Decentralized Utility Maximization for Adaptive Management of Shared Resource Pools

Emerson Loureiro, Paddy Nixon, and Simon Dobson
Systems Research Group
School of Computer Science and Informatics
University College Dublin, Dublin, Ireland
Email: {emerson.loureiro,paddy.nixon,simon.dobson}@ucd.ie

Abstract

Resource pools are collections of computational resources which can be shared by different applications. The goal with that is to accommodate the workload of each application, by splitting the total amount of resources in the pool among them. In this sense, utility functions have been pointed as the main tool for enabling self-optimizing behaviour in such pools. The goal with that is to allow resources from the pool to be split among applications, in a way that the best outcome is obtained. Whereas different solutions in this context exist, it has been found that none of them tackles the problem we deal with in a total decentralized way. In this paper, we then present a decentralized utility maximization approach for adaptive and optimal management of shared resource pools.

1. Introduction

Resource pools are collections of computational resources (e.g., servers) which can be used by different applications in a shared way [1]. The goal with that is to accommodate the workload of each application, by splitting the total amount of resources in the pool amongst them. This is possible through the use of *Resource Containers* [2], an abstraction which encapsulates a certain amount of resources, making it available to a specific application. In many cases, applications have QoS parameters that have to be met. Therefore, the resources available to them should be such that their QoS parameters are met, if possible.

The problem in this scenario is that the workload of the applications is likely to vary over time, and as a consequence, their resource demands will vary too [3][4]. Statically-defined resource shares, based for example on average or worst-case scenarios, are not suitable [5]. It is likely that resources will be wasted this way, for instance by allocating unnecessarily large shares and thus running the risk of failing to meet some applications' QoS. A better approach, instead, is to allow shares to be defined in an adaptive fashion, using the workload and QoS requirements of each application as input [1].

A usual trend, however, is not just to split the resources in the pool in a way that it meets the QoS parameters, but to do that in the best possible way. Precisely, that means finding the distribution of resources that yields the best outcome. To this end, utility functions have been pointed as the main tool for enabling this kind of behaviour [6], since they do not distinguish between desirable and undesirable allocations. Instead, allocations are distinguished by having a lower or higher utility, which then enables to find the best allocation, i.e., the one with the highest utility. Finding such an allocation consists, basically, on modelling the resource management problem as an optimization one, and eventually solving it. This has been called *Utility Maximization* (UM).

A number of solutions employing utility maximization for managing shared resource pools have been proposed. Many are based on centralized architectures, which are known to be not very scalable and suffer from fault-tolerance issues, i.e., crash of the centralizer. Some distributed solutions have also been proposed. They are all modelled hierarchically, however, and so coordination is centralized at the root of the hierarchy. Given the increasing scale of distributed systems and a stronger demand in terms of their autonomy [7], a truly decentralized solution is preferable, since they provide improved scalability and are naturally fault-tolerant. Whereas decentralized solutions exist in similar domains, they are not applicable to the problem being studied in this paper.

This work is supported by UCD Ad Astra Scholarships

Given that, in this paper we propose a true decentralized utility maximization (DUM) model for managing shared resource pools. To the best of our knowledge, this is the first work to present such a solution. For achieving that, we have employed the method of the Lagrange multipliers. Such methods have been used in similar works involving non-linear optimization. However, the problem being studied here along with the absolute decentralization characteristic of our DUM model, give it a crucial differential when compared to those works.

The rest of this paper is then organized as follows: in Section 2 some fundamental concepts are presented; our DUM model is presented in Section 3; an evaluation is presented in Section 4, demonstrating the feasibility of the model in a practical scenario, through simulations; related works in the area are presented and discussed in Section 5; finally, in Section 6, we conclude the paper with some final remarks and future directions of this work.

2. Fundamentals

In this section we provide basic concepts involved in our DUM solution. Firstly, because we are aiming at a decentralized approach, we view the system as a network of agents, where one agent can be reached by any other, directly or indirectly. In this case, each agent represents an application that consumes resources from the pool. We then denote by S the system itself and by a^i an agent in S, for $i \in [1, n(t)]$ where $n(t)$ is the number of agents in the system at time t.

