Decentralized Self-optimization in Shared Resource Pools

Emerson Loureiro*, Paddy Nixon, and Simon Dobson

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 and self-optimizing approach for resource management in 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], abstractions that can co-exist in a physical resource (e.g., server),

Emerson Loureiro · Paddy Nixon
Systems Research Group
School of Computer Science and Informatics
University College Dublin, Dublin, Ireland
e-mail: {emerson.loureiro,paddy.nixon}@ucd.ie

Simon Dobson
School of Computer Science
University of St Andrews,
St Andrews, United Kingdom
e-mail: sd@cs.st-sndrews.ac.uk

* This work is supported by UCD Ad Astra Scholarships.

S. Caballé, F. Xhafa & A. Abraham (Eds.): INCoS 2009, SCI 329, pp. 149–170.
springerlink.com

each of them receiving a slice of it. Examples of resource containers include virtual machines and virtual disks. These resource containers are then aggregated into Resource Shares, thus forming partitions of the total amount of resources available in the pool.

In this scenario, the applications might 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 case, 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 the 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 such a self-optimizing 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 process as an optimization problem, and eventually solving it. This has been called Utility Maximization (UM).

As a consequence of the above, employing Utility Maximization provides benefits, over other methods, when faced with conflicting scenarios. An example of such is during an overload in the system; i.e., the overall resource demand is greater than the amount of resources available to be allocated. In that case, it is clear that not all QoS requirements will be met. Still, with Utility Maximization, it is possible to find a way of maximizing resource usage, given the overload condition, thus providing directions as to how to act in such a conflicting scenario. In another case, the system could be facing a low load, in which case there might be several distributions of resources that meet all QoS requirements. Utility functions, again, provide unambiguous guidance towards the best way to do so. The important aspect, then, is that there will be situations where different resource distributions are possible. In these cases, unambiguous guidance as to how actually to do so is then crucial. Whereas other methods might be able to provide such guidance, Utility Maximization achieves that with an optimality aspect, regardless of the current setting.

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 though, 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 in similar domains exist, they are not applicable to the problem being studied in this paper.

Given that, here we propose a truly Decentralized Utility Maximization (DUM) model for managing shared resource pools, in an adaptive and optimal way. 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 related to our DUM solution. More precisely, we present an overview of the ideas behind shared resource pools and also how utility functions are linked to these ideas, then leading to a proper formalization of the problem being studied here.

2.1 Shared Resource Pools and Related Paradigms

Shared resource pools are collections of computational resources, aggregated in a way to allow concurrent access to them[1]. This is done by splitting the resources from the pool into resource shares, or simply shares, which are in turn distributed to applications in need. In more practical terms, as discussed previously, each share is an aggregation of Resource Containers. Even though different terminologies can be found in the literature, e.g., Cluster Reserve[28], Server[29], and Application Environment[8], for the purposes of this paper, the term Resource Share will be used.

An example of a shared resource pool is illustrated in Figure 1. In this figure, a collection of servers, i.e., the pool, is split into shares – the solid lines surrounding the different sets of servers – which are then assigned to particular applications (APP in the figure). The same idea applies, for example, to a Distributed Rate Limiting scenario[20]. In this case, the pool is composed solely by the bandwidth capacity available. This capacity is then partitioned into shares to be provided to different traffic limiters. This way, they can serve their network flows in an optimal way, without, however, overusing a specific bandwidth capacity.

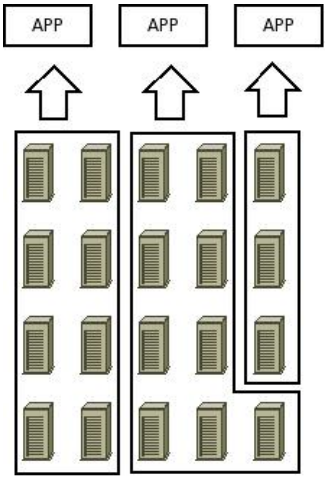

Fig. 1 Shared resource pool in a data center scenario

It is important to also point out that the idea of resource allocation and shared pools is found in the context of Utility Computing and Cloud Computing. Utility Computing is the on-demand packaging of computing resources so as to meet customers' needs, by dynamically creating virtual partitions of the resources available[30]. In this paradigm, the resource pool is viewed as a public infrastructure[31], in the sense that it is available to whoever has the need and is willing to pay for it.

