Distributed Orchestration in Cloud Data Centers

Bill McCormick*, Hassan Halabian*, Carol J. Fung†
*Canada Research Center, Huawei Technologies, 303 Terry Fox Dr., Kanata, ON, K2K 3J1, Canada
†Computer Science Department, Virginia Commonwealth University, Richmond, Virginia, USA
Emails: *{hassan.halabian,bill.mccormick}@huawei.com, †cfung@vcu.edu

***Abstract*—Orchestration systems in cloud platforms are responsible for creating, managing and assigning the computational and network bandwidth resources to the requesting services. Conventional orchestration approaches in data centers are based on centralized solutions where they are a single point of failure, and a potential performance bottleneck. In this paper, using the notions of Markov approximation method and auction theory, we propose a fully distributed resource management scheme for data centers. The proposed solution takes into account the operational and economic constraints of the services and the servers in the data center and maximizes a global system utility function in a fully distributed manner. Simulation results show the effectiveness of the proposed solution in terms of speed of convergence, accuracy and resource utilization for applicability in next generation cloud systems.**

***Index Terms*—Cloud orchestration, data centers, distributed algorithms, Markov approximation**

I. Introduction

The emergence of new computing services over cloud computing data centers has resulted in transformation of many conventional business services into cloud-based services. Cloud-based services have the advantage of lower cost, agility and flexibility, hardware and location independence, easier performance monitoring and improved security and maintenance. Different business sectors are defining their next generation services into cloud models. In telecommunications, two cloud-based projects are CORD (Central Office Rearchitected as a Data Center) [1] and CRAN (Cloud Radio Access Network) [2] defined for 5G mobile networks.

A cloud system consists of a number of physical computation (CPU, RAM, etc.) and communication resources (network bandwidth) which are shared among the requesting services.

In resource management for cloud orchestration, different objective functions as well as different placement approaches can be considered. The examples of attributes considered in the objective functions are energy consumption, resource utilization and traffic engineering [3].

Conventional resource management architectures in data centers use a central controller which is responsible for receiving requests for resources, assigning resources to the services and implementing the resource assignments in the form of VMs or containers. This approach is used by both commercial cloud systems and open-source platforms such as Kubernetes [4] and OpenStack [5].

In centralized orchestration systems, the orchestrator becomes a potential bottleneck and single point of failure. Another disadvantage of centralized solutions is that if the objective function includes the service utility functions, the services need to disclose their (economic) utility functions to data centers which is not a preferable solution for service providers.

On the other hand, distributed systems have a storied history on the internet. Interior gateway protocols such as OSPF [6] and IS-IS [7] are fully distributed without any central control. Distributed file sharing applications such as BitTorrent [8] have made a substantial impact in content distribution. Session initiation protocol [9] is a distributed implementation of telephony and BlockChain [10] implements a distributed trust framework. All of these technologies have the fundamental aim of improving their core performance by removing centralized entities.

In this paper, we investigate a distributed orchestration system for data centers that can avoid the single-point-of-failure and scalability issues of centralized orchestration solutions. We model the resource allocation problem as a mixed integer non-linear problem with the objective of maximizing the global system utility function. We then propose a distributed solution to solve this optimization problem based on the notions of Markov approximation and auction theory. The solution is designed such that each service acts independently of the others with local knowledge of the system while the actions of all the services result in maximizing the total system utility. Our simulation results demonstrate that our proposed solution can effectively find near optimal resource allocation solutions in a distributed manner, and the solution is robust and scalable.

The rest of this paper is organized as follows: In Section II, we go over related projects and publications in the literature; we formally define our system model and the resource allocation problem in Section III and propose our distributed resource allocation solution in Section IV; we present our experimental results in Section V; finally, we discuss and conclude our work in Section VI.

II. Related Work

Cloud platforms come in two flavours - the big commercial platforms offered by Amazon [11], Google [12] and Microsoft [13] (and others), and open source tools that allow an organization to run its own cloud platform. The commercial platforms use a proprietary implementation and there is not much information publicly available about their internals.