Secondly, for utility maximization purposes, our solution is based on the approach proposed in [8]. For that, each agent a^i is assigned a utility function $u^i(x)$, stating how useful a particular resource share x from the pool is at a particular point in time. From that, a collective utility function $U(X)$ is defined, as follows:

$$U(X) = \sum_{a^i \in S} u^i(X_i) \tag{1}$$

where $X = \{X_1, X_2, \ldots, X_{n(t)}\}$ is an allocation vector and X_i is the resource share assigned to agent a^i. In practical terms, X_i could be, for example, the number of servers or amount of bandwidth allocated to a particular agent. Such an approach then maps every possible distribution of resources to a real-scalar value, which is used to distinguish between two different allocations. To find the best allocation at any point in time, the following optimization model, proposed in [9][10], is used:

$$\begin{aligned} &\max_{X \in \mathbb{R}^{n(t)}} U(X) \\ &\text{subject to: } \sum_{i=1}^{|X|} X_i = K(t), \end{aligned} \tag{2}$$

where $K(t)$ is the amount of resources available in the pool at time t, e.g., 100 servers. The constraint limits the sum of all resource shares to $K(t)$. In a practical setting, the value of $K(t)$ could be set by system administrators from a management station, then being propagated throughout the system [11].

3. Decentralized Utility Maximization

In this section we present our DUM solution for shared resource pools. More precisely, our solution consists on how to solve the optimization problem in Equation 2 in a truly decentralized way. For that, first, the utility function of the agents is defined as follows:

$$u^i(x) = 1 - e^{-\alpha^i(t)x}, \tag{3}$$

where x is the amount of resource from the pool being allocated to a^i and $\alpha^i(t)$ is a parameter that indicates a^i's demand for resources at time t. The smaller $\alpha^i(t)$ is, the greater is the agent's demand for resources. The reason for using such a utility function is because it will enable us to break down the optimization problem into separate models that each agent can use to find its optimal share. Like ours, other works have also used specific utility functions for different purposes [9][12].

Some plots of $u^i(x)$ are presented in Figure 1. The sharpness of the utility is controlled by $\alpha^i(t)$. The less sharp the utility is, the smaller is $\alpha^i(t)$, thus indicating a greater demand for resources. We assume $\alpha^i(t)$ might, and most likely will, change over time. However, it should remain constant during the actual process of finding the optimal allocation. Since $\alpha^i(t)$ represents an agent's resource demand at that particular time slot, it does not make much sense for it to change during the actual process of finding the optimal allocation, i.e., solving the problem in Equation 2.

From $u^i(x)$, we then transform the constrained optimization problem in Equation 2 into an unconstrained one. Using the method of the Lagrange multipliers, our new problem can be formulated as:

$$\max_{X \in \mathbb{R}^{n(t)}, \lambda \in \mathbb{R}} L(X, \lambda) \tag{4}$$

where $L(X, \lambda)$ is the Lagrangian of the problem in Equation 2, being defined as:

$$L(X, \lambda) = U(X) - \lambda \left(\sum_{i=1}^{|X|} X_i - K(t) \right). \tag{5}$$

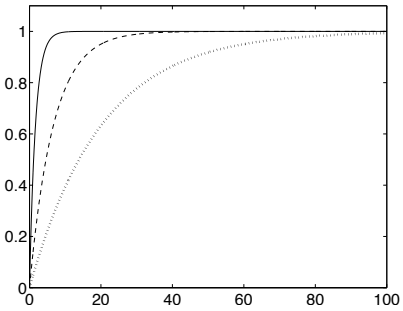

Figure 1: Sample plots of the utility of the agents

We then solve 4, in a way that it decomposes into the models that will calculate each agent's optimal share. For that, we can solve

$$\nabla L(X, \lambda) = 0,$$

which gives us the set of equations below.

$$\begin{aligned} \frac{\partial L}{\partial X_i} = 0, \forall i \in [1, |X|] \\ \frac{\partial L}{\partial \lambda} = 0. \end{aligned} \tag{6}$$