The idea of Cloud Computing employs a similar model to go even further; moving away both data and computing from clients, placing them into large data centres[32], i.e., the cloud. The Cloud Computing paradigm, however, does not really focus on whether the infrastructure is public or private, in which case its resources are only available to the applications of whoever owns it[31]. Instead, its focus is more on providing a platform where not only hardware resources, but also applications, are provided as a service.

The work in this paper, putting it into the perspective of both paradigms, is neutral in terms of whether the infrastructure is public or private, like in Cloud Computing. Unlike Cloud, however, it is specifically focused on the delivery of hardware resources, and not applications. From the point of view of this work, applications can then be deployed as a service (e.g., a Google Doc-like spreadsheet application) or not (e-commerce application). Regardless of the access level to the resource infrastructure and nature of the applications, technically speaking, the allocation of resources can be done in the same way. In the end, it all boils down to consumers with resource demands over a common amount of resources. Consequently, the work presented here is clearly applicable to both the Utility and Cloud computing paradigms, even though it is being put in a more general context.

2.2 Problem Formalization

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]. In this case, each agent a^i is assigned a utility function $u^i(x)$, stating how useful a 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:

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

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 DUM Model

In this section we present our solution for decentralized self-optimization in 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 resource demand at time t. The smaller $\alpha^i(t)$ is, the

greater is the agent's resource demand. 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 2. 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, i.e., solving the problem in Equation 2. 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 within such an interval. This is a similar requirement for the centralized case, where a central entity, a Solver in this case, solves the optimization problem. In this case, once the Solver starts trying to find the solution for the optimization problem, the variables are not allowed to change until it is finished, or the solution found will just not be the correct one. That is then not a limiting factor from our DUM model, but simply something inherent to the scenario we are dealing with.

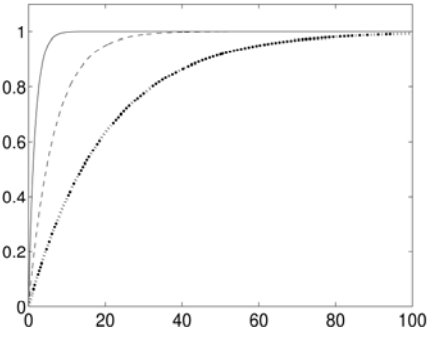

Fig. 2 Sample plots of the utility of the agents

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, the new problem can be formulated as:

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

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

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

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.

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

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 can be shown, 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. The proofs for those can be found in the Appendix. Back to Equations 6, each $\partial L/\partial X_i = 0$ will yield in:

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

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 $X_i \in R$, 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 $\partial L/\partial \lambda = 0$, 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* λ, ending up with:

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

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

Because *ln* λ 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* λ 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 are then combined with its own and used as input to Equation 8. In the second one, each agent a^i would hold two values, $ln\ \alpha^i(t)/\alpha^i(t)$ and $1/\alpha^i(t)$. 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 solution in terms of performance. Further discussion on the actual algorithms for computing *ln* λ, 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, according to 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. This then models the fact that agents are "happy" with an amount of resources that causes their TRT to be met, but also being "happier" if more resources are given.

4.2 Simulation Results

Based on the data center model presented, a series of experiments have been run, 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 reallocation 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 3, 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].

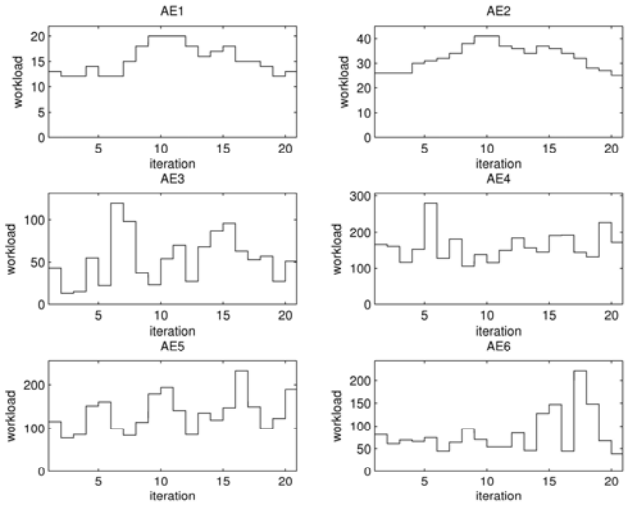

Fig. 3 EAWs of each AE in Scenario 1

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

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

After running the simulation during twenty iterations, the shares found by each agent were as presented in Figure 4. 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.