Two mainstream open source projects are OpenStack [5]and Kubernetes [4]. OpenStack is an open-source cloud com-

978-3-903176-15-7

puting platform for managing virtual machines. Kubernetes is an open-source container orchestration system originally designed by Google. Both systems use a greedy centrallized resource scheduler where resources are assigned to services on a first come/first server basis.

Apart from the industrial platforms for resource scheduling/orchestration in data centers, resource management for cloud-based data centers has been well studied in academic literature in the past few years [14], [15]. For instance, Ngenzi *et al.* [16] proposed a dynamic centralized resource management algorithm based on bin-packing algorithms to improve the utilization of CPU and memory in data centers. Byde *et al.* [17] proposed to allocate resources based on market demand and supply and used a bidding mechanism to determine resource allocation. Zhang *et al.* [18] propose a distributed Stackelberg game to determine the price of resources dynamically among service subscribers and data center operators. Nezarat *et al.* [19] use a repetitive auction game with incomplete information in a non-cooperative environment to determine the pricing of resources. The service utility function in [19] is service specific and defined as the inverse of the cost value of each service. There is no notion of global system utility maximization in any of the solutions presented above. The model considered in our paper is a more comprehensive one which covers both operational and economic system constraints.

The closest work to our paper is from Metwally *et al.* [20] which proposes an auction-based resource allocation solution in geographically distributed data centers. The difference between our work to [20] is that our approach is fully distributed and run over all servers of the data center regardless of their geographical location while the solution of [20] is running separate auctions per data center domain. Moreover, no notion of service utility optimization is used in [20].

III. Problem Specification

In this section, we describe the architectural model for distributed orchestration and also formulate the global system utility optimization problem based on the system constraints.

A. Architectural Model

Our system model consists of a data center with K servers and a number of services denoted by N depicted in Fig. 1. For this initial work, services negotiate for compute resources and we assume adequate memory, storage and network capacity exists. Each service has a set of constraints that define its operating envelope.

Each service n is assigned a budget B_n to purchase resources from the data center. Each server k has a minimum price for its compute resources called the *reserve price* of server k and is denoted by θ_k. We define $s_{n,k}$ as the amount spent by service n over server k. Due to budget limit B_n, we have the following inequality satisfied for any service n.

$$\sum_k s_{n,k} \leq B_n \quad \forall n \tag{1}$$

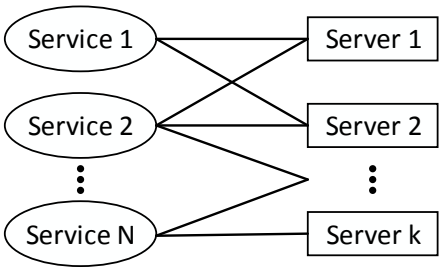


Fig. 1: System model - services, servers,and their relationship

Each service n has a constraint on the number of servers it can run on. We denote this limit by L_n also called *max server limit* for service n. The indicator variable $I_{n,k}$ is defined to be "1" if service n gets resources from server k and "0" otherwise. Therefore, any feasible allocation should satisfy the following inequality for any service n.

$$\sum_k I_{n,k} \leq L_n \quad \forall n \tag{2}$$

On the server side we have the following capacity constraint which states that the total amount of resource allocation on each server cannot exceed the total number of resource units available within the server.

$$\sum_n b_{n,k} I_{n,k} \leq r_k \quad \forall k \tag{3}$$

The total amount of processing allocated to each service is calculated through the following equation.

$$b_n = \sum_k b_{n,k} I_{n,k} \quad \forall n \tag{4}$$

We also assume that each service n demand is limited to D_n, i.e., the maximum aggregate amount of resource units required by service n is D_n units. Thus,

$$b_n \leq D_n \quad \forall n. \tag{5}$$

In terms of the resource pricing, we follow the idea of Johari-Tsitsiklis auction [21]. Therefore, the unit price of each resource unit over each server k denoted by p_k is determined by the total spending amount from the services and the total available resource, written as $\frac{\sum_n s_{n,k} I_{n,k}}{r_k}$. However, since each server also has a reserve price on its resource units, the resource unit price on server k is determined by

$$p_k = \max\left\{\frac{\sum_n s_{n,k} I_{n,k}}{r_k}, \theta_k\right\}. \tag{6}$$

