Decentralized and Optimal Control of Shared Resource Pools

EMERSON LOUREIRO, University College Dublin
PADDY NIXON, University College Dublin
SIMON DOBSON, University College Dublin

Resource pools are collections of computational resources (e.g., servers) which can be used by different applications in a shared way. A crucial aspect in these pools is to allocate resources so as to ensure their proper usage, taking into account workload and specific requirements of each application. An interesting approach, in this context, is to allocate the resources in the best possible way, aiming at optimal resource usage. Workload, however, varies over time, and in turn, resource demands will vary too. To ensure that optimal resource usage is always in place, resource shares should be defined dynamically and over time. It has been claimed that utility functions are the main tool for enabling such self-optimizing behaviour. Whereas many solutions with this characteristic have been proposed to date, none of them presents true decentralization within the context of shared pools. In this paper, we then propose a decentralized model for optimal resource usage in shared resource pools, providing practical and theoretical evidence of its feasibility.

Categories and Subject Descriptors: G.1.6 [**Numerical Analysis**]: Optimization—*constrained optimization*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*multiagent systems*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications*

General Terms: Algorithms

Additional Key Words and Phrases: Decentralized algorithms, decentralized optimization, resource management, utility maximization, resource pools

ACM Reference Format:
Loureiro, E., Nixon, P., Dobson, S. 2012. Decentralized and Optimal Control of Shared Resource Pools. ACM Trans. Autonom. Adapt. Syst. 9, 4, Article 39 (March 2012), 32 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Resource pools are collections of computational resources (e.g., servers) which can be used by different applications in a shared way [Rolia et al. 2006]. The goal with that is to accommodate the workload of each application (a resource consumer, or consumer, in more general terms), by allocating resources from the pool to them. This is possible through the use of *Resource Containers* [Banga et al. 1999], abstractions which encapsulate a certain amount of resources, and can be put together into *resource shares* (or shares) to be made available to a specific consumer. In this case, a collection of shares is referred as a *resource allocation*, or simply allocation. Usually, consumers have specific requirements, stating properties that they would like to be met. A crucial aspect in shared pools is thus allocate resources ensuring their proper usage, taking

This work is supported by UCD Ad Astra Scholarships.
Authors' address: E. Loureiro, P. Nixon, and S. Dobson, Systems Research Group, School of Computer Science and Informatics, University College Dublin, Belfield, Dublin 6, Dublin, Ireland.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

into account workloads and requirements defined [Rolia et al. 2006]. Together, they determine the *resource demand* of each consumer.

In this context, an interesting approach is to allocate the resources in the best possible way, in other words, to perform the so-called *self-optimization* aspect of autonomic computing systems [Kephart and Chess 2003]. For the purposes of this work, "best" means an allocation which will ensure optimal resource usage, with a best-effort approach to satisfy specific consumer requirements. Finding such a allocation consists, basically, on modelling it in terms of an optimization problem, and eventually solving this problem. More formally, the allocation is viewed as a vector of real values X, each representing the share of a consumer. A function of X is defined - e.g., $f(X)$ - which is internally modelled in terms of individual resource demands and is set as the target of an optimization problem whose constraints represent resource availability. What we are interested is then on a vector X^* that maximizes f. That is regarded as our best allocation, which not only ensures optimal resource usage, given the availability, but also does its best to satisfy consumer requirements, since they are the driving forces behind $f(X)$.

The problem, in this case, is that the workload is likely to vary over time, causing resource demands to vary in turn [Wang et al. 2008; Guitart et al. 2008; Gmach et al. 2008]. Consequently, static allocations would not guarantee that optimal resource usage is always in place [Padala et al. 2007]. For that, allocations should be defined dynamically, over time. We refer to this process as *resource management process*. Many solutions for performing such a resource management in shared pools have been proposed to date. Some of them, however, employ centralized architectures. These solutions can scale well depending on system size. Still, fault tolerance is an issue, since the crash of the centralizer compromises the entire system. On the other hand, some solutions tackle the problem in a distributed way, by having different entities in the system coordinating so as to determine the optimal resource usage. Still, these solutions are organized hierarchically, where coordination is done at the top of the hierarchy. From the fault tolerance point of view, they present problems similar to centralized architectures. In truly decentralized solutions, all entities have the same role, interacting with each other and converging to the optimal solution using only a limited (e.g., local) view of the system. Decentralization is, consequently, key to provide distributed systems with improved fault-tolerance. Even though decentralized solutions exist in similar contexts, they are not applicable to the problem being studied.