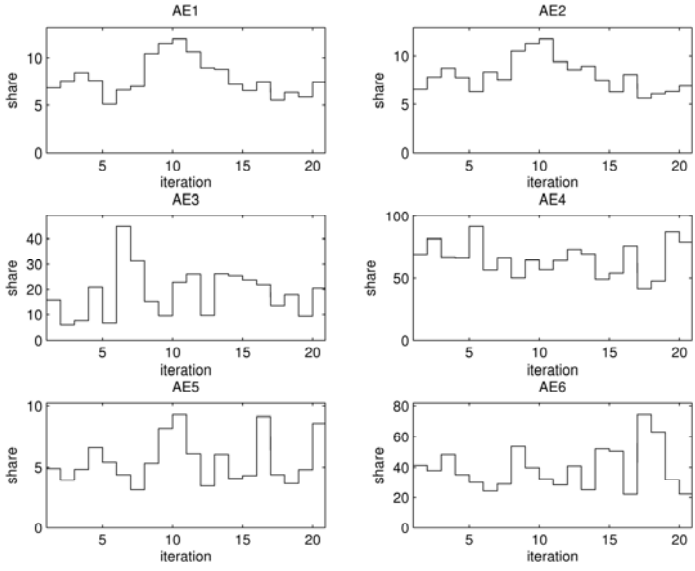

Fig. 4 Shares of each AE in Scenario 1

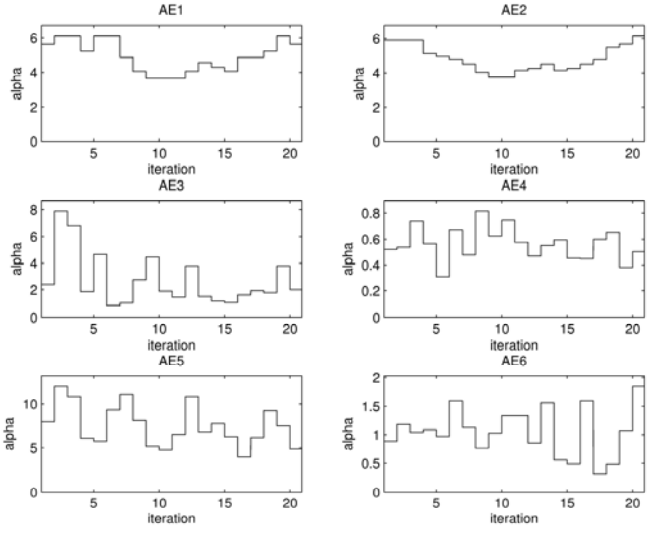

Fig. 5 $\alpha^i(t)$ of each AE in Scenario 1

The fact that our DUM model captures demand correctly is reinforced by the results presented in Figure 5, 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.

As a consequence of properly capturing $\alpha^i(t)$, the EARTs for all AEs end up as in Figure 6. 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 $U(X) \approx 6$, i.e., the maximum under any condition.

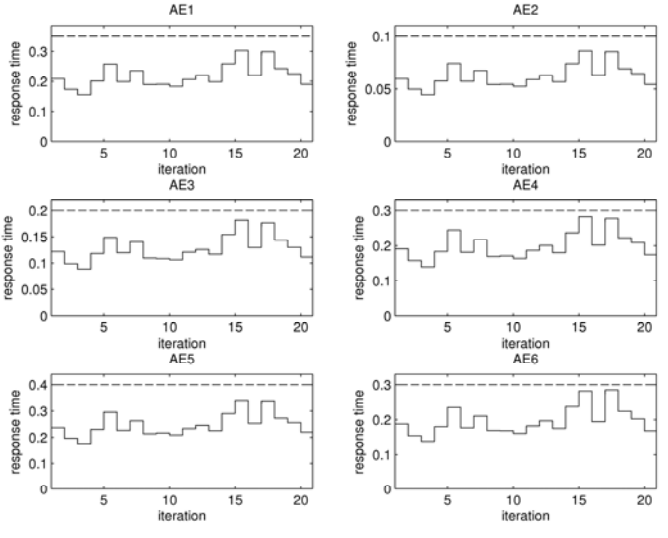

Fig. 6 EARTs of each AE in Scenario 1

4.2.2 Scenario 2: Varying Number of AEs

In a real world 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 the CPU times for this scenario are the same as for the first one.

The EAWs for this scenario are then illustrated in Figure 7. After running the simulation, the shares found were such that the EARTs in Figure 8 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). This demonstrates that our DUM model supports these changes smoothly. The figures for the $a^i(t)$'s and shares found were similar to the ones presented in the first scenario, and so we omitted them.