Therefore, if service n spends $s_{n,k}$ on server k, its total allocation from server k would be

$$b_{n,k} = \frac{s_{n,k} I_{n,k}}{p_k}. \tag{7}$$

B. Problem Formulation

The goal of distributed orchestration is to determine how to allocate compute from the servers to the services with the objective of maximizing the global system utility function $U(\cdot)$ while satisfying the constraints mentioned above. The utility function $U(\cdot)$ is defined as the summation of all the services' utility functions $U_n(\cdot),\ \forall n$, i.e.,

$$U(\cdot) = \sum_n U_n(\cdot) \tag{8}$$

Depending on the choice of the service utility functions, we can control the resource allocation result, in terms of fairness and resource utilization efficiency. Choosing the logarithmic function as the utility function of each service results in proportionally fair allocation of the resources to the services. Proportional fairness is a well-accepted notion of fairness in resource allocation problems in networking and wireless systems [22], [23]. We use a weighted logarithmic function which results in a weighted proportional fair allocation of the resources weighted by the service budgets, i.e., $U_n(b_n) = B_n \log(b_n)$ Therefore, the resource allocation can be formulated in the following optimization problem:

$$\begin{aligned} &\underset{I_{n,k}, b_{n,k},\ \forall n,k}{\texttt{Maximize:}} && U(\mathbf{b}) = \sum_n B_n \log(b_n) \\ &\texttt{Subject to:} && \text{Constraints in (1) – (7)} \end{aligned} \tag{9}$$

Problem (9) is a mixed integer nonlinear optimization problem which is not easy to solve even via centralized approaches. Moreover, a centralized solution requires detailed information about the resource capacities of all the servers as well the service utility functions. In the following section, we propose a fully distribution scheme for solving this problem.

IV. Distributed Orchestration in Data Centers

In this section, we present our distributed approach for allocating computational resources in data centers to the services. The proposed solution provides a distributed mechanism for solving the optimization problem in (9).

We use the Markov approximation method in conjunction with the concept of auction theory to introduce a fully distributed orchestration architecture for solving resource allocation in data centers.

A. Markov Approximation Theory

Markov approximation was first proposed in [24] as a randomized method for approximating the solution to combinatorial optimization problems. In [24] Chen *et al.* showed that using the log-sum-exp function as an approximation for the objective function of a network utility maximization problem, we can build a time-reversible Markov chain whose steady state distribution solves the optimization problem. If we define

$$g_\beta(\boldsymbol{U}) = \frac{1}{\beta} \log \left(\sum_f \exp\left(\beta \sum_n U_n(f) \right) \right), \tag{10}$$

where f denotes a specific resource allocation configuration and $U_n(f)$ denotes the utility of user n given configuration f, then the log-sum-exp function $g_\beta(\boldsymbol{U})$ approaches $\max_{f \in F}(\sum_n U_n)$ as $\beta \to \infty$. By taking the conjugate of function $g_\beta(\boldsymbol{U})$ twice, we obtain Eq. (11) as

$$\max_{\mathbf{p}} \sum_f p_f \sum_n U_n(f) - \frac{1}{\beta} \sum_f p_f \log p_f \tag{11}$$

where $\sum_f p_f = 1$. Since the conjugate of a conjugate function returns the original function, Eq. (10) and Eq. (11) are equivalent, so that a solution to one is equivalent to a solution to the other. Eq. (11) is a concave function of variables p_f. We can analytically solve problem (11) using the Karush-Kuhn-Tucker equations to develop an "optimal" probability distribution as

$$p_f^*(\boldsymbol{U}) = \frac{\exp(\beta \sum_n U_n(f))}{\sum_{f'} \exp(\beta \sum_n U_n(f'))}. \tag{12}$$

$p_f^*(\boldsymbol{U})$ in (12) specifies the probability of being in each allocation configuration f. Since Eq. (11) is an approximation of combinatorial problem (10), the probabilities in (12) are such that they concentrate the probability into the optimal configuration. As β gets larger, the solution becomes closer to the optimal value. In practice, β is limited by the numerical precision that can be achieved with the exponential function.