In theory, though, two points must be highlighted. First, for X to be an optimal solution of the optimization problem it must satisfy the Karush-Kuhn-Tucker (KKT) Conditions. Second, by solving $\nabla L(X, \lambda) = 0$, we would actually find a set of stationary points, each of which being a maximum, a minimum, or a saddle point. It is straightforward to show, however, that in our case, the solution to $\nabla L(X, \lambda) = 0$ satisfies the KKT conditions, is unique, i.e., only one stationary point exists, and also that such a stationary point is necessarily a maximum, and consequently the global maximum. Due to space constraints, we omitted the proofs from this paper. Back to Equations 6, each $\frac{\partial L}{\partial X_i} = 0$ will yield in:

$$\alpha^i(t) e^{\left(-\alpha^i(t) X_i\right)} - \lambda = 0.$$

Solving the equation above for X_i, gives us:

$$X_i = \frac{\ln \alpha^i(t) - \ln \lambda}{\alpha^i(t)}, \tag{7}$$

which then enables each agent to find its own share, such that $U(X)$ in Equation 2 is maximized. Note, first, that $X_i \in \mathbb{R}$, and so, fine-grained shares are supported. Second, coordination in this case is totally decentralized. To calculate X_i, however, agents need, besides their own $\alpha^i(t)$, the value of $\ln \lambda$, which is the global information that binds them together. Therefore, to compute their shares, they would need to compute $\ln \lambda$ first, also in a decentralized way. For that, we start with $\frac{\partial L}{\partial \lambda} = 0$, from Equation 6, which yields in:

$$\left(\sum_{i=1}^{|X|} -X_i \right) + K(t) = 0.$$

Substituting 7 in the above, we then have that:

$$\left(\sum_{i=1}^{|X|} \frac{\ln \lambda - \ln \alpha^i(t)}{\alpha^i(t)} \right) + K(t) = 0.$$

We can isolate $\ln \lambda$, ending up with:

$$\ln \lambda = \frac{\left(\sum_{i=1}^{|X|} \frac{\ln \alpha^i(t)}{\alpha^i(t)} \right) - K(t)}{\sum_{i=1}^{|X|} \frac{1}{\alpha^i(t)}}. \tag{8}$$

With that, each agent can then calculate $\ln \lambda$, and, once that is done, their own share through Equation 7.

Because $\ln \lambda$ depends on the α of all agents, and because we do not want any kind of centralization in the system, we assume that either each $\alpha^i(t)$ will be disseminated throughout the system, eventually reaching every other agent [13], or $\ln \lambda$ will be computed using approaches for calculating aggregates in networked systems [14][15].

In the first case, each agent will end up with the α of the others, which is then combined with its own and used as input to Equation 8. In the second one, each agent a^i would hold two values, $\frac{\ln \alpha^i(t)}{\alpha^i(t)}$ and $\frac{1}{\alpha^i(t)}$. From that, one run of an aggregate algorithm would be executed for each value, to perform a SUM of all of such values. When the two sums are computed, each agent uses them appropriately in Equation 8, so as to find their own share. Both approaches can be performed in large-scale networks in very reasonable time, thus not compromising our DUM solution in terms of performance. Further discussion on the actual algorithms for computing $\ln \lambda$, however, is out of the scope of this paper.

4. Evaluation

In this section we present experiments we have performed using our DUM model. To this end, we have modelled a scenario where a number of Application Environments (AEs) are deployed in a data center, as proposed in [9], each AE processing one type of transaction. The scenario we illustrate here will then deal with the allocation of servers from the data center to the AEs deployed in it. In this case, each AE is represented by an agent implementing our DUM model.

4.1. Data Center Model

Each AE has an Expected Average Workload (EAW) at different points in time, in terms of number of requests per second. That can be obtained using online or offline prediction techniques. For our experiments, these workloads have been obtained from the analytical data of different web sites. Also, all AEs have a policy defining a Target Response Time (TRT) that should be guaranteed for the transactions they process. The resource management process will then find the optimal distribution of servers amongst the AEs, considering their EAW and TRT.