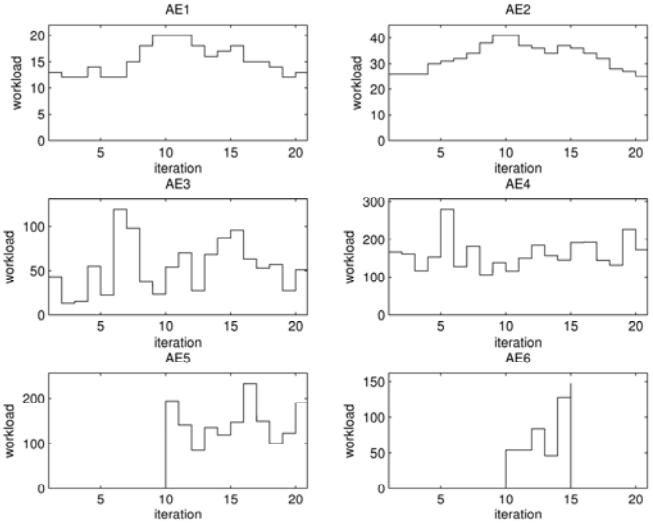

Fig. 7 EAWs of each AE in Scenario 2

4.2.3 Scenario 3: Varying QoS Parameters

Just like the system structure is prone to changes in real world settings, so are the QoS requirements of the applications in a shared resource pool. Contracts between customers and infrastructure providers might change over time, thus directly impacting in resource demands. We should then be able to show that our DUM model can also handle such variations in QoS requirements, so as to make it fully applicable in practice.

For that, we have run simulations similar to the ones for the first scenarios, but this time varying the TRT of the AEs deployed in the data center. The workloads of the AEs, as well as the number of servers and the CPU times of the transactions, were the same as for the first scenario. The variation of the TRTs, along with the final EARTs obtained, for each AE, is presented in Figure 9. Note that, even though the TRTs were varying over time, our DUM model was still able to deliver resource shares meeting all such requirements, for all AEs. It is worthy to point out, in this case, that since not only the workload, but also the QoS requirements, are varying, the $a^i(t)$ will now vary based on a combination of both, as presented in Figure 10. This contrasts with the first scenario, and it does so because in that case the TRTs were constant over time, and so, only the workload would affect the $a^i(t)$. In that case then, it would be easy to realize the behaviour of $a^i(t)$,

i.e., an increase/decrease in the workload, would necessarily cause an increase/decrease in resource demand, ending up with a decrease/increase of $\alpha^i(t)$. For this scenario, we omitted the figures for the shares obtained, since they present results similar to what has already been shown.

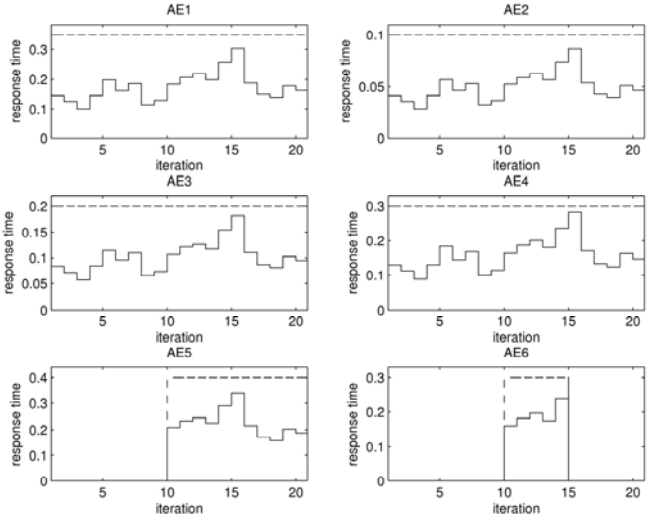

Fig. 8 EARTs of each AE in Scenario 2

4.2.4 Scenario 4: Overload

Finally, in a fourth scenario, we observed how our solution behaves when facing overload in the data center. In other words, in some iterations, we allowed the total 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. Again, the CPU times used and the workloads were as in the first scenario.

The overload is illustrated in Figure 11, which plots the variation of the total server demand over the iterations (the solid line represents the number of servers available). Because of that, the EARTs were then as in Figure 12. Since overloading was being considered, in some iterations, the TRTs of some, or all, AEs could not be met. To better illustrate that, we present in Figure 13 the variation of the aggregated 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. Consequently, the point where the aggregate utility reached its lowest value was the exact moment where the total server demand reached its highest value.