If the probabilities in (12) are treated as the stationary probability distribution of a continuous-time time-reversible ergodic Markov chain, the design idea would be to build such a Markov chain and determine its transition probabilities. Existance of such a Markov chain was proven in [24].

Detailed balance equations provide the rules that we can use to develop transition rates between the states of the Markov chain from the equilibrium probabilities (12). The detailed balance equations state that for any two states f and g the transition rates between them are governed by the relationship $p_f q_{fg} = p_g q_{gf}$ where q_{fg} is the transition rate from state f to state g and p_f and p_g are the stationary probabilities of states f and g, respectively. Any transition rate solution that satisfies the local balance equations for all configurations will result in a valid continuous-time Markov chain. The Markov chain is then created by starting from a feasible initial state and then generating independent events which cause transitions to subsequent states. In this manner, instead of modeling the entire Markov chain, the distributed actors (i.e., services) need to implement transitions that conform to the transition rates described by the balance equations. This property of the Markov approximation method makes it a suitable candidate in design of a distributed orchestration system.

B. Computation of Stationary Probability and Transition Rates

Using the resource allocation formulation in (9) and also the Markov stationary probabilities in (12), we can determine the stationary probabilities of the Markov chain associated to our allocation problem as

$$p_f^*(\boldsymbol{U}) = \frac{\exp\left(\beta \sum_n B_n \log\left(\sum_k b_{n,k}\right)\right)}{\sum_{f'} \exp\left(\beta \sum_n B_n \log\left(\sum_k b_{n,k}\right)\right)}. \tag{13}$$

Using the detailed balance equations, the ratio of transition rates from the current state to a new state is given by

$$\frac{\exp\left(\beta \sum_n B_n \log\left(\sum_k b'_{n,k}\right)\right)}{\exp\left(\beta \sum_n B_n \log\left(\sum_k b_{n,k}\right)\right)}, \tag{14}$$

where $b'_{n,k}$ terms are defining the new allocation reflecting the new state we transition to in the Markov chain. In a fully distributed system, we have no mechanism to synchronize actors, i.e., the services. The services are expected to operate independently with no knowledge of the operation of the other services. Therefore, if only service n changes its state at a particular time, then most of these terms drop out, leaving us with a simplified transition rate

$$\frac{\exp\left(\beta B_n \log\left(\sum_k b'_{n,k}\right)\right)}{\exp\left(\beta B_n \log\left(\sum_k b_{n,k}\right)\right)}. \tag{15}$$

The numerator and denominator of Eq. (15) can both become very large. In practice, this expression should be evaluated as

$$\left(\frac{\sum_k b'_{n,k}}{\sum_k b_{n,k}}\right)^{\beta B_n}. \tag{16}$$

Eq. (16) determines the ratio of transition rates between two states. For a viable implementation of the Markov chain, we need to provide an absolute implementation of the transition rates between two states. If we set the transition rate from allocation b_n to allocation b'_n according to (17), we can observe that the ratio of the transition rates between allocations b_n and b'_n satisfies (16) and the Markov chain converges to the stationary probabilities in (13).

$$\left(\frac{\sum_k b'_{n,k}}{\sum_k b_{n,k}}\right)^{\frac{\beta B_n}{2}} \tag{17}$$

C. Resource Allocation Auction Design

At this point, we have designed a Markov chain that approximates our resource allocation problem whose accuracy is controlled by β. The challenge is that $b_{n,k}$ must be locally controller to implement a distributed system. To solve this problem, we propose to run Johari-Tsitsiklis auctions [21] on each server along with the requesting services from which the resource allocation variables $b_{n,k}$ are derived based on the services' spend values on each server. According to Johari-Tsitsiklis auction [21], the allocation $b_{n,k}$ is determined by the spend value from service n and the server resource price, written as $b_{n,k} = \frac{s_{n,k}}{p_k}$ where p_k is calculated using (6). Assuming that all services are informed of the resource price on each server, we design our resource allocation auction using the following rules:

Service Rules:

- Each service n implements a local Markov event generator that determines the increase/decrease of the service spend value $s_{n,k}$ on each server k and the transition rate to new spend values according to (17).
- Each server checks transition feasibility according to the constraints (1), (2) and (5).
- Service n initiates an auction on server k by sending an auction request message and spent $s_{n,k}$ on that auction.
- After the auction is over, service n receives a resource update message from server k containing its new allocation and the updated resource unit price of server k.
- The server resource unit price is then used by the services in Eq. (17) to determine the transition rates for increasing/decreasing the spend values.