We denote by $r^i(s, w)$ the Expected Average Response Time (EART) of an AE during time t, representing the response time an AE will obtain given a workload w and a certain number of servers s allocated to it. We define $r^i(s, w)$ based on the model proposed in [9], as:

$$r^i(s, w) = \frac{w.c^i}{s}, \tag{9}$$

where c^i is the CPU time of the transaction processed by AE i (in seconds), w is the EAW of the AE (in requests per second), and s is the amount of servers assigned to it. From that, we derive $q^i(w)$, the required amount of servers that should be assigned to an AE, in order to meet its TRT, as follows:

$$q^i(w) = \frac{w.c^i}{T^i} \tag{10}$$

where w and c^i are as in $r^i(s, w)$ and T^i is the AE's TRT. The required amount of servers $q^i(w)$ is necessary for defining $\alpha^i(t)$, which, as defined in our DUM model, represents an AE's resource demand at a particular time t. Such a parameter is calculated as:

$$\alpha^i(t) = -\frac{\ln(1 - H)}{q^i(w)}, \tag{11}$$

where H represents the value of the agents' utility when the EART of its AE meets its TRT, i.e., a value very close to 1.

4.2. Simulation Results

Based on the data center model presented, a series of experiments have been ran, using different scenarios. In these experiments, a random epidemic algorithm for disseminating all $\alpha^i(t)$ has been used. Also, we assumed that the resource management process runs at distinct points in time, called *iterations*. In a real-world setting, these iterations could represent different hours of the day, on which a re-allocation of the servers would take place. The results for the experiments are then presented next.

4.2.1. Scenario 1: Static Number of AEs. In this scenario, the number of AEs over the entire simulation is constant. We considered that six AEs, whose EAWs are as in Figure 2, are deployed in the data center. Also, we assumed that 145 servers are available on the data center and that the CPU times of the transactions processed by each AE are as presented in Table 1. The latter has been based on values provided in [9].

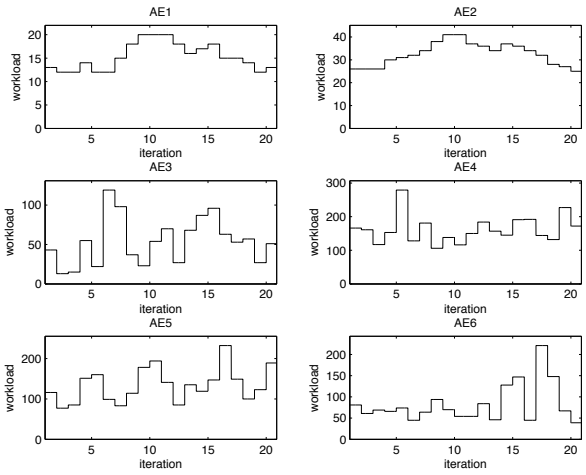

Figure 2: EAWs of each AE in Scenario 1

AE	CPU Time
1	0.11
2	0.015
3	0.045
4	0.08
5	0.01
6	0.096

Table 1: CPU Times (in seconds) for the transactions processed by the AEs

After running the simulation during twenty iterations, the shares found by each agent were as presented in Figure 3. The important aspect to note is the way the shares vary. Note that, the general shape of the graphs of the shares vary similarly to the way the workload does. Therefore, from a high-level perspective, our DUM solution captures the demands correctly, and acts properly towards the optimal share. At a lower level, one can see that, sometimes variations between the shares and workload do not match. In a general way, it is clear that those were the variations that yielded in the highest $U(X)$, even though the specific reasons for such can vary. As an example, notice that, at iteration 5, AEs 2, 4, 5, and 6 have an increase on their workload, but only AE 5 has an increase on its share. That is because the workload increase in AE

5 was simply too high to allow an increase in the shares of AEs 2, 4, and 6 such that $U(X)$ would be maximized.