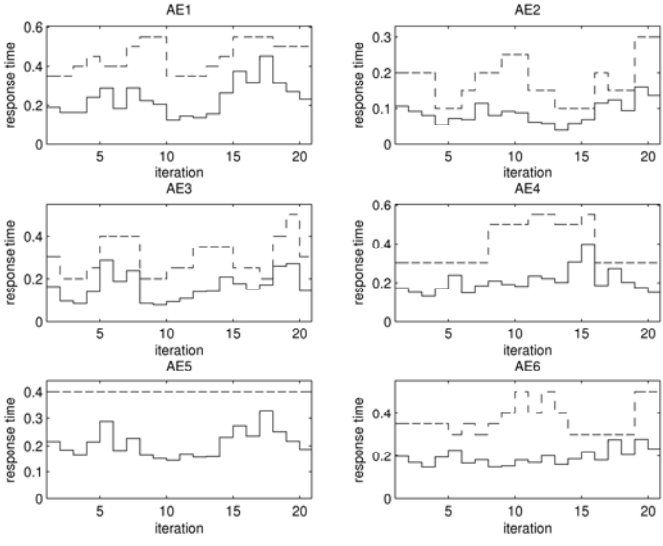

Fig. 9 TRTs and EARTs of each AE in Scenario 3

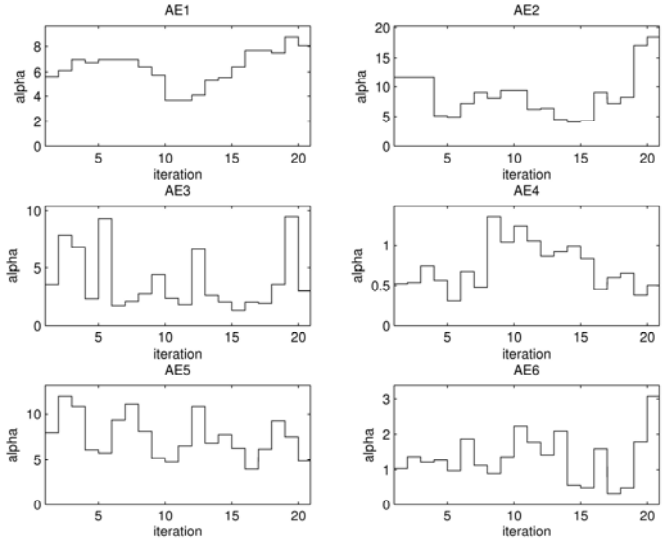

Fig. 10 $\alpha^i(t)$ of each AE in Scenario 3

Still, our DUM model distributed the shares in a way that always maximized the aggregated utility, as stated in the original problem formulation. Apart from the formal proofs provided in the Appendix concerning the maximization of the problem formulation, we provide, in Table 2, practical evidence that the

allocations found by our DUM solution do maximize the aggregated utility. More precisely, we compared the optimal allocation found, lets call it X^*, for a particular run, with different allocations, lets call them $X^{'}$, by giving/taking 0.1 resource shares to/from the AEs, in a way that the total amount of resources allocated did not change. Then we compared the value of the aggregated utility *U(X)* for the two allocations. As one can see in Table 2, the value obtained with the allocation X^* is always higher. That then further demonstrates that such an allocation, found by our DUM model, is, indeed, the one yielding to the maximum of the original optimization problem.

