Adaptive Management of Shared Resource Pools with Decentralized Optimization and Epidemics

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

Shared resource pools are facilities featuring a certain amount of resources which can be used by different applications. For managing resources in such pools, the demand of each application can be used. Such a demand, however, is driven by the workload, which varies over time. For that reason, adaptive approaches have been proposed for the management of shared resources pools. Whereas a number of solutions exist in this context, they are either not truly decentralized or do not apply to the problem we are dealing with. In this paper, we then present Darma*, an approach for managing shared resource pools in a truly decentralized, adaptive, and optimal way.*

1. Introduction

In shared resource pools, a collection of resources, e.g., servers, is available to be shared by different applications [1]. In many cases, these applications have QoS parameters that have to be met. Therefore, the amount of resources available to each of them should be such that their QoS parameters are met.

To distribute the resources in the pool, the demand of each application is used. The problem is that such a demand is driven by the workload, which varies over time [2][3]. Allocating resources using worst or average-case scenarios can lead to waste of resources and shares that do not keep up with the applications' QoS parameters [4]. It is more suitable to adapt the share of each application, over time, considering their QoS parameters and current workload. Usually, such an adaptation is performed not just to meet such parameters, but to do so in the best possible way. For that, the resource management task is typically viewed as an optimization problem, whose solution yields the best possible outcome.

This work is supported by UCD Ad Astra Scholarships

Many solutions for such a management in shared resource pools have been proposed. Some of them, however, employ central servers for distributing the resources. These solutions can perform well, but they suffer from scalability and fault-tolerance. Distributed solutions have also been proposed, but they have been modelled hierarchically, and thus, coordination is centralized. Whereas decentralized solutions in other contexts can be found, they are not applicable to the problem being studied here.

Based on that, in this paper we present *Darma*, **D**ecentralized and **A**daptive **R**esource **Ma**nagement, an approach for managing shared resource pools in a truly decentralized, adaptive, and optimal way. To the best of our knowledge, this is the first work in this context contemplating those features. The rest of this paper is then organized as follows: in Section 2 fundamental concepts are presented; *Darma* is presented in Section 3; in Section 4 we present an evaluation, using a data center scenario; related works in the area are presented in Section 5; finally, in Section 6, we present our conclusions and future directions of this work.

2. Basic Concepts

Since we aim at a decentralized solution, we view the system as a network of agents, where one agent can be reached by any other, directly or indirectly via a series of hops. We denote by $V(t)$ and $n(t)$ the set and number of agents in the system at time t. We denote by a^i an agent in the system, for $i \in [1, \ldots, n(t)]$ and $i \in \mathbb{N}$, and by $N^i(t)$ the set of neighbours of a^i at time t. Finally, each agent represents an application that consumes resources from the pool.

For optimization purposes, as proposed in [6], we assume each agent has a utility function $u^i(x)$, spec-

ifying how useful a particular resource share x from the pool is. From that, an aggregate utility $U(X)$ is derived, as follows:

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

where $X = \{X_1, \ldots, X_{n(t)}\}$ is an allocation vector and X_i is the share allocated to agent a^i. Finally, based on [7], the resource management task is modelled as the following optimization problem:

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

In the above, $K(t)$ is the amount of resources in the pool (e.g., 200 servers). The constraint then limits the sum of all shares received to such an amount.

3. Darma

Darma is actually composed by a set of mathematical models, formalizing the entire resource management process. These models are then presented next.

3.1. Optimization Model

The Optimization Model defines how to solve the optimization problem defined in a decentralized way. For that, each agent a^i is assigned the following utility function $u^i(x)$:

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

where x is the amount of resources being allocated to a^i and $\alpha^i(t)$ is a parameter that indicates a^i's demand at time t. The smaller it is, the greater is the agent's demand. This parameter can change over time, but it should remain the same over the resource management process itself. The reasons for choosing such a utility function is that it will allow us to break down the optimization problem into separate models that calculate each agent's optimal share. Like ours, other works have also used specific utility functions for different purposes [7][8].