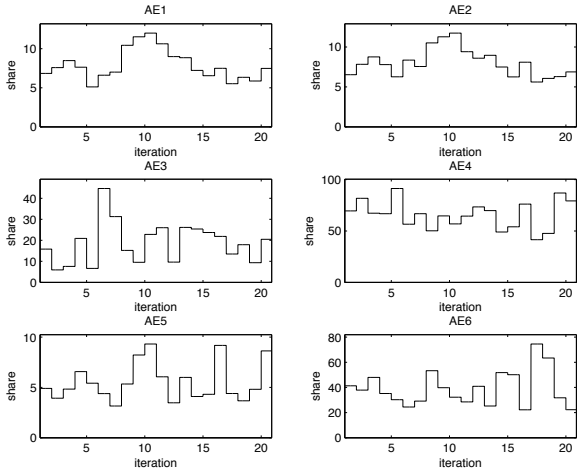

Figure 3: Shares of each AE in Scenario 1

The fact that our DUM model captures demand correctly is reinforced by the results presented in Figure 4, where the $\alpha^i(t)$ of each AE over all iterations are presented. Note that the values of $\alpha^i(t)$ vary exactly the opposite to the way the workload does. This thus matches the definition of the agents' utility function, on which it is stated that the greater the workload is, the smaller is the value of $\alpha^i(t)$, indicating a greater demand for resources.

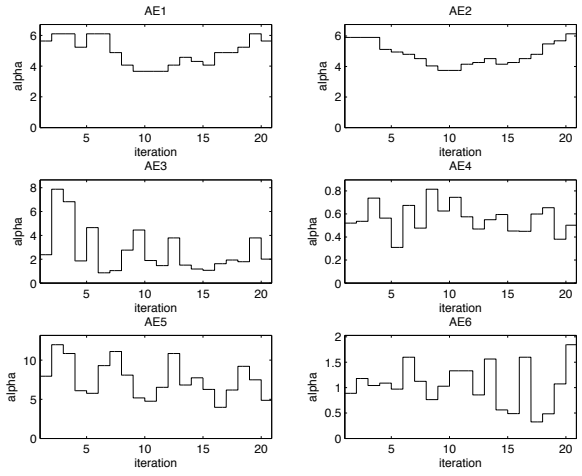

Figure 4: $\alpha^i(t)$ of each AE in Scenario 1

As a consequence of properly capturing $\alpha^i(t)$, the EARTs for all AEs end up as in Figure 5. Note that, for all AEs, such response times are always smaller than what is specified in their TRT, represented by the dashed horizontal line in each graph of the figure. Because, in this scenario, the data center always hosted more servers than the demand, the aggregate utility was always such that $U(X) \approx 6$.

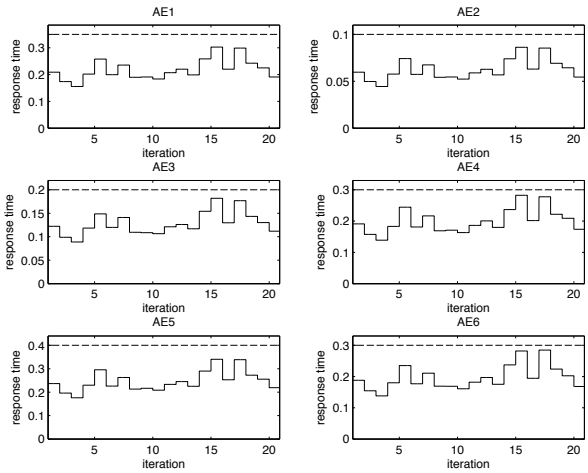

Figure 5: EARTs of each AE in Scenario 1

4.2.2. Scenario 2: Varying Number of AEs. In a practical setting, we cannot expect the system to be static. As it evolves, AEs will join and leave the data center. Consequently, our solution should support such dynamics, which is what we have simulated in this scenario. To this end, the data center was initially set up with four AEs, until iteration ten, when two AEs join the system. Then, at iteration fifteen, one of them leaves the system, keeping this setting until the end of the simulation. The number of servers and CPU times for this scenario are the same as for the first one.

The EAWs for this scenario are then illustrated in Figure 6. After running the simulation, the shares found were such that the EARTs in Figure 7 were obtained. As with the previous scenario, note that the EARTs of each AE is always smaller than their TRT (dashed horizontal line in each graph). The figures for the $\alpha^i(t)$'s and shares found were similar to the ones presented in the first scenario, and so we omitted them from this paper.

4.2.3. Scenario 3: Overload. Finally, in a third scenario, we observed how our solution behaves when facing overload in the data center. In other words, in some iterations, we allowed the demand to be greater than the number of servers available in the Data Center. For that, the number of servers has been set to 100. Furthermore, the CPU times used and workloads were as in the first scenario.