To the best of our knowledge, this is the first work to present a decentralized solution for optimal resource usage in shared pools, providing mathematical and algorithmic models as well as practical and theoretical evidence of its feasibility. In this paper, we then present *Darma*, **D**ecentralized and **A**daptive **R**esource **Ma**nagement. This work is an extension of previous research we have done [Loureiro et al. 2009] [Loureiro et al. 2010], and here we provide a more extensive formalization and sets of results, as well as a theoretical analysis. The rest of this paper is organized as follows: in Section 2, a background on shared pools and related works is presented. Core concepts of *Darma* are presented in Section 3. In Section 4, *Darma* is presented, with convergence aspects analyzed in Section 5. An evaluation is presented in Section 6, and finally, in Section 7, we conclude the paper.

2. BACKGROUND

In this section we present an overview of shared resource pools and a discussion of the most relevant related works in the area.

2.1. Shared Resource Pools

As we mentioned, shared resource pools allow concurrent access to computing capacity [Rolia et al. 2006]. Such a capacity is usually an aggregation of a collection of resources, which can be split into shares and allocated to different resource consumers. In more practical terms, each resource share from the pool is made up of one or more resource containers, e.g. a virtual machine or a virtual disk, being associated with a particular resource consumer. This same abstraction can be found under different terminologies, e.g., Cluster Reserve [Aron et al. 2000], Server [Loureiro et al. 2008], Application Environment [Bennani and Menascé 2005], but for the purposes of *Darma*, the term resource share will be used.

Such a resource pool model can be mapped into many real-world scenarios. An example of that is illustrated in Figure 1. In this figure, a collection of servers, i.e., the pool, is split into a set of resource shares (the solid lines wrapping different sets of servers). In turn, each share is assigned to their respective application (APP), i.e., resource consumer. The same idea applies, for example, to a Distributed Rate Limiting scenario [Raghavan et al. 2007]. In this case, the pool is composed by the bandwidth capacity available. This capacity is partitioned into different shares, which are assigned to the traffic limiters (i.e., consumers). This way, traffic limiters can continue to serve their network flows, without, however, overusing a specific bandwidth capacity. We show in Section 6 that these ideas hold in practice, when modelling a real-world scenario as a shared resource pool using *Darma*.

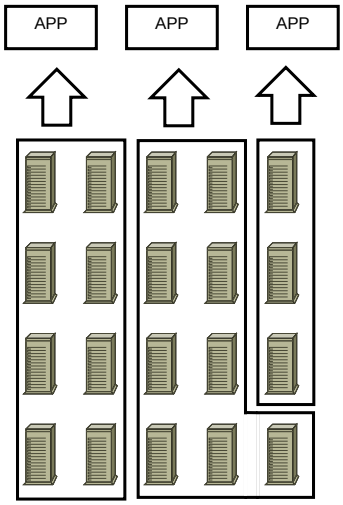


Fig. 1: Resource pool in a data center scenario

2.2. Related Work

A number of solutions for performing resource management in shared pool has been proposed to date. Many of them, however, approach the problem using centralized models [Wang et al. 2008; Bennani and Menascé 2005; Boutilier et al. 2003; Paton et al. 2009; Samaan 2008; Urgaonkar et al. 2008; Aron et al. 2000; Padala et al. 2007; Xu et al. 2008]. In this case, a central authority is in charge of not only fetching resource demands across the system, but also deciding the resource shares each participating entity should receive. Even though these solutions can perform well, fault-tolerance is

an issue. More precisely, a crash of the central authority, or even lack of communication with it, compromises the entire resource management process.

Other solutions employ a more distributed computation model, in which the processing for finding allocations is spread throughout different entities in the system. In [Bai et al. 2008], an example of such a solution is proposed, using market agents. Still, a centralizing entity, called broker, is employed. Other distributed solutions employ a hierarchy, like the decomposition methods presented in [Palomar and Chiang 2006]. Such methods decompose an optimization problem into smaller ones, where each of them can be further decomposed, forming a hierarchy. Other solutions employing a hierarchical model would include [Nowicki et al. 2005; Piovesan et al. 2008; Byde et al. 2003]. Coordination in these solutions, however, happens at the root of the hierarchy, and thus in a centralized way. From a fault-tolerance perspective, they face the same problems as their centralized counterparts.