Server Rules:

- When a server receives an auction request message from any service, it starts an auction by sending an auction start message to all participating services and asks for their latest spend values.
- After receiving the spend values, the server first determines the resource unit price according to (6) and then determines the allocation to each service using (7). The new allocations together with the new resource unit price are then sent to the services using resource update messages.

D. Price Estimation Methods

As explained in Sub-section IV-B, each service needs to know the resource unit price at each server when generating the Markov transitions. We consider four different methods for a service to estimate the resource unit price on a server listed in the order of increasing accuracy below:

1) The server's reserve unit price is used if the service is not currently purchasing resources from that server, otherwise it uses the price it is currently paying for resources. This is called "Naive" price estimation.
2) The servers distribute their current unit price to all the services on an ongoing basis using a broadcast or a gossip protocol. This is called "Price taker" price estimation.
3) The servers distribute their current unit price to all services and each service estimates the effect of its spend on the unit price. This is called "Anticipative" estimation.
4) All the services are instantaneously aware of the accurate unit price on all servers and make decisions based on the accurate price. This is called "Perfect" estimation.

Among these estimation methods only the "Naive" one is really practical. In "Price taker", it takes some time for servers to distribute their resource unit prices across the system. Thus, there is always the risk that the price information is out of date. For "Anticipative" one, building knowledge of the server's resource allocation method into each service would be a poor design choice as this would make it difficult to change server resource allocation behavior. Finally "Perfect" price estimation cannot be implemented in a distributed manner, as this would require perfect knowledge of the entire system.

V. Simulation Results

In this section, we present the simulation results demonstrating the effectiveness of the proposed scheme. Our simulation results present the performance of our solution in terms of convergence, resource utilization and accuracy. In the following sub-sections, we consider one fixed over-committed scenario

TABLE I: Service configurations for Scenario 1

Service name	Max servers	Max resources	Budget	Target proportional fair allocation
Service-1	15	100	100	80
Service-2	5	40	40	32
Service-3	5	40	40	32
Service-4	10	60	60	48
Service-5	10	60	60	48

and two operational dynamic scenarios. In all scenarios, we set $\beta = 4$.

A. Scenario 1: Over-committed requirements

In the first scenario, we consider a system with 20 servers from two server classes (10 servers of each class). Server class 1 has 8 compute units with a reserve price of 2 and server class 2 has 16 compute units with a reserve price of 1. There are 5 services as specified in Table I.

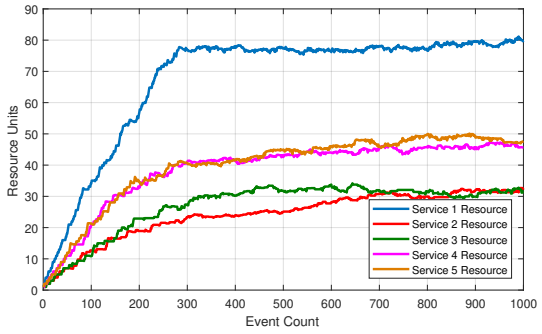
This is an over-committed scenario as the services request a total number of 300 resource units while only 240 resource units are available. This problem can be solved analytically, yielding the last column of Table I.

Our simulations implement the four different price estimation methodologies, namely "Naive", "Price taker", "Anticipative" and "Perfect". We have implemented and simulated these four methods in a proprietary event-driven simulator developed in Java. Fig. 2a shows the resource allocation results only for Naive price estimation using $\beta = 4$. It is observed that after about 700 Markov transitions, the Markov chain reaches its steady state where the allocations would not change considerably. We observe that in the steady state, the services reach their proportionally fair share of resources, i.e., last column of Table I. Note that the the length of the simulation in this scenario is 2000 seconds and the first 1000 events map to 57 seconds of the simulation which demonstrates very rapid convergence of the proposed scheme in terms of elapsed time.