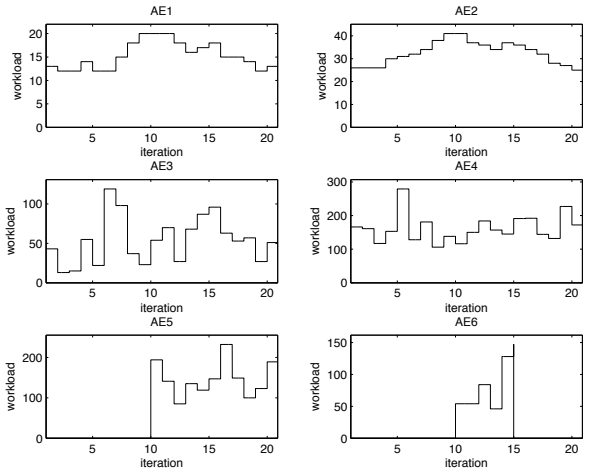

Figure 6: EAWs of each AE in Scenario 2

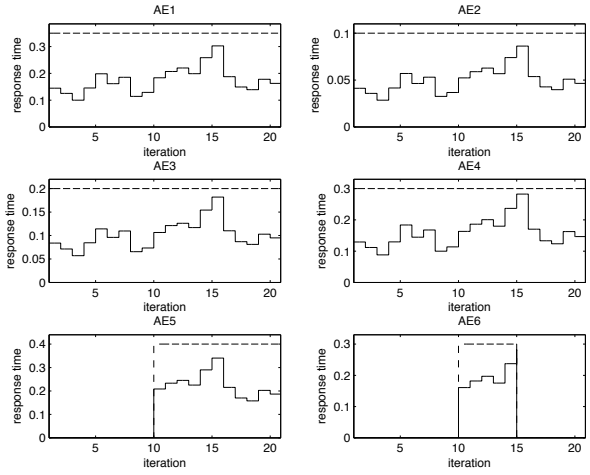

Figure 7: EARTs of each AE in Scenario 2

The overload is illustrated in Figure 8, which plots the variation of the total server demand over the iterations (the solid line represents the number of servers). Because of that, the EARTs were then as in Figure 9. Since overloading was being considered, in some iterations, the TRTs of some, or all, AEs could not be met. Still, our DUM model distributed the shares in a way that maximized the utility as stated in the problem formulation. To see that, we present in Figure 10 the variation of the aggregate utility. Note that, on the iterations where overload did not happen, the utility obtained was still the highest possible, i.e., $U(X) \approx 6$, consequently decreasing during overload periods. Also, the point where the aggregate utility reached its lowest value was the exact moment where the total server demand reached its highest value.