From $u^i(x)$, the optimization problem is broken down. To this end, it is reformulated as follows:

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

In this formulation, $L(X, \lambda)$ is the lagrangian of the original optimization problem in 2, being defined as:

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

We break down the problem above by solving $\nabla L(X, \lambda) = 0$ and isolating X_i, which yields in:

$$X_i = \frac{\ln \alpha^i(t) - \ln \lambda}{\alpha^i(t)}. \tag{6}$$

Equation 6 then allows each agent a^i to find its share in a way that it solves the optimization problem in 2. To calculate X_i, besides their own $\alpha^i(t)$, agents also need the value of $\ln \lambda$, which is the global information *Darma* relies on. Such a value is also found when the problem in 4 is broken down, resulting in:

$$\ln \lambda = \frac{\left(\sum_{i=1}^{|X|} \frac{\ln \alpha^i(t)}{\alpha^i(t)} \right) - K(t)}{\sum_{i=1}^{|X|} \frac{1}{\alpha^i(t)}}. \tag{7}$$

With that, the original problem has been reduced to finding the value of $\ln \lambda$ in a decentralized way. It is because $\ln \lambda$ depends on the α of all agents that we use epidemic algorithms in *Darma*. With such algorithms, wen can have every $\alpha^i(t)$ to be disseminated in such a way that it reaches every other agent in the system. Once that happens, they are then ready to calculate $\ln \lambda$ and from that their optimal share.

3.2. Epidemic Model

In this model we define the methods for disseminating the α values so that $\ln \lambda$ can be calculated. In principle, $\ln \lambda$ can be calculated using solutions for computing aggregates. In our case, however, these approaches would not be very feasible in practice. That would be due to the specificity of $\ln \lambda$ and our requirements in terms of precision. We restrict ourselves from a further analysis about this for now, but provide, in Section 5, a more in-depth discussion.

For disseminating the α values we use an *Anti-entropy* epidemic approach [9]. It consists, basically, on synchronizing replicas held by two nodes, by means of pushing and/or pulling updates that are missing in each of them. By having all nodes to perform such a synchronization with a randomly chosen neighbour, over time, the replicas can converge to the same. As we will show, such an approach fits perfectly our needs, and that has been the reason for choosing it in *Darma*.

For that, we define firstly the format of the messages used in the dissemination, as $< \rho^i, \alpha^i(t) >$, where ρ^i is the identifier of agent a^i, and $\rho^i \in \mathbb{N}^*$. Because two α values can be the same, the identifier of the agents will allow them to determine whether they already received a particular α or not. Based on that, let $o^i(t)$ be the set containing the identifiers of the agents from which a^i has received an α value. Also, let $m^i(t)$ be a multiset that holds all α values an agent a^i received until time

t. We then define $o^i(t)$ as the replica that each agent will synchronize with the others. Consequently, $m^i(t)$ will be synchronized too.

A traditional implementation of an epidemic algorithm, however, might not perform well when replicas are large, which would be our case eventually. If an agent chooses a neighbour such that their replicas are the same, they will waste time comparing two large and similar replicas. As proposed in [9], a checksum can be used for that. In this case, agents would first analyze the checksum, only comparing the replicas if their checksums are different. Using the agents' identifiers, we define the checksum of the replica of agent a^i to be represented by a function $k^i(t)$, defined as:

$$k^i(t) = \sum_{\rho \in o^i(t)} 2^\rho, \tag{8}$$

The $k^i(t)$ function guarantees a unique signature in terms of the agents from which another agent a^i has received α values. That alone would already enable us to check whether two replicas are similar or not, just by checking if their checksum $k^i(t)$ match. Such a function, however, enable us to go further. Note that, if the checksum of two replicas A and B do not match, an agent still do not know if: 1) A has values that B does not; 2) B has values that A does not; or 3) A and B have values that the other does not. Further verification will be necessary just for that. The $k^i(t)$ function enables us to not just check if two replicas match; it can also tell us if a replica A is a subset of another replica B, which can be done with:

$$\sum_{a \in A} 2^a \ \& \ \sum_{b \in B} 2^b, \tag{9}$$

where $\&$ is a bitwise AND operation. If the result is $\sum_{a \in A} 2^a$, then $A \subset B$. Similarly, the $+$ and $-$ operators can be used to obtain the union and difference of two replicas.