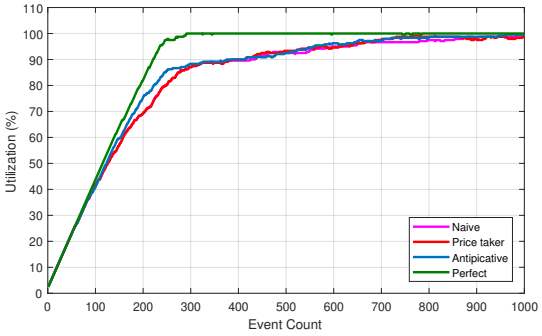
Average resource allocation for each service is shown in Table II. Since the system is over-committed - a total of 300 resource units have been requested, but only 240 are available - we can see that the proposed solution is performing within a few percent of target. We next investigate how well system resources are being utilized. Since the system is over-committed, we would like to see close to 100% resource utilization. Similarly, we can track performance via the values of the objective function. These results are shown in Fig. 2b and Fig. 2c. The "Perfect" method tracks target performance nicely, achieving around 95% resource allocation. The other methods results are functionally close to the "Perfect" method.

B. Scenario 2: Adding new services

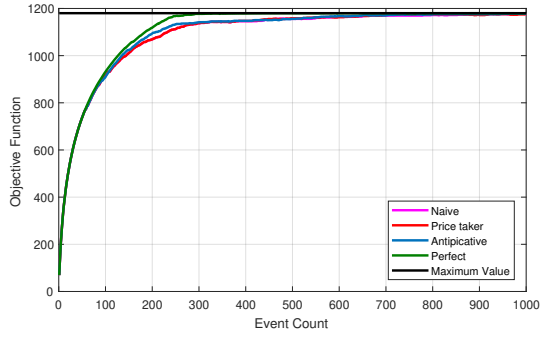
For these simulations we modify our base configuration in Scenario 1 to start the five services at different times. Service-1 starts at time 0, Service-2 starts at time 400, Service-3 starts at time 800 and so on. The data center configuration is unchanged. The results are shown in Fig. 3a for the Naive



(a) Resource allocation



(b) Resource utilization



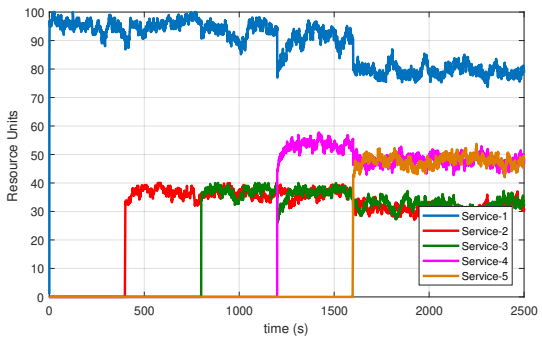
(c) Objective function

Fig. 2: Scenario 1 Simulation Results

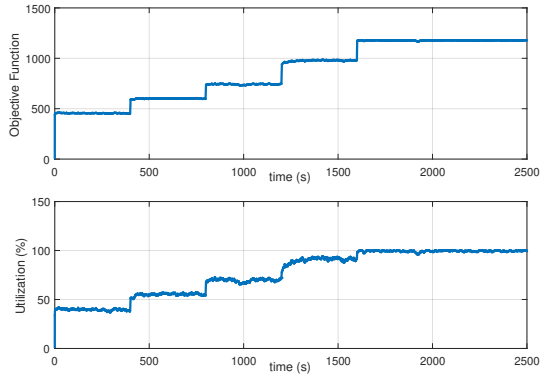
TABLE II: Allocations for different price estimation methods

Service name	Naive	Price taker	Anticipative	Perfect	Target
Service-1	79.4	79.5	81.9	81.9	80
Service-2	31.8	31.1	29.4	31.9	32
Service-3	31.6	31.3	29.4	31.7	32
Service-4	47.3	48.0	49.2	48.3	48
Service-5	48.1	48.3	49.7	47.0	48

price estimation. The system behaves in a manner consistent with the steady state simulations we have just studied, i.e., the system adjusts to the increased load consistently, ultimately achieving the target objective. Fig. 3b shows the system objective function as well as the resource utilization for this scenario. It is observed that by adding the services, the resource utilization and the objective function increase up