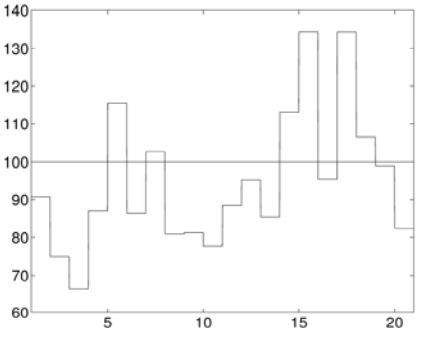

Fig. 11 Total amount of required servers over the overload scenario

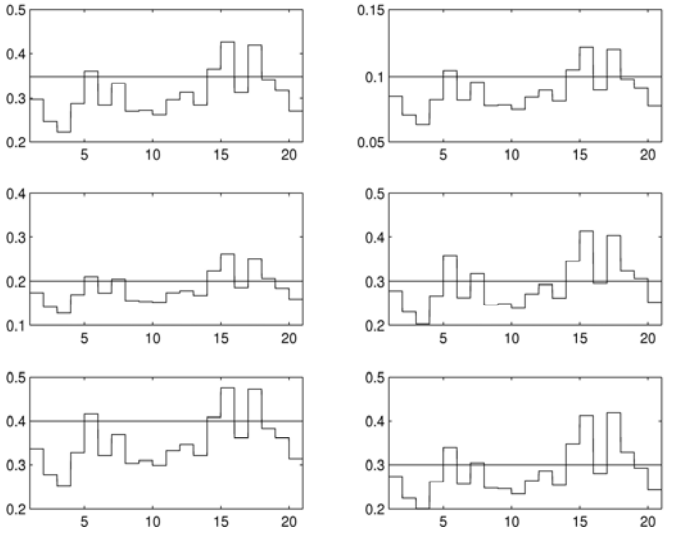

Fig. 12 EART of the AEs over the overload scenario

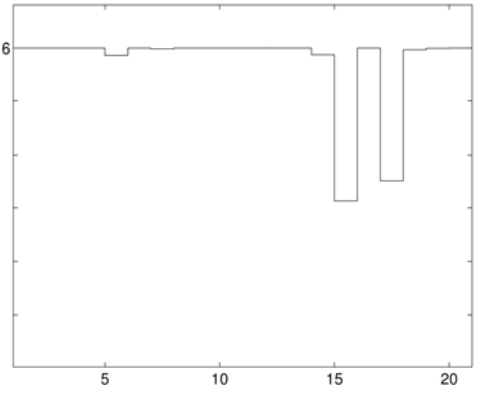

Fig. 13 Aggregate utility over the overload scenario

Table 2 Comparison of the values of the aggregate utility *U(X)*

$U(X^*) - U(X')$
6.767E-11
5.678E-11
1.191E-10
1.318E-10

5 Related Work

A number of solutions for performing self-optimizing resource management in shared resource pools 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. Unlike our solution, it relies 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. The solution is different from ours 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 focused on the control of multiple multicasts in P2P systems and does not apply to the problem formulation being used here.

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. As we showed, centralized and hierarchical solutions exist in this context, but none of the decentralized ones cope with the specific problem being studied here. That thus gives our DUM model an innovative feature. To the best of our knowledge, this is the first work to present a decentralized solution in the domain of shared resource pools.

An evaluation has been presented, through simulations, using 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 as well as the QoS parameters in the system vary, 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, we demonstrated that 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 $\alpha^j(t)$ 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 straightforwardly.

References

1. Rolia, J., Cherkasova, L., Arlitt, M., Machiraju, V.: Supporting application quality of service in shared resource pools. Communications of the ACM 49(3), 55–60 (2006)
2. Banga, G., Druschel, P., Mogul, J.C.: Resource containers: A new facility for resource management in server systems. In: Proceedings of the Third Symposium on Operating Systems Design and Implementation. USENIX Association, pp. 45–58 (1999)
3. Wang, X., Du, Z., Chen, Y., Li, S.: Virtualization-based autonomic resource management for multi-tier web applications in shared data center. Journal of Systems and Software 81(9), 1591–1608 (2008)
4. Guitart, J., Carrera, D., Beltran, V., Torres, J., Ayguade, E.: Dynamic CPU provisioning for self-managed secure web applications in smp hosting platforms. Computer Networks 52(7), 1390–1409 (2008)
5. Padala, P., Shin, K.G., Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A., Salem, K.: Adaptive control of virtualized resources in utility computing environments. In: Proceedings of the 2007 European Conference on Computer Systems, pp. 289–302. ACM Press, New York (2007)
6. Kephart, J.O., Das, R.: Achieving self-management via utility functions. IEEE Internet Computing 11(1), 40–48 (2007)
7. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer 36(1), 41–50 (2003)
8. Tesauro, G., Kephart, J.O.: Utility functions in autonomic systems. In: Proceedings of the First International Conference on Autonomic Computing, pp. 70–77. IEEE Computer Society, Washington (2004)
9. Bennani, M.N., Menasce, D.A.: Resource allocation for autonomic data centers using analytic performance models. In: Proceedings of the Second International Conference on Autonomic Computing, pp. 229–240. IEEE Computer Society, Washington (2005)
10. Tesauro, G., Walsh, W.E., Kephart, J.O.: Utility-function-driven resource allocation in autonomic systems. In: Proceedings of the Second International Conference on Autonomic Computing, pp. 342–343. IEEE Computer Society, Washington (2005)
11. Johansson, B., Adam, C., Johansson, M., Stadler, R.: Distributed resource allocation strategies for achieving quality of service in server clusters. In: Proceedings of the 45th Conference on Decision and Control, pp. 1990–1995. IEEE Computer Society, Washington (2006)
12. Bai, X., Marinescu, D.C., Boloni, L., Siegel, H.J., Daley, R.A., Wang, I.J.: A macroeconomic model for resource allocation in large-scale distributed systems. Journal of Parallel Distributed Computing 68(2), 182–199 (2008)
13. Kermarrec, A.M., Van Steen, M.: Gossiping in distributed systems. SIGOPS Oper. Syst. Rev. 41(5), 2–7 (2007)
14. Jelasity, M., Kowalczyk, W., van Steen, M.: An approach to massively distributed aggregate computing on peer-to-peer networks. In: Proceedings 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing, pp. 200–207. IEEE Computer Society, Washington (2004)
15. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science. IEEE Computer Society, Washington (2003)