We combine such properties of the $k^i(t)$ function for defining how an agent a^i picks neighbours for disseminating α values. When doing that, a^i will choose a random neighbour among those who have not received at least one of the α it has. More precisely, to choose such a neighbour, an agent a^i uses the $G^{ij}(t)$ function below, for each neighbour a^j:

$$\begin{aligned} G^{ij}(t) = \left(k^i(t) + 2^{\rho^i}\right) - \left(2^{\rho^j} \ \& \ k^i(t)\right) - \\ \left(k^j(t) \ \& \ \left(k^i(t) + 2^{\rho^i}\right)\right), \end{aligned} \tag{10}$$

which represents all the ρ and α values that a^i can disseminate to a^j. If no such a pair exists, then $G^{ij}(t)$ is 0. Therefore, every neighbour for which $G^{ij}(t)$ is not 0 is eligible to be chosen for the dissemination.

The point of using this approach, and not a purely random one, is that the time to deliver all α is improved, since agents only choose neighbours such that at least one pair of ρ and α could be disseminated. We provide evidence of that in Section 4.

When a neighbour is chosen, a push and a pull are performed. For the *push*, all the ρ and α that a^i received, including ρ^i and $\alpha^i(t)$, such that the neighbour has not, are pushed to it. For the *pull*, a^i pulls from the neighbour all the ρ and α that it received, including the neighbour's ρ and α value, but a^i has not. That is done by checking if

$$2^{\rho^k} \ \& \ G^{ij}(t) = 2^{\rho^k} \tag{11}$$

for the push, for all $\rho^k \in o^i(t) \cup \{\rho^i\}$, and

$$2^{\rho^k} \ \& \ G^{ji}(t) = 2^{\rho^k} \tag{12}$$

for the pull, for all $\rho^k \in o^j(t) \cup \{\rho^j\}$. Every pair $< \rho^k, \alpha^k(t) >$ such that ρ^k satisfies one of the above conditions is then pushed or pulled, as appropriate. Because the agents always choose a neighbour who has not received some of the α it has, if any, it is guaranteed that all α are delivered to all agents. Results showing such a completeness of the Epidemic Model are provided in the evaluation section.

3.3. Consensus Model

Besides disseminating the α values, agents still need to know *when* they have received *all* of them. This is necessary because it will tell them when they should stop trying to disseminate information and waiting for incoming messages, thus using the α they received to compute $\ln \lambda$ and then their own share. The Consensus Model then defines how agents will interact in order to achieve a consensus as to when the dissemination is finished, in a decentralized and fault-tolerant fashion.

To this end, we designed a mechanism where agents exchange signaling information, relying solely on their neighbourhood. For that, we define a time-varying value $x^i(t)$, held by each agent, defined as:

$$x^i(t) = \frac{\left(k^i(t) + 2^{\rho^i}\right) + \sum_{a^j \in N^i(t)} x^j(t)}{|N^i(t)| + 1}, \tag{13}$$

where $x^i(0) = 2^{\rho^i}$. The value of $x^i(t)$ gives agents an idea of how much information has been disseminated to their neighbourhood. As it increases, so does the amount of α disseminated. Notice that it only depends on each agent's neighbours' $x^i(t)$, thus being totally decentralized.

The rationale behind $x^i(t)$ is that we expect it to converge once all α are disseminated. We call such

a state *equilibrium*. More precisely, it can be shown that all $x^i(t)$ converge to $2^{\rho^i} + \sum_{\rho \in o^i(t)} 2^\rho$ at the end of the dissemination, and that it only happens at that point. That then guarantees that equilibrium will only be reached when every α reaches every other agent.