Truly decentralized solutions exist in similar domains. In [Lewis et al. 2008; Maheswaran and Başar 2003], for example, market agents are used, but the problem studied is not focused on optimal resource usage. In [Johansson et al. 2006], a decentralized algorithm is proposed for allocating servers to a set of class of services. Its focus, however, is on providing near-optimal solutions and specifically on the allocation of servers. Gossiping is used in [Raghavan et al. 2007] to allow a set of network traffic limiters, connected in a P2P fashion, to control the bandwidth they use. Despite the complete decentralized control presented by the solution, it does not tackle the optimal resource usage problem. In [Masuishi et al. 2005; Batouma and Sourrouille 2010], solutions employing a borrowing mechanism are proposed, allowing nodes to use resources from each other when needed. Whereas decentralization is successfully achieved, these solutions do not focus on the self-optimization aspect.

In terms of decentralized optimization, in [Nedic and Ozdaglar 2009], subgradient methods are used to optimize the aggregate of a set of agents' cost function. The solution, however, does not support resource constraints, network delays, or fault-tolerance, limiting its applicability in practice. In [Chen et al. 2008], a decentralized utility maximization model is proposed. It is focused on the control of multiple multicasts in P2P systems, and thus does not apply to the problem we are dealing with. Distributed Constraint Optimization (DCOP) is a technique on which a number of agents assign values to a set of variables, such that a function of these variables is maximized/minimized, also satisfying a set constraints [Chechetka and Sycara 2006; Petcu and Faltings 2005]. Given the problem being studied here, mainly the nature of its search space, a more analytical-oriented approach is required though.

3. FUNDAMENTALS

In this section we present the basic concepts involved in our solution. Most importantly, a formalization of the resource management problem is presented.

3.1. System Model

Given the decentralization of *Darma*, we view the set of resource consumers as a multiagent system, each agent representing a consumer. In this case, agents are connected in an arbitrary topology and an agent can be reached by others, directly or indirectly. Also, agents are homogeneous, meaning that they have the same roles in the system. We model such a multiagent system as a graph $S(t)$, defined as:

Definition 3.1. Let $S(t)$ be a graph, then $S(t) = (V(t), E(t))$, where $V(t)$ is a set of vertices (i.e., agents) and $E(t)$ is a set of edges (i.e., links between agents). $S(t)$ is connected and for any edge $\{u, v\} \in E(t)$, we assume $u \neq v$.

We represent by a^i an agent in the system and by $N^i(t)$ the set of its neighbours at time t. Also, $n(t)$ is the number of agents in the system at time t (i.e., $n(t) = |V(t)|$). Note that $V(t)$ and $E(t)$ are time-dependent, and so, the set of agents as well as the topology of the system are allowed to change over time. More details as to how that is supported will be given in Section 4.1.3. Finally, we assume each agent runs asynchronously, with its own time line. As we present in Section 4, however, we have put all the formalization in terms of a system-wide time line, just to simplify the presentation of our approach.

3.2. Self-Optimization Model

To perform the decentralized and optimal control of a shared resource pool, it is essential to provide a way of categorizing all possible allocations at a point in time, in terms of how good their outcome will be for the system. For that reason, we use the utility function framework in *Darma*. Utility functions have been pointed as the main tool for enabling self-optimizing behaviour [Kephart and Das 2007], which, in turn, will allow *Darma* to obtain optimal resource usage, at all times.

For that, we assume each agent a^i has a utility function $u^i(x)$, specifying how useful a particular share x is. Then, the following collective utility function $U(X)$, proposed in [Tesauro and Kephart 2004], is used:

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

where $X = \{X_1, \ldots, X_{n(t)}\}$ is an allocation vector over all agents in the system and X_i is the share of agent a^i. Such a utility associates a real-scalar value with every possible allocation at a particular time, allowing them to be compared..

3.3. Problem Formalization

Because we are interested in obtaining optimal resource usage at any point in time, the collective utility function $U(X)$ will be used as part of an optimization problem. For our purposes, the optimization model proposed in [Bennani and Menascé 2005; Tesauro et al. 2005] has been employed. The resource management problem is then stated as follows:

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

where $R(t)$ is the amount of resources available in the pool at time t and the constraint limits the sum of the allocated resources to $R(t)$. In a practical setting, the value of $R(t)$ could be set by system administrators from a management station [Johansson et al. 2006], then being propagated throughout the system. That would be necessary whenever the total resource capacity of the pool changes.