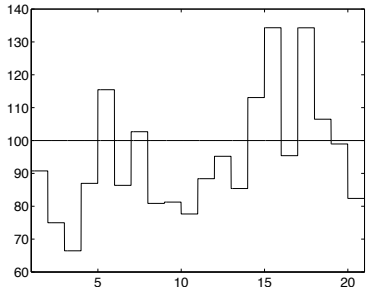

Figure 8: Total amount of required servers over the overload scenario

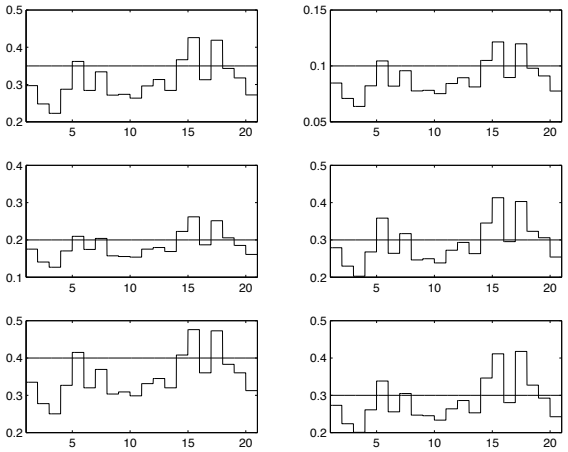

Figure 9: EART of the AEs over the overload scenario

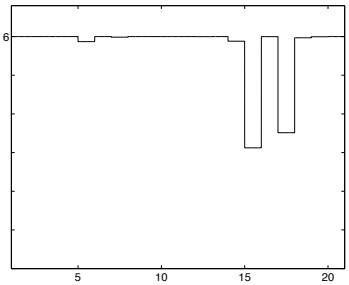

Figure 10: Aggregate utility over the overload scenario

5. Related Work

A number of solutions for performing resource management has been proposed. Many of them, however, approach the problem using centralized models [3][9][5]. In this case, a central authority is in charge of deciding the resource shares across the system. Even though these solutions can perform well, they are not very scalable because of the centralizers. Also, they are not fault-tolerant, since the crash of the centralizer compromises the entire system.

Solutions with a more distributed characteristic have also been proposed. In [12], for example, market agents are used. The solution differs from ours in the sense that centralizing entities, called brokers, are inserted in the resource management process. The decomposition methods presented in [16] are another distributed solution. These methods are similar to our DUM model in that they also employ Lagrange multipliers to decompose an optimization problem into smaller problems, which can further decomposed, forming a hierarchy. They then rely on a messaging scheme which employs a central problem. Similarly, a hierarchical optimization model is presented in [17]. In both cases, coordination is done at the root of the hierarchy, whereas in our case, this is decentralized.

Solutions featuring decentralized control exist in similar domains. Examples of such solutions are [18][19], which employ market agents. Their focus, however, is not on utility maximization, unlike our DUM model. In [11], it is presented a decentralized solution for allocating servers to different classes of service. This solution is modelled differently though, in that resource providers, and not consumers, solve the optimization problem, like in our DUM model. Besides, it is specifically focused on server allocation, whereas we have aimed at a more general approach. In [20], gossiping is used to allow a set of P2P-connected traffic limiters to control the bandwidth they use. It is different from our solution in the sense that it does not focus on utility maximization. The same can be said from the approach proposed in [21], which allows servers to be allocated to applications in a decentralized way. In terms of distributed optimization, in [22], subgradient methods are used to optimize the aggregate of a set of agents' cost function. The solution, however, does not incorporate resource constraints, limiting its applicability in practical resource management scenarios. Finally, in [23], a DUM model is proposed, but it is specifically focused on the control of multiple multicasts in P2P systems, and thus, does not apply to the problem we are dealing with.

6. Conclusions

In this paper, we presented a decentralized utility maximization (DUM) model for managing shared resource pools in an adaptive and optimal way. More precisely, we employed the method of the Lagrange multipliers along with the utility functions theory to devise a method where each agent in the system knows how to calculate its share, so that the best outcome can be obtained. The problem being studied, along with the total decentralized characteristic of our DUM model, give it a distinctive feature. To the best of our knowledge, then, this is the first work to present such a decentralized solution in the domain of shared resource pools.

An evaluation has been presented, through simulations, in a server allocation scenario in a data center. We demonstrated that our DUM model is able to capture resource demands properly and deliver shares that meet all applications' QoS parameters, when possible. Scenarios where the number of applications in the system varies, which are to happen in the real-world, have been simulated. As we showed, our DUM model also handles these scenarios in an optimal way. Finally, in overload situations, even though not all QoS parameters could be met, our solution was still able to find the allocation leading to the best outcome.

As future work, we are aiming at a specific epidemic algorithm for disseminating the $\alpha^i(t)$ values throughout the system. The main reason for such is that the current methods for computing aggregates like our $\ln \lambda$ would not suit us in terms of scalability, precision, and fault-tolerance. Furthermore, we will apply our DUM model to other shared resource pools scenarios, to have an insight on how general it really is. We do believe, however, that our DUM model could handle other scenarios quite well.

References

[1] J. Rolia, L. Cherkasova, M. Arlitt, and V. Machiraju, "Supporting application quality of service in shared resource pools," *Commununications of the ACM*, vol. 49, no. 3, pp. 55–60, March 2006.