Agents, however, cannot rely solely on the convergence of $x^i(t)$ for determining whether all α have been disseminated or not. Because of network delays, the $x^i(t)$ of an agent could get stuck at a particular value for some time. In turn, that would lead the agent to believe that the dissemination is finished, when in fact it is not. For that reason, from $x^i(t)$, we define the following difference function $d^i(t)$:

$$d^i(t) = \sum_{a^j \in N^i(t)} |x^i(t) - x^j(t)| \qquad (14)$$

The importance of the difference function comes from the fact that all $x^i(t)$ converge to the same value at the end of the dissemination. That consequently causes all $d^i(t)$ to go to 0, which is the value that agents have to look after for determining when the dissemination is finished. That is guaranteed to happen only at the end of the dissemination. Even if $d^i(t)$ gets stuck at any value other than zero, due to delays, agents will know that the dissemination is not over yet.

Apart from decentralization, $x^i(t)$ and $d^i(t)$ still provide support for system changes. The distributed systems we are dealing with will most likely change over time, in terms of agents joining and leaving. Agents might join because a new customer application has been added to the system. Similarly, an agent might leave because its application has been dropped, or even because the agent, or the node where it is running, crashed or was disconnected from the network. We analyze how *Darma* supports such changes from two scenarios; when they happen 1) *in between* two resource management processes and 2) *during* a resource management process.

In the first scenario, the convergence of the system is not affected. This is because our Consensus Model does not rely on any global information about the number of agents in the system. Instead, it relies solely on the neighbourhood of each agent. Any changes in the set of agents are then automatically picked up by the Consensus Model once the resource management process starts. This would ensure that *planned* changes in the system structure could be made in between two resource management processes, without the need to restart any part of the system.

In the second scenario, a few issues might come up. In terms of agents joining, convergence cannot be guaranteed. If an agent joins before any of the $d^i(t)$ converges, the system would continue executing, eventually reaching equilibrium. If an agent joins after at least one $d^i(t)$ converges, the system will not reach equilibrium. As for agents leaving the system, convergence is still guaranteed, as long as the remaining topology consists of single graph. This scenario of agents joining and leaving during a resource management process is more typical of *unplanned* changes. Examples include, as we mentioned, the crash of an agent. We believe, then, that unplanned additions of applications, and consequently agents, are not to happen in practice. Therefore, without any loss of flexibility, we assume that applications can only join the system in between two resource management processes.

Finally, we should also expect topology changes. Due to the fact that our Consensus Model relies only on the neighbourhood of the agents, it is able to guarantee that the system will reach equilibrium in face of topology changes, even if they happen during the resource management process. Again, any changes in the topology should result in a single graph. As we will show in the evaluation presented in Section 4, all properties of the Consensus Model hold in practice, even in the face of the system changes supported.

3.4. Demand Model

As defined in the Optimization Model, $\alpha^i(t)$ indicates an agent's demand for resources at time t. Such a demand is determined by the workload, which varies over time. We then need a way of mapping a given workload to $\alpha^i(t)$. For that reason, we defined the Demand Model. The problem, though, is that such a mapping is very application specific. Consequently, there is no way the Demand Model could employ a general model of calculating $\alpha^i(t)$. Therefore, the Demand Model is actually a hot spot of *Darma*, which should be extended for each application scenario. For that, then, we define the following functions: 1) $w^i(t)$: expected workload at time t of the application associated with agent a^i; 2) $e^i(w)$: returns the $\alpha^i(t)$ representing the demand of agent a^i given the workload w. As we will show in the evaluation, with such a model, we can map workload and demand features from different scenarios into *Darma*, straightforwardly.

4. Evaluation

In this section we evaluate *Darma* using a data center scenario where a number of Application Environments (AEs) are deployed, as in [7], each processing one type of transaction. Our evaluation scenario consists of allocating servers from the data center to the AEs, where each AE is represented by an agent. For

the purposes of this evaluation, we assume the resource management processes happen at pre-defined points over time. In a real-world setting this could represent, for example, different hours of the day, on which some re-allocation of the servers would take place.

Each AE has different workload values at different points in time (i.e., $w^i(t)$ function of the Demand Model), in terms of requests per second. For our experiments, the values for the $w^i(t)$ function of each AE have been obtained from the analytical data of different web sites. The QoS parameter of each AE is defined in terms of a Target Response Time (TRT) that must be satisfied for the transactions they process.

