulos,jon.schewe}@raytheon.com

³ Department of Electrical and Computer Engineering, University of Iowa {soura-dasgupta, yuanqiu-mo}@uiowa.edu

Abstract. Edge computing is a paradigm for improving the performance of cloud computing systems by performing data processing at the edge of the network, closer to the users and sources of data. As data processing is traditionally done in large data centers, typically located far from their users, the edge computing paradigm will reduce the communication bottleneck between the user and the location of data processing, thereby improving overall performance. This becomes more important as the number of Internet-of-Things (IoT) devices and other mobile or embedded devices continues to increase. In this paper, we investigate the use of distributed load balancing problem in edge computing problems. Specifically, we (i) provide a mapping of the distributed load balancing problem in edge computing to a DisCSP; (ii) propose two DisCSP algorithms to solve such problems; and (iii) provide some preliminary analysis on a simple topology.

1 Introduction

Cloud computing is unequivocally the backbone of a large fraction of AI systems, where it provides computational functionality and data storage to the ever-growing number of Internet-of-Things (IoT), mobile, and embedded devices. In today's traditional cloud computing architecture, the compute and storage resources are typically housed in data centers that may be managed by different public and private organizations. For example, Amazon's AWS and Microsoft's Azure systems are two examples of popular cloud computing services that are provided by Amazon and Microsoft to the public for a fee.

As the number of IoT and similar devices continue to grow [2], so will the demand for cloud services. This increase in demand will eventually strain the bandwidth limitations to the data centers, thereby resulting in a drop in the quality of service of such services. To alleviate this limitation, researchers have proposed a new paradigm called *edge computing*, whereby the compute and storage resources are migrated from the data centers distant from users to compute resources that are closer to the user devices at the edges of the network. Figure 1 illustrates these two paradigms. Figure 1(a) shows



FIG. 1: Illustration of Cloud & Edge Computing Paradigms. Figure adapted from [9]

the traditional cloud computing paradigm, where the colors of the arrows denote the congestion in the network – green arrows represent uncongested links, yellow arrows represent marginally congested links, and red arrows represent very congested links. Figure 1(b) shows an edge computing paradigm, where services are hosted at resources at nodes labeled with 'S' and requests are routed to those nodes, resulting in decreased network congestion, and thus in increased performance of services. How to manage decisions about dispersal and placement of services in such a paradigm, however, is still an open problem.

In this paper, we model the distributed load balancing problem for edge computing as a distributed constraint satisfaction problem, where agents that control nodes in the network need to coordinate to identify which node should host which services, subject to constraints on the capacity of the nodes and the requirement to satisfy all expected incoming requests. We also propose two algorithms, called *Distributed Constraintbased Diffusion Algorithm* (CDIFF) and *Distributed Routing-based Diffusion Algorithm* (RDIFF), to solve this problem, and show preliminary results on a simple topology.

2 Background: DisCSP

A Distributed Constraint Satisfaction Problem Problem (DisCSP) [15] is a tuple $\langle \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{A}, \alpha \rangle$, where: $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of variables; $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of finite domains (i.e., $x_i \in D_i$); $\mathcal{C} = \{c_1, \ldots, c_e\}$ is a set of constraints, where each constraint c_i is defined over its scope $\mathbf{x}^{c_i} \subseteq \mathcal{X}$ and specifies the satisfying combination of value assignments in its scope; $\mathcal{A} = \{a_1, \ldots, a_p\}$ is a set of agents; and $\alpha : \mathcal{X} \to \mathcal{A}$ is a function that maps each variable to one agent. To ease readability, in the following, we assume that all agents control exactly one variable. Thus, we will use the terms "variable" and "agent" interchangeably and assume that $\alpha(x_i) = a_i$. A solution is a value assignment σ for all the variables of the problem that is consistent with the variables' domains. The goal is to find a solution that satisfies all constraints in the problem.

3 Load Balancing in Edge Computing

We now provide a simplistic description of the load balancing problem in edge computing architectures. For more detailed discussions, we refer readers to the following resources [11, 12, 14]. Assume that the network can be represented as a graph $G = \langle V, E \rangle$, where each vertex $v \in V$ corresponds to a compute node in the network that is able to host services and each edge $e \in E$ indicates that the two nodes connected by that edge can communicate directly with each other. Each node v has an associated capacity cap(v) that indicates the amount of resources it has to host services. Some of the nodes in the graph are data centers, which are default nodes that host these services.

Some of the nodes at the edge of the cloud are connected to pools of IoT devices, referred to as *client pools*. Further, assume that an estimate of the load of service per-client is available, and such loads load(v) are available for each node v that hosts the service. The goal of the problem is to distribute the hosting of services across the compute network in such a way that all loads can be successfully served. Finally, the problem has a secondary objective of minimizing the latency of the service requests where possible (i.e., services should be hosted as close to the client pools as possible).