16. Palomar, D.P., Chiang, M.: A tutorial on decomposition methods for network utility maximization. IEEE Journal on Selected Areas in Communications 24(8), 1439–1451 (2006)
17. Nowicki, T., Squillante, M.S., Wu, C.W.: Fundamentals of dynamic decentralized optimization in autonomic computing systems. In: Babaoğlu, Ö., Jelasity, M., Montresor, A., Fetzer, C., Leonardi, S., van Moorsel, A., van Steen, M. (eds.) SELF-STAR 2004. LNCS, vol. 3460, pp. 204–218. Springer, Heidelberg (2005)
18. Lewis, P.R., Marrow, P., Yao, X.: Evolutionary market agents for resource allocation in decentralised systems. In: Rudolph, G., Jansen, T., Lucas, S., Poloni, C., Beume, N. (eds.) PPSN 2008. LNCS, vol. 5199, pp. 1071–1080. Springer, Heidelberg (2008)
19. Maheswaran, R., Basar, T.: Nash equilibrium and decentralized negotiation in auctioning divisible resources. Group Decision and Negotiation 12(5), 361–395 (2003)
20. Raghavan, B., Vishwanath, K., Ramabhadran, S., Yocum, K., Snoeren, A.C.: Cloud control with distributed rate limiting. SIGCOMM Comput. Commun. Rev. 37(4), 337–348 (2007)
21. Masuishi, T., Kuriyama, H., Ooki, Y., Mori, K.: Autonomous decentralized resource allocation for tracking dynamic load change. In: Proceedings of the 7th International Symposium on Autonomous Decentralized Systems, pp. 277–283 (2005)
22. Nedic, A., Ozdaglar, A.: Distributed Subgradient methods for multi-agent optimization. IEEE Transactions on Automatic Control 54(1), 48–61 (2009)
23. Chen, M., Ponec, M., Sengupta, S., Li, J., Chou, P.A.: Utility maximization in peer-to-peer systems. In: Proceedings of the 2008 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp. 169–180. ACM, New York (2008)
24. Kuhn, M.: The Karush-Kuhn-Tucker theorem, `http://webrum.uni-mannheim.de/vwl/mokuhn/public/KarushKuhnTucker.pdf`
25. Weisstein, E.W.: Concave function, `http://mathworld.wolfram.com/ConcaveFunction.html`
26. Weisstein, E.W.: Convex function, `http://mathworld.wolfram.com/ConvexFunction.html`
27. Gallini, A.: Affine function, `http://mathworld.wolfram.com/A_neFunction.html`
28. Aron, M., Druschel, P., Zwaenepoel, W.: Cluster reserves: a mechanism for resource management in cluster-based network servers. In: Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp. 90–101. ACM Press, New York (2000)
29. Loureiro, E., Nixon, P., Dobson, S.: A fine-grained model for adaptive on-demand provisioning of CPU shares in data centers. In: Hummel, K.A., Sterbenz, J.P.G. (eds.) IWSOS 2008. LNCS, vol. 5343, pp. 57–108. Springer, Heidelberg (2008)
30. Lysne, O., Reinemo, S.A., Skeie, T., Solheim, A.G., Sodring, T., Huse, L.P., Johnsen, B.D.: Interconnection networks: Architectural chalenges for utility computing data centers. Computer 41(9), 62–69 (2008)
31. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: Above the clouds: A Berkeley view of cloud computing. Technical report, University of California at Berkeley (2009)
32. Dikaiakos, M.D., Katsaros, D., Mehra, P., Pallis, G., Vakali, A.: Cloud computing: Distributed internet computing for it and scientific research. IEEE Internet Computing 13, 10–13 (2009)