From that, and based on the model proposed in [7], we define $r^i(s, w)$, the Expected Average Response Time (EART) of an AE, representing the response time obtained given a workload w and a number of servers s allocated to it. We define $r^i(s, w)$ as $r^i(s, w) = \frac{w.c^i}{s}$, where c^i is the CPU time of the type of transaction processed by AE i (in seconds). From $r^i(s, w)$, we define $q^i(w)$, the number of servers needed by an AE so that its TRT can be met, as follows: $q^i(w) = \frac{w.c^i}{T^i}$, where w and c^i are as in $r^i(s, w)$ and T^i is the AE's TRT. With that, we then define the mapping function $e^i(w)$ of the Demand Model as $e^i(w) = -\frac{\ln(1-H)}{q^i(w)}$, 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. As defined in the Demand model, $e^i(w)$ then returns the demand $\alpha^i(t)$ of an AE. Using this data center model, we performed several experiments, whose results are presented next.

4.1. Allocation

Firstly, we analyzed *Darma* in terms of allocations. For that, our data center scenario employed 145 servers. Because the focus here is on the allocations found and their outcome, we assumed that six AEs are deployed in the data center. Simulations with larger-scale systems are presented in the following subsections. Each simulation, for these allocation experiments, ran over 20 iterations. The values for the CPU times (c^i in $q^i(w)$) for these experiments were based on [7]. Due to space constraints, however, we omitted such values from this paper.

The workload of each AE is as illustrated in Figure 1. From that, agents use the mapping function $e^i(w)$ to find their demand $\alpha^i(t)$, which are illustrated in Figure 2. Note that $\alpha^i(t)$ varies oppositely to the way the workload does. That matches the definition of the agent's utility, i.e., the higher the workload is, the higher is the demand, and thus the smaller is $\alpha^i(t)$.

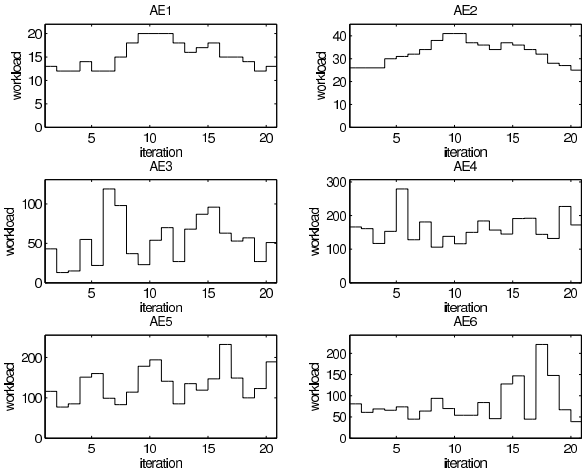


Figure 1: $w^i(t)$ function of each AE

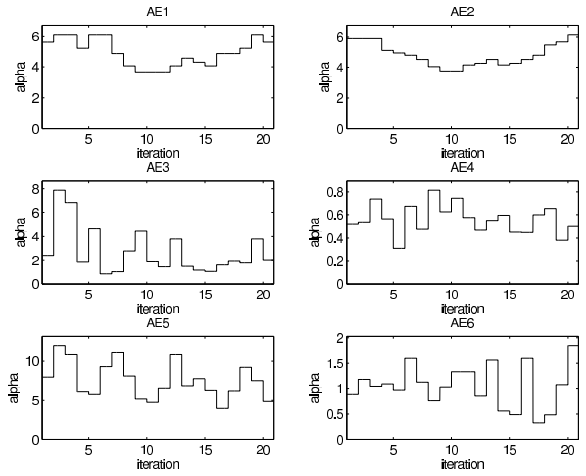


Figure 2: $\alpha^i(t)$ of each AE

As a result of the $\alpha^i(t)$, the shares for each AE ended up as in Figure 3. It is easy to see that the shares vary similarly to the way the workload does. That shows the ability of *Darma* to capture the resource demands properly and act towards the optimal distribution. Consequently, the response times for the transactions of each AE are always below their TRT. This is illustrated in Figure 4, where the dashed line in each graph represents the TRT of the AE.