There are two points to be highlighted regarding such a formulation. First, it has been chosen because it models very well what we are looking for (optimal resource usage). Even though consumer requirements will be incorporated into the utility functions, sacrifices might be made sometimes to ensure that property, in terms of satisfying these requirements. As a consequence of that, no consumer has explicit priority over any other. Second, notice that $R(t)$ represents a single *type* of resource, like bandwidth or disk storage, and consequently we are assuming that one resource management process deals with one type of resource. Different resources could surely be packed into a *server*, in which case servers would be the resource to be allocated.

4. DARMA

In this section we present *Darma*. Firstly, we present the *Mathematical Model* behind it, which defines how a set of agents solve the optimization problem in Equation 2 in a decentralized way. Secondly, we present a *Multiagent Model*, which provides more concrete realization of the Mathematical Model, defining possible algorithms for the agents.

4.1. Mathematical Model

The Mathematical Model is composed by four sub-models. The *Optimization Model* defines how to break down the original optimization problem into smaller models which are assigned to each agent. Then, because of the way the optimization problem is broken down, an *Epidemic Model* defines how agents communicate and exchange information. Due to this information exchange process, a *Termination Model* is defined, to allow agents to determine when such a process is finished. Finally, a *Demand Model* is defined, serving as an extension point of *Darma*.

4.1.1. Optimization Model. To decouple the optimization problem in Equation 2, we first define the utility function of the agents. For that, let $u^i(x)$ be the utility function of agent a^i, then:

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

where x is the amount of resource 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 demand for resources. The motivation for using such a utility function is threefold. First, it represents the satisfaction of an agent very well, in the sense that, eventually, giving it more resources will bring little improvement to its utility. Initially, however, such an improvement is exponential, indicating the importance of having the agent to receive some resources. From an utility's perspective, this would be the expected behaviour, and the chosen function models that correctly. Second, such an utility function grows to a limit, 1 in this case, allowing us to determine when an agent is satisfied enough; for the purposes of defining its resource demand, as illustrated in the evaluation. Finally, by using this function, it will be possible to break down the optimization problem into models that each agent can use to find its own share.

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. Whereas we understand that $\alpha^i(t)$ will change over time, we assume that it will comply with the following:

ASSUMPTION 1. *Let $t^{'}$ and $t^{''}$ be the start and end time of a resource management process, then, for all $a^i \in V(t)$ and t such that $t^{'} \leq t \leq t^{''}$, $\alpha^i(t) = \alpha^i(t^{'})$.*

which guarantees that $\alpha^i(t)$ will remain constant during the resource management process itself. Nothing prevents, for example, a scenario where an agent detects a considerable increase/decrease in its resource demand, during a resource management process, and given that, signal other agents to engage in a *new* process. In this case, from the perspective of our work, one resource management process is terminated, before its end, and another one started, when all changes in resource demands will be taken into account. Clearly, this triggering of when a resource management starts can be done in different ways; in the evaluation, for example, we have used timed events. That is, however, outside the scope of our work, and *Darma* focuses instead of the actual process of adapting the shares.

Having defined $u^i(x)$, the first step is to transform the constrained optimization problem in Equation 2 into an unconstrained one, which can be done with the method

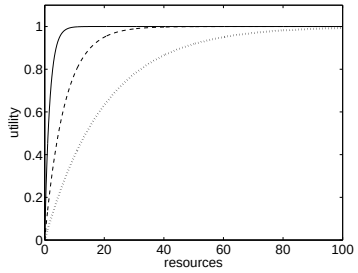


Fig. 2: Sample plots of the utility of the agents

of the Lagrange multipliers. In this case, the new problem is:

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

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

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

The next step is to decompose the new problem into the models that will calculate each agent's share. For that, we can use

$$\nabla L(X, \lambda) = 0, \tag{5}$$

which gives us the set of equations below.

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

However, for X to be an optimal solution of the optimization problem, it must satisfy the Karush-Kuhn-Tucker (KKT) Conditions. Also, 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 are straightforward, and due to space constraints, we have decided to leave them out of the paper.