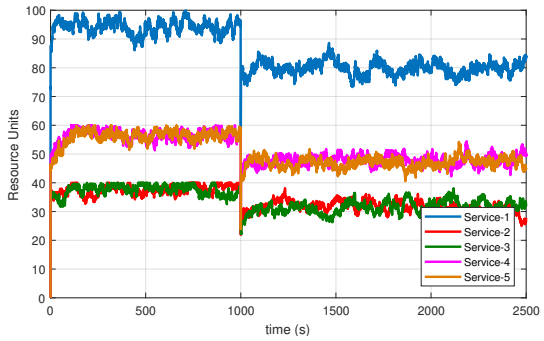


(a) Resource allocation

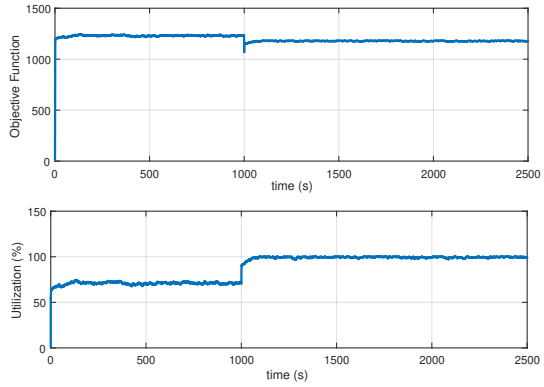


(b) Objective and Utilization

Fig. 3: Scenario 2: Sequential Service Activation



(a) Resource allocation



(b) Objective and Utilization

Fig. 4: Scenario 3: Server decommissioning

to the point the system becomes over-committed at which point the resource utilization reaches to 100% and the objective function approaches its maximum. Moreover, we can observe how fast the resource allocation converges to its optimal value after adding each service.

C. Scenario 3: Decommissioning servers

For these simulations, we added an extra 10 servers to the base configuration of Scenario 1 so that the data center starts with 20 servers in class 2. At time 1000, we decommission 10 of these servers and return to the base configuration. Simulation results are shown in Fig. 4a. The scenario starts with an under-committed system so that all services are able to buy resources for the reserve price. When 10 of the class 2 servers are decommissioned, the system rapidly returns to the proportional fair allocation we observed in Section V-A. Fig. 4b shows the system objective function as well as the resource utilization for this scenario. We observe that by decommissioning the extra servers the system becomes over-committed again and the resource utilization gets back to 100% again. We also observe that decommissioning the extra servers decreases the objective function slightly. This occurs as the scenario transitions undercommitted where service requirements are met, to overcommitted where service requirements cannot be met.

VI. Discussion and Conclusions

In this paper, we proposed a fully distributed orchestration system based on the notions of Markov approximation and auction theory. In the proposed scheme, the services/servers act independently and their actions collectively result in maximization of a global system utility function. Using simulations, we showed the effectiveness of the solution in different scenarios in terms of convergence, resource utilization and accuracy.

We are currently building a medium sized prototype system in our lab, consisting of 20 servers. This will allow us to study the impact of network messaging with non-zero delays and processing times, and to assess the impact of high rate events during system startup and reconfiguration.

Another area of interest is to study the efficacy of the method in more complex scenarios, where a service is implemented as a set of microservices. Resource dependencies between the individual microservices will place more demanding constraints on the system, and it will be important to understand how the method performs as the constraint set increases.

Finally, in the Markov approximation approach the actors cooperation results in the global optimal solution. An alternative assumption is that the actors are competitive instead of cooperative. Studying the impact of misbehaving actors on the overall system performance and also designing a robust solution that still provides global optimal solution would be the focus of future work in this are of research.

REFERENCES

[1] L. Peterson, A. Al-Shabibi, T. Anshutz, S. Baker, A. Bavier, S. Das, J. Hart, G. Palukar, and W. Snow, "Central office re-architected as a data center," *IEEE Communications Magazine*, vol. 54, no. 10, pp. 96–101, October 2016.