4.2. Dissemination

We have also performed experiments to evaluate how quick the α are disseminated throughout the system. Firstly, we have compared our checksum dissemination approach against a purely random one, where agents choose random neighbours to disseminate to in

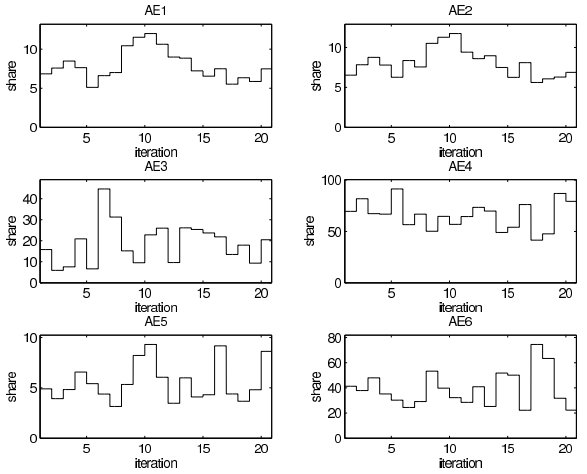


Figure 3: Shares of each AE

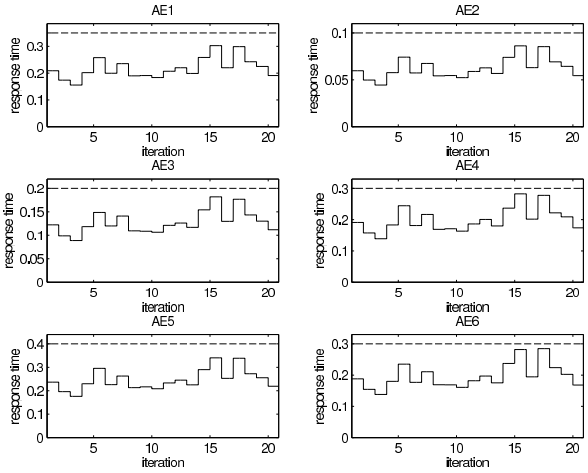
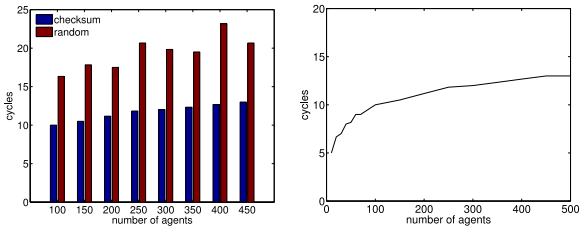


Figure 4: EARTs of each AE

a "blind" way, i.e., without considering if they have unsynchronized replicas. This comparison has been performed in terms of the *number of cycles* that it takes to deliver all α. A cycle, in this case, is considered as one step from each agent sequentially, where a step means an agent choosing a neighbour to disseminate to. Even though the system is not synchronous, this provides a common ground for analyzing not only the dissemination but also the convergence in *Darma*.

A purely random approach leads to situations where the neighbour chosen by an agent is such that no pair ρ^i and $\alpha^i(t)$ can be exchanged, but there are neighbours satisfying such a property. Consequently, more cycles would be required to disseminate all α. Looking at Figure 5a, one can see that such a claim holds. Each group of bars displays the number of cycles for disseminating all α, using our checksum approach and a purely random one. Clearly, our approach needs always fewer cycles, thus making it more effective.

To demonstrate further effectiveness of our Epidemic Model, we have analyzed how the number of cycles to deliver all α vary as system scale. As one can see on Figure 5b, the number of cycles grows logarithmically, even when system scale by the hundreds. That then shows how well *Darma* can perform in large-scale systems. Time values for each cycle, however, could not be measured appropriately, since the simulation ran on a single machine, thus with no true parallelism.