To model this problem as a DisCSP:

- Each vertex $v \in V$ in the graph maps to an agent/variable $x_i \in \mathcal{X}$.
- The range of capacity [0, cap(v)] of the vertex $v \in V$ maps to the domain D_i of the variable.
- Finally, a constraint $\sum_{v \in V} load(v) \leq \sum_{x_i \in \mathcal{X}} v_i$ is imposed to ensure that the total load can be satisfied by the network, where v_i is the value assignment for variable x_i in the solution.

4 **Proposed Algorithms**

We now discuss our two distributed algorithms to solve this problem – *Distributed Constraint-based Diffusion Algorithm* (CDIFF) and *Distributed Routing-based Diffusion Algorithm* (RDIFF).

CDIFF: This algorithm is inspired by other diffusion-based algorithms in the literature [1, 8, 6]. At a high level, each overloaded agent (i.e., those agents that control nodes where load(v) > cap(v)) identifies to which subset of other agents that it should shed its excess load. Figure 2 illustrates its three phases, where numbers in circles are the current loads of the nodes and red numbers are the capacities. Node F is the overloaded agent, and nodes A, B, and D are possible nodes that can absorb the excess load from A.¹ We now describe the three phases:

• **Phase 1:** Each overloaded agent sends a message to all its neighbors with the amount of excess load it needs to shed as well as a hop count indicator that is initialized to 1.² When an agent receives such a message for the first time, it will propagate the received information to its neighbors after incrementing the hop counter by 1. The agent will then ignore subsequent Phase 1 messages by other neighbors and respond to them after Phase 3. This propagation of information continues until it reaches either nodes with sufficient capacity to accept the excess load or nodes that have received information from a closer overloaded agent. At the end of this phase, the

¹ While the figure illustrates an example with only one overloaded region, our description below generalizes to the case where there are multiple overloaded regions.

² The indicator counts the number of hops a node is from the overloaded agent.



FIG. 2: Illustration of CDIFF Operations. Figure adapted from [9]

agents have built a directed spanning forest, with roots at every overloaded agent.³ In the example in Figure 2, node F is the sole root as it is the only overloaded agent and nodes A, B, and D are the leaves.

- Phase 2: Each leaf agent v of the spanning forest sends a message to its parent with its node ID, its available capacity cap(v) load(v), and the number of hops it is away from its root. When an agent receives this information from all its children, it aggregates the information received so that the sum of available capacities is at most the amount of excess load needed to be shed, preferring nodes with smaller hop counts, and sends the aggregated information to its parent. This process continues until each root (i.e., an overloaded agent) receives the messages from all its children.
- **Phase 3:** Each root agent then sends a message to each of its children indicating the amount of excess load it intends to shed to them and their descendants in the spanning tree. This information gets propagated down to the leaves, which then terminates the algorithm. For example, in Figure 2, node F sheds 5 units of load 2 units to node D and 3 units to node A.

These three phases continue until all overloaded regions successfully shed their loads.

RDIFF: A limitation of CDIFF is that it does not take into account information of where the client pools are located when deciding where the overloaded agents should shed its load. As such, it is not able to optimize the secondary objective of our problem. RDIFF addresses this limitation by shedding not only the excess load of overloaded agents, but as much load as possible to the agents that are of close proximity to the client pools. To do so, the agents operate in the following manner:

- **Phase 1:** Each data center propagates its entire load received from each client pool back towards that client pool by back-tracing the paths the requests took from the client pool to the data center. At the end of this phase, the agents have built a directed graph, where each branch of the graph corresponds to the path requests from a client pool took to get to a data center.⁴
- **Phase 2:** Each client pool will host as much of the load it received as possible, up to its capacity, and sheds its excess load to its parent (the next node along the branch from client pool towards the data center). This process repeats until all of the excess load is hosted. In the worst case where none of the agents along the branch has excess capacity, the data center will host the entire load.

³ If an agent receives this information from more than one neighbor at the same time, it breaks ties by identifiers.

⁴ If there are multiple paths per client pool, we randomly choose one of them.

5 Experimental Results

We investigate both CDIFF and RDIFF against the default strategy of hosting all services at the data center using the same simple topology shown in Figure 2. In our scenario, all clients are in a single client pool that is connected to node A and the data center is at node F. The capacity of all nodes is 8 units. The clients make a total of 1000 service requests that are divided equally across 10

Hops from	Number of requests		
clients at A	CDIFF	RDIFF	Default
0	300	900	0
1	62	0	0
2	76	0	0
3	562	100	1000

TABLE 1: Experimental Results

batches, with each batch starting a minute after the previous batch. Each service request will induce a load of 0.04 units. For CDIFF and RDIFF, we set the thresholded capacity of nodes to be 80% of their actual capacity. A node is a considered overloaded if its predicted load is greater than its thresholded capacity. Finally, we run the algorithms 10 times, once for each batch of service requests.