[2] Huawei. Cloud RAN & the next-generation mobile network architecture. [Online]. Available: https://www-file.huawei.com/-/media/CORPORATE/PDF/mbb/cloud-ran-the-next-generation-mobile-network-architecture.pdf

[3] S. Challita, F. Paraiso, and P. Merle, "A study of virtual machine placement optimization in data centers," in *7th International Conference on Cloud Computing and Services Science, CLOSER'17*, Porto, Portugal, April 2017.

[4] kubernetes.io. Production grade container orchestration - kubernetes. [Online]. Available: https://kubernetes.io

[5] openstack.org. Open source software for creating public and private clouds. [Online]. Available: https://www.openstack.org

[6] J. Moy, "Ospf version 2," Internet Requests for Comments, RFC Editor, STD 2328, April 1998, http://www.rfc-editor.org/rfc/rfc2328.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2328.txt

[7] "Intermediate system to intermediate system intra-domain routeing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode network service," International Organization for Standardization, Standard, 2002.

[8] B. Cohen. The bittorrent protocol specification. [Online]. Available: http://www.bittorrent.org/

[9] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "Sip: Session initiation protocol," Internet Requests for Comments, RFC Editor, RFC 3261, June 2002, http://www.rfc-editor.org/rfc/rfc3261.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3261.txt

[10] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[11] aws.amazon.com. Aws well-architected framework. [Online]. Available: https://d1.awsstatic.com/whitepapers/architecture/AWS_Well-Architected_Framework.pdf

[12] cloud.google.com. Google cloud platform. [Online]. Available: https://cloud.google.com/docs/compare/

[13] microsoft.com. (2018) Azure application architecture guide. https://docs.microsoft.com/en-us/azure/architecture/guide/.

[14] V. Anuradha and D. Sumathi, "A survey on resource allocation strategies in cloud computing," in *2014 International Conference on Information Communication and Embedded Systems (ICICES)*, Chennai, India, Feb. 2014.

[15] K. Wang, Q. Zhou, S. Guo, and J. Luo, "Cluster frameworks for efficient scheduling and resource allocation in data center networks: A survey," *IEEE Communications Surveys & Tutorials*, Jul. 2018, early Access.

[16] A. Ngenzi, S. R, and S. R. NAIR, "Dynamic resource management in cloud datacenters for server consolidation," *arXiv preprint arXiv:1505.00577*, May 2015.

[17] A. Byde, M. Sallé, and C. Bartolini, "Market-based resource allocation for utility data centers," *HP Lab, Bristol, Technical Report HPL-2003-188*, Sep. 2003.

[18] H. Zhang, Y. Xiao, S. Bu, R. Yu, D. Niyato, and Z. Han, "Distributed resource allocation for data center networks: A hierarchical game approach," *IEEE Transactions on Cloud Computing*, April 2018, early Access.

[19] A. Nezarat and G. Dastghaibifard, "Efficient nash equilibrium resource allocation based on game theory mechanism in cloud computing by using auction," *PloS one*, vol. 10, no. 10, p. e0138424, Oct. 2015.

[20] K. Metwally, A. Jarray, and A. Karmouch, "A distributed auction-based framework for scalable iaas provisioning in geo-data centers," *IEEE Transactions on Cloud Computing*, Feb. 2018, early Access.

[21] R. Johari and J. N. Tsitsiklis, "Efficiency loss in a network resource allocation game," *Math. Oper. Res.*, vol. 29, no. 3, pp. 407–435, Aug. 2004.

[22] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research Society*, vol. 49, no. 3, pp. 237–252, Feb. 1998.

[23] G. Miao, J. Zander, K. W. Sung, and B. Slimane, *Fundamentals of Mobile Data Networks*. Cambridge University Press, 2016.

[24] M. Chen, S. C. Liew, Z. Shao, and C. Kai, "Markov approximation for combinatorial network optimization," *IEEE Transactions on Information Theory*, vol. 59, no. 10, pp. 6301–6327, Oct. 2013.