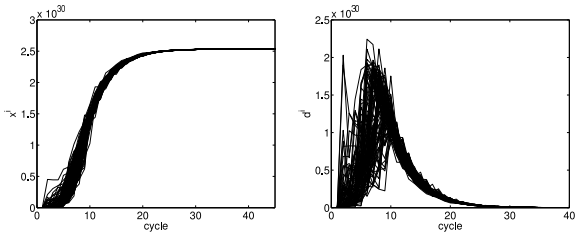
(a) Checksum Vs. purely random (b) Cycles to deliver all α

Figure 5: Results of the dissemination

4.3. Convergence

Our last experiments analyze the convergence properties of the Consensus Model. In these experiments, we simulated systems of smaller scale so as to ease the understanding of the plots. As we presented, it is expected that all $x^i(t)$ converge to a specific value when the dissemination is finished. We demonstrate in Figure 6a that such a property holds, where each $x^i(t)$ of a system with 100 agents is plotted. Note that, as the dissemination approaches the end, at the 10th cycle, $x^i(t)$ starts to converge exponentially. At some point, that gives way to a slower rate, giving $x^i(t)$ a "s-shaped" form, characteristic of epidemic-based approaches. Consequently, $d^i(t)$ also converges as expected, i.e. to 0, which is illustrated in Figure 6b.

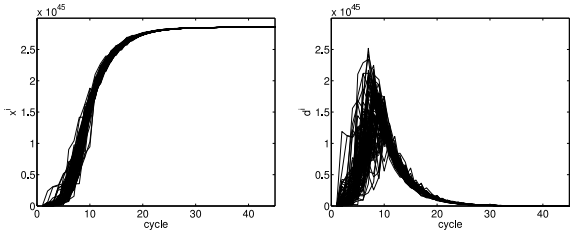
Finally, we demonstrate the convergence of $x^i(t)$ and $d^i(t)$ in face of node crashes and topology changes. For the former, we simulated a system with 150 agents, where one third of them crash at the 10th cycle. As one can see in Figure 7, the convergence of $x^i(t)$ and $d^i(t)$ is not compromised. Then, we have simulated a system with 100 agents where the topology changes at cycles 3, 10, 15, and 20. In each case, the topology changes in such a way that it is completely different from the cycle before. The results are presented in Figure 8, and again, the convergence of $x^i(t)$ and $d^i(t)$ is not



(a) (b)

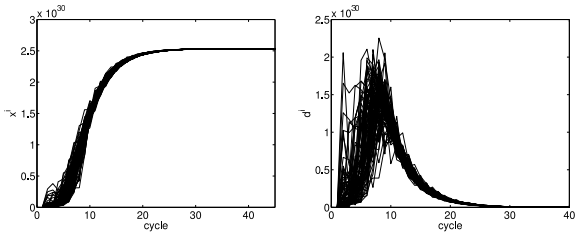
Figure 6: Convergence of $x^i(t)$ and $d^i(t)$ for 100 agents

compromised. In particular, notice that $x^i(t)$ converges as in Figure 6a, where no topology change happened, thus clearly demonstrating the ability of *Darma* to handle situations where topology changes.



(a) (b)

Figure 7: Convergence of $x^i(t)$ and $d^i(t)$ for 150 agents. One third of the agents crash at cycle 10



(a) (b)

Figure 8: Convergence of $x^i(t)$ and $d^i(t)$ for 100 agents, under changing topologies

5. Related Work

Many solutions for performing resource management have been proposed. Many of them, however, employ central servers [2][7][4]. They can perform well, but suffer from scalability and fault-tolerance. Distributed solutions have also been proposed. An example are market-based approaches. However, they either do not focus on optimization [10][11] or employ central entities called *brokers* [8]. The decomposition methods presented in [12] are another distributed solution. They employ a messaging scheme relying on a central entity, and therefore are not suitable in our case. A truly decentralized solution is presented in [13]. This solution is modelled differently though, in that resource providers, and not consumers, solve the optimization problem. Also, it is focused on server allocation, whereas we aimed at a more general approach.

Other works in related areas include [14], where gossiping is used to allow a set of P2P-connected traffic limiters to control the bandwidth they use. It does not focus on optimal allocations though. In [15], subgradient methods are used to optimize the aggregate of a set of agents' cost function. The solution does not incorporate resource constraints and network delays, limiting its applicability in practical scenarios. In [16], a decentralized utility maximization model is proposed, but it focus on controlling multicasts in P2P systems.