Appendix

A. Satisfiability to the Karush-Kuhn-Tucker (KKT) Conditions

Consider the following non-linear optimization problem:

$$\max_{x \in R^n} f(x)$$
$$\text{subject to}: g_i(x) \le 0, h_j(x) = 0$$

where $g_i(x)$, $i \in [1, m]$, are inequality constraints and $h_j(x) = 0$, $j \in [1, n]$, are equality constraints. It is known that if $f(x)$ and all $g_i(x)$ are concave and all $h_j(x)$ are affine functions, then, $\nabla L(X, \lambda) = 0$ is a sufficient condition for a maximum[24], i.e., it satisfies the KKT conditions. Translating that into our optimization problem, then:

$$f(x) = U(X)$$
$$h(x) = \sum_{a^i \in S(t)} X_i - K(t),$$

since we have no inequality constraints. We then start by showing that $U(X)$ is concave. For that, we can simply show that $-U(X)$ is strictly convex [25]. If a function $f(x)$ has a second derivative, for it to be strictly convex it is necessary and sufficient that $\forall x, f''(x) \ge 0$ [26]. We know that:

$$\nabla U(X) = \left(\frac{\partial U}{\partial X_1}, ..., \frac{\partial U}{\partial X_{n(t)}} \right)$$
$$\nabla - U(X) = -\left(\alpha^1(t) e^{-(\alpha^1(t) X_1)}, ..., \alpha^{n(t)}(t) e^{-(\alpha^{n(t)}(t) X_{n(t)})} \right)$$

Consequently

$$\nabla^2 - U(X) = -\left(-\alpha^1(t)^2 e^{-(\alpha^1(t) X_1)}, ..., -\alpha^{n(t)}(t)^2 e^{-(\alpha^{n(t)}(t) X_{n(t)})} \right)$$
$$\nabla^2 - U(X) = \left(\alpha^1(t)^2 e^{-(\alpha^1(t) X_1)}, ..., \alpha^{n(t)}(t)^2 e^{-(\alpha^{n(t)}(t) X_{n(t)})} \right) \quad , \quad (21)$$
$$\nabla^2 - U(X) \ge 0$$

which is what we wanted to show. Finally, showing that h(x) is affine is straightforward. Any affine function is of the following form [27]:

$$f(x_1, ..., x_n) = A_1 X_1 + ... + A_n X_n + b,$$

where A_i can be a scalar. Clearly, then, $h(x)$ is affine, since

$$h(x_1, ..., x_n) = X_1 + ... + X_n + (-K(t)),$$

where $A_1 = A_2 = ... = A_n = 1$. By showing that, then, we have that $\nabla L(X, \lambda) = 0$ is a sufficient condition for a maximum.

B. Existence and Uniqueness of the Maximum of *L(X, λ)*

Based on the optimization problem defined in Equation 4, we now show that, by solving $\nabla L(X, \lambda) = 0$ in the Optimization Model, we are not only finding a maximum of $L(X, \lambda)$, instead of a minimum or a saddle point, but also that such a maximum is unique, which then makes it the global maximum. For that, we start showing that $L(X, \lambda)$ is strictly concave. From the general non-linear optimization problem given in Appendix A, $L(X, \lambda)$ is strictly concave if $f(x)$, $g_i(x)$, and $h_j(x)$ are strictly concave[24]. From Equations 21, we already have that $f(x) = U(X)$ is strictly concave. Because we have no $g_i(x)$ constraints, it only remains for us to show that $h(x) = \Sigma_{ai \in S(t)} X_i - K(t)$ is strictly concave. For that, as in Appendix A, we have to show that $-h(x)$ is strictly convex, which can be done by checking if $\forall x, -h''(x) \geq 0$, if $-h''(x)$ is defined. We then have the following:

$$\nabla - h(x) = -(-K(t)_1, ..., K(t)_{n(t)})$$
$$\nabla^2 - h(x) = (0, ..., 0) \quad ,$$

which makes $h(x)$ strictly concave. Now that we know $L(X, \lambda)$ is indeed strictly concave, the following can be stated:

1. A stationary point of $L(X, \lambda)$, i.e., X', λ' such that $L(X', \lambda') = 0$, if any, is necessarily a maximum;
2. For a X', λ' such that $L(X', \lambda') = 0$, it can be said that X', λ' are unique,

which then ensures what we wanted to show.