Table 1 presents our results. For all three algorithms, all 1000 requests were successfully served. With the default strategy, all 1000 requests were served by the data center at node F, which is 3 hops away from the client pool at node A. For CDIFF, we can observe that there is some migration towards the client pool, but a majority of the requests are still served at the data center. Finally, for RDIFF, requests of the first batch were served at the data center but requests of all subsequent batches were served at node A where the clients are located. These preliminary experimental results thus show that both CDIFF and RDIFF are able to diffuse the load away from the data center and RDIFF is more promising in optimizing the secondary objective of the problem, which is to host the services as close to the client pools as possible.

6 Conclusions and Future Work

In this paper, we have empirically evaluated the feasibility of modeling and solving a distributed load balancing problem in edge computing problems, obtaining preliminary results that indicate this is a promising direction. Future work includes generalizing our algorithms to multiple data centers and services as well as evaluating the algorithms on larger networks. We also plan to consider formulating the problem as a *distributed constraint optimization problem* (DCOP) [7, 10, 3] and develop DCOP algorithms that are resilient to dynamic changes [13] as well as proactively take into account anticipated future changes in the system [4, 5].

7 Acknowledgment

This research is supported by Defense Advanced Research Projects Agency (DARPA) contract HR001117C0049. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. This document does not contain technology or technical data controlled under either U.S. International Traffic in Arms Regulation or U.S. Export Administration Regulations. Approved for public release, distribution unlimited (DARPA DISTAR 31172, 4/3/19).

References

- 1. Cybenko, G.: Dynamic load balancing for distributed memory multiprocessors. Journal of Parallel and Distributed Computing 7(2), 279–301 (1989)
- 2. Evans, D.: The internet of things: How the next evolution of the internet is changing everything. CISCO White Paper **1**(2011), 1–11 (2011)
- Fioretto, F., Pontelli, E., Yeoh, W.: Distributed constraint optimization problems and applications: A survey. Journal of Artificial Intelligence Research 61, 623–698 (2018)
- Hoang, K.D., Fioretto, F., Hou, P., Yokoo, M., Yeoh, W., Zivan, R.: Proactive dynamic distributed constraint optimization. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS). pp. 597–605 (2016)
- Hoang, K.D., Hou, P., Fioretto, F., Yeoh, W., Zivan, R., Yokoo, M.: Infinite-horizon proactive dynamic DCOPs. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS). pp. 212–220 (2017)
- Hu, Y., Blake, R.: An optimal dynamic load balancing algorithm. Tech. rep., SCAN-9509056 (1995)
- Modi, P., Shen, W.M., Tambe, M., Yokoo, M.: ADOPT: Asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligence 161(1–2), 149–180 (2005)
- Muthukrishnan, S., Ghosh, B., Schultz, M.H.: First-and second-order diffusive methods for rapid, coarse, distributed load balancing. Theory of Computing Systems 31(4), 331–354 (1998)
- Paulos, A., Dasgupta, S., Beal, J., Mo, Y., Hoang, K., Lyles, J.B., Pal, P., Schantz, R., Schewe, J., Sitaraman, R., Wald, A., Wayllace, C., Yeoh, W.: A framework for self-adaptive dispersal of computing services. In: IEEE Self-Adaptive and Self-Organizing Systems Workshops (June 2019)
- Petcu, A., Faltings, B.: A scalable method for multiagent constraint optimization. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). pp. 1413– 1420 (2005)
- Puthal, D., Obaidat, M.S., Nanda, P., Prasad, M., Mohanty, S.P., Zomaya, A.Y.: Secure and sustainable load balancing of edge data centers in fog computing. IEEE Communications Magazine 56(5), 60–65 (2018)
- Rabinovich, M., Xiao, Z., Aggarwal, A.: Computing on the edge: A platform for replicating internet applications. In: Web Content Caching and Distribution, pp. 57–77. Springer (2004)
- Rust, P., Picard, G., Ramparany, F.: Self-organized and resilient distribution of decisions over dynamic multi-agent systems. In: Proceedings of the International Workshop on Optimization In Multi-Agent Systems (2018)
- 14. Shi, W., Dustdar, S.: The promise of edge computing. Computer 49(5), 78-81 (2016)
- Yokoo, M., Durfee, E., Ishida, T., Kuwabara, K.: Distributed constraint satisfaction for formalizing distributed problem solving. In: Proceedings of the International Conference on Distributed Computing Systems (ICDCS). pp. 614–621 (1992)