The problem of computing $\ln \lambda$ can, indeed, be seen as a *Node Aggregation* problem. From the perspective of *Darma*, some solutions in this context are limited to aggregates that do not fully fulfill our needs, e.g., AVERAGE [17]. Others compute the aggregates in a very general way [18], thus not being directly applicable to our $\ln \lambda$, requiring different runs of the aggregate algorithm, in our particular case, thus affecting scalability. In [19] the scalability problem seems solved; however, the mechanisms employed for deciding when an aggregate computation is finished do not guarantee that the exact value will be reached. A more general-purpose solution is presented in [20]. It is focused on providing less precision so as to lessen messaging. That does not suit us, since imprecise aggregates will generate sub-optimal allocations.

6. Conclusions

In this paper we presented *Darma*, an approach for managing shared resource pools in a truly decentralized, adaptive, and optimal way. As we showed, *Darma* is based on a set of mathematical models, which together formalize the resource management task. Lagrange multipliers have been used for providing decentralized optimization, whereas epidemic and consensus models were employed for disseminating information used when calculating the optimal shares. All that makes *Darma* truly decentralized, delay and fault tolerant, also supporting topology changes. The addition of applications is supported, in between two

resource management processes, which, as we discussed, is sufficient for real-world scenarios.

An evaluation has been presented, demonstrating how to use *Darma* to build an adaptive data center. As we showed, only a few functions had to be provided. Extending *Darma* for a specific scenario is thus straightforward. In terms of allocations, we showed that *Darma* was able to deliver shares that always met the policies of all AEs. Also, we demonstrated that the properties for workloads, demand, and the $\alpha^i(t)$ of the agents hold in practice. Aspects of dissemination have also been analyzed. With that, we were able to show not only that our checksum approach does improve the number of cycles to deliver all α, but also that such a number grows slowly as system size increases. The convergence of the agents has been analyzed, demonstrating that they do reach equilibrium, even when facing topology changes and agents leaving the system. Finally, as future work, we will be looking at applying *Darma* to other shared resource pool scenarios, to have an insight of how general it really is.

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] 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 Syst. and Softw.*, vol. 81, no. 9, pp. 1591–1608, 2008.

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

[4] 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 *Proc. of the 2nd European Conf. on Computer Systems 2007*. ACM, 2007, pp. 289–302.

[5] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, January 2003.

[6] 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.

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

[8] 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," *J. Par. Distr. Comp.*, vol. 68, no. 2, pp. 182–199, 2008.

[9] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic algorithms for replicated database maintenance," in *Proc. of the 6th annual ACM Symp. on Principles of Distributed Computing*. New York, NY, USA: ACM Press, 1987, pp. 1–12.

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

[11] 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.

[12] D. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization," *IEEE J. Sel. Are. Comm.*, vol. 24, no. 8, pp. 1439–1451, 2006.

[13] 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 Conf. on Decision and Control*. IEEE Computer Society, December 2006, pp. 1990–1995.

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

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

[16] M. Chen, M. Ponec, S. Sengupta, J. Li, and P. A. Chou, "Utility maximization in peer-to-peer systems," in *Proc. of the 2008 ACM SIGMETRICS Int. Conf. on Measurement and Modeling of Computer Systems*. New York, NY, USA: ACM, 2008, pp. 169–180.

[17] M. Mehyar, D. Spanos, J. Pongsajapan, S. H. Low, and R. M. Murray, "Asynchronous distributed averaging on communication networks," *IEEE/ACM Trans. Netw.*, vol. 15, no. 3, pp. 512–520, 2007.

[18] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *Proc. of the 44th Annual IEEE Symp. on Found. of Comp. Science*. Washington, DC, USA: IEEE Computer Society, 2003.

[19] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Trans. Comp. Syst.*, vol. 23, no. 3, pp. 219–252, August 2005.

[20] M. Haridasan and R. van Renesse, "Gossip-based distribution estimation in peer-to-peer networks," in *Proc. of The 7th Int. Work. on Peer-to-Peer Syst.*, 2008.