[2] G. Banga, P. Druschel, and J. C. Mogul, "Resource containers: a new facility for resource management in server systems," in *Proceedings of the Third symposium on Operating systems design and implementation*. Berkeley, CA, USA: USENIX Association, 1999, pp. 45–58. [Online]. Available: http://dx.doi.org/10.1145/224057.225831

[3] X. Wang, Z. Du, Y. Chen, and S. Li, "Virtualization-based autonomic resource management for multi-tier web applications in shared data center," *Journal of Systems and Software*, vol. 81, no. 9, pp. 1591–1608, 2008.

[4] J. Guitart, D. Carrera, V. Beltran, J. Torres, and E. Ayguadé, "Dynamic CPU provisioning for self-managed secure web applications in smp hosting platforms," *Computer Networks*, vol. 52, no. 7, pp. 1390–1409, 2008.

[5] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, "Adaptive control of virtualized resources in utility computing

environments," in *Proceedings of the 2007 European Conference on Computer Systems*. New York, NY, USA: ACM Press, 2007, pp. 289–302.

[6] J. O. Kephart and R. Das, "Achieving self-management via utility functions," *IEEE Internet Computing*, vol. 11, no. 1, pp. 40–48, 2007.

[7] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, January 2003. [Online]. Available: http://dx.doi.org/10.1109/MC.2003.1160055

[8] G. Tesauro and J. O. Kephart, "Utility functions in autonomic systems," in *ICAC '04: Proceedings of the First International Conference on Autonomic Computing*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 70–77.

[9] M. N. Bennani and D. A. Menascé, "Resource allocation for autonomic data centers using analytic performance models," in *Proceedings of the Second International Conference on Automatic Computing*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 229–240.

[10] G. Tesauro, W. E. Walsh, and J. O. Kephart, "Utility-function-driven resource allocation in autonomic systems," in *Proceedings of the Second International Conference on Autonomic Computing*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 342–343.

[11] B. Johansson, C. Adam, M. Johansson, and R. Stadler, "Distributed resource allocation strategies for achieving quality of service in server clusters," in *Proceedings of the 45th Conference on Decision and Control*. IEEE Computer Society, December 2006, pp. 1990–1995.

[12] X. Bai, D. C. Marinescu, L. Bölöni, H. J. Siegel, R. A. Daley, and I. J. Wang, "A macroeconomic model for resource allocation in large-scale distributed systems," *Journal of Parallel Distributed Computing*, vol. 68, no. 2, pp. 182–199, 2008.

[13] A. M. Kermarrec and M. Van Steen, "Gossiping in distributed systems," *SIGOPS Operating Systems Review*, vol. 41, no. 5, pp. 2–7, 2007.

[14] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Transactions on Computer Systems*, vol. 23, no. 3, pp. 219–252, August 2005.

[15] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 2003.

[16] D. P. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 8, pp. 1439–1451, 2006.

[17] T. Nowicki, M. S. Squillante, and C. W. Wu, "Fundamentals of dynamic decentralized optimization in autonomic computing systems," in *Self-star Properties in Complex Information Systems*. Springer-Verlag, 2005, pp. 204–218.

[18] P. R. Lewis, P. Marrow, and X. Yao, "Evolutionary market agents for resource allocation in decentralised systems," in *Proceedings of the 10th International conference on Parallel Problem Solving from Nature*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 1071–1080.

[19] R. Maheswaran and T. Başar, "Nash equilibrium and decentralized negotiation in auctioning divisible resources," *Group Decision and Negotiation*, vol. 12, no. 5, pp. 361–395, 2003.

[20] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren, "Cloud control with distributed rate limiting," *SIGCOMM Computer Communication Review*, vol. 37, no. 4, pp. 337–348, 2007.

[21] T. Masuishi, H. Kuriyama, Y. Ooki, and K. Mori, "Autonomous decentralized resource allocation for tracking dynamic load change," in *Proceedings of the 7th International Symposium on Autonomous Decentralized Systems*, 2005, pp. 277–283.

[22] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *Automatic Control, IEEE Transactions on*, vol. 54, no. 1, pp. 48–61, 2009.

[23] M. Chen, M. Ponec, S. Sengupta, J. Li, and P. A. Chou, "Utility maximization in peer-to-peer systems," in *Proceedings of the 2008 ACM SIGMETRICS International conference on Measurement and modeling of computer systems*. New York, NY, USA: ACM, 2008, pp. 169–180.