We now must isolate X_i in Equations 6, more precisely in $\frac{\partial L}{\partial X_i} = 0$. Developing it, gives us:

$$\alpha^i(t) e^{\left(-\alpha^i(t) X_i\right)} - \lambda = 0, \tag{7}$$

and, by solving the above for X_i, we end up with:

$$X_i = \frac{\ln \alpha^i(t) - \ln \lambda}{\alpha^i(t)}. \tag{8}$$

That was the decomposition we were looking for, enabling each agent to find its own share such that $U(X)$ in Equation 2 is maximized. Notice that, in order to do so, agents need not only their own $\alpha^i(t)$ but also the value of $\ln \lambda$. This is the global information that binds the models of all agents together, and so they need to calculate it before

finding their share. Given the characteristics of *Darma*, this has to be done in a decentralized way. For that, we start with $\frac{\partial L}{\partial \lambda} = 0$, from Equation 6, which results in:

$$\left(\sum_{i=1}^{|X|} -X_i \right) + R(t) = 0. \tag{9}$$

Substituting Equation 8 in Equation 9, we have that:

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

By isolating $\ln \lambda$, we end up with:

$$\ln \lambda = \frac{\left(\sum_{i=1}^{|X|} \frac{\ln \alpha^i(t)}{\alpha^i(t)} \right) - R(t)}{\sum_{i=1}^{|X|} \frac{1}{\alpha^i(t)}}. \tag{10}$$

With that, agents are now able to find the value of $\ln \lambda$, and from that, calculate their own share through Equation 8. Because $\ln \lambda$ depends on the α of all agents, each $\alpha^i(t)$ will be disseminated through the system, eventually reaching every other agent. We will call this process *dissemination*. This approach, similarly used in [Nedic and Ozdaglar 2009], can be achieved with epidemic protocols [Babaoglu and Jelasity 2008; Kermarrec and Van Steen 2007]. In *Darma*, the epidemic protocol is defined in the Epidemic Model, which we present next.

4.1.2. Epidemic Model. This model formalizes the dissemination of the α values, so as to calculate $\ln \lambda$. In principle, it could be calculated with solutions for computing aggregates [Kempe et al. 2003; Jelasity et al. 2005]. Some of the solutions in this context, however, are limited to general aggregates - e.g., SUM, AVERAGE, and MAX - which do not fully satisfy our needs. Other solutions do not focus on providing precise aggregates, which would cause sub-optimal solutions in our case, or assume some kind of hierarchy on the network, which is not the case with *Darma*. Due to the specificity of $\ln \lambda$ and requirements of *Darma* in terms of decentralization, scalability, precision, and fault-tolerance, a more tailored approach would suit it better.

In *Darma*, the dissemination of the α values is performed using an *Anti-entropy* epidemic approach [Demers et al. 1987]. 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. With this approach, an agent would be able to synchronize its replica with another chosen agent, and by doing so over time, all replicas can be synchronized. To apply such an approach in *Darma*, we first need to define what would be the format of the messages being sent by the agents, which is: $< \rho^i, \alpha^i(t) >$. In this case, ρ^i is the identifier of agent a^i, for $\rho^i \in \mathbb{N}^*$, and $\alpha^i(t)$ its resource demand, as discussed before.

We assume that agents keep the ρ^i and $\alpha^i(t)$ received in separate places. More precisely, let $O^i(t)$ be the set containing the identifiers of the agents from which a^i has received an α value, plus its own identifier ρ^i. Also, let $M^i(t)$ be a multiset that holds all α values an agent a^i received until time t, plus its own $\alpha^i(t)$. We then define that $O^i(t)$ is the replica that each agent will try to synchronize with the others. As a consequence of that, $M^i(t)$ will be synchronized too.

Before going any further with the synchronization of $O^i(t)$, we make a few points in terms of the performance of the anti-entropy algorithm. In our case, eventually the replicas will be too large, depending on the system size. Given that, if agents compare the replicas element by element, they might potentially do so only to find out that the

replicas are the same, thus wasting time and communication. Instead, as proposed in [Demers et al. 1987], a checksum can be used. In that case, agents would first analyze the checksum, only comparing the actual replicas if their checksum happen to be different. We define the checksum of the replica of agent a^i, represented by $k^i(t)$, using its identifier as well as the ones received, as follows:

$$k^i(t) = \sum_{\rho \in O^i(t)} 2^\rho, \tag{11}$$

That guarantees a unique signature in terms of the α values that agent a^i holds. By checking whether two checksum functions are the same or not, an agent would be able to find out if the corresponding replicas are synchronized. Knowing that two replicas are not synchronized is only part of the process though. Given two replicas A and B, an agent still does not not know whether: 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. The interesting aspect of the $k^i(t)$ function is that it actually allows agents to check if a replica A is a subset of a replica B, with the following:

$$\sum_{a \in A} 2^a \ \& \ \sum_{b \in B} 2^b, \tag{12}$$

where $\&$ is a bitwise AND operation. If the result of this operation is $\sum_{a \in A} 2^a$, then A is a subset of B. Otherwise, there is at least one a in A that is not in B. In more general terms, actually, Equation 12 returns the *intersection* between sets A and B. Similarly, we can combine the "+", "-", and "&" operators to obtain numbers that represents the union and the complement between replicas A and B.

The advantage of having these characteristics on the $k^i(t)$ function is that the dissemination process can be more directed. More precisely, when disseminating α values, we can set agents to only do so to neighbours who have not received at least one of the α it has, if any. For that, they would choose a random neighbour among those satisfying this condition, using the $g^{ij}(t)$ function below, for a neighbour a^j:

$$g^{ij}(t) = k^i(t) - \left(k^j(t) \ \& \ k^i(t)\right), \tag{13}$$

which translates into a number representing the ρ and α values that a^i can disseminate to a^j. If no such a pair exists, then $g^{ij}(t) = 0$. This approach provides more efficiency to the dissemination process, over a purely random one for choosing the neighbours, as we demonstrate in the evaluation.

Once an agent picks a neighbour, a push and a pull is required to synchronize their replicas. When pushing, all $\rho \in O^i(t)$ and $\alpha \in M^i(t)$ such that the neighbour does not have, are pushed to it. When pulling, a^i pulls from the neighbour a^j all $\rho \in O^j(t)$ and $\alpha \in M^j(t)$ such that a^i does not have. That is done by checking if

$$2^\rho \ \& \ g^{ij}(t) = 2^\rho \tag{14}$$

for the push, for all $\rho \in O^i(t)$, and

$$2^\rho \ \& \ g^{ji}(t) = 2^\rho \tag{15}$$

for the pull, for all $\rho \in O^j(t)$. Every pair $< \rho, \alpha >$ such that ρ satisfies one of the above conditions is pushed or pulled, as appropriate. An important consequence of employing this method is that we have a guarantee that all α values will be delivered to all agents. Results showing how quick the α are disseminated, and that all of them are indeed disseminated, are provided in the evaluation.

4.1.3. Termination Model. This model defines how agents interact locally in order to find out when the dissemination is finished. This is an essential step in the resource management process of *Darma*, as it will tell agents when they are ready to use the α they hold to calculate $\ln \lambda$ and then their own share. Again, this has to be done in a decentralized and fault-tolerant way. To do so, agents will exchange signaling information, relying solely on their neighbourhood. The basis of this exchange process relies on a time-varying function $x^i(t)$, defined as:

$$x^i(t) = \begin{cases} \frac{k^i(t)+\sum_{a^j \in N^i(t)} x^j(t)}{|N^i(t)|+1} & \text{if } t > 0 \\ 0 & \text{if } t = 0 \end{cases} \tag{16}$$

The value of such a function 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. What we want are properties stating the convergence of $x^i(t)$ as the dissemination goes on, since they are the basis for the agents to determine its end. For that reason, we define *equilibrium* as a state of the system on which the $x^i(t)$ of each agent converges, formalized by the following:

Definition 4.1. Let t be a point in time. It is said that a system $S(t)$ is in a state of equilibrium when, for all $a^i \in V(t)$ and $t^{'} \geq t$, $x^i(t^{'}) = c^i$.

The rationale behind $x^i(t)$ and the idea of equilibrium is that we expect the system to reach such a state once all α are disseminated. More precisely, it can be shown that, at that point, all $x^i(t)$ will converge to a specific value, as stated by the following lemmas:

LEMMA 4.2. *Let t be a point in time, such that, for all $a^i \in V(t)$ and $t^{'} \geq t$, $k^i(t^{'}) = l^i$. Then, $x^i(t^{'}) \to c^i$ when $t^{'} \to \infty$.*

LEMMA 4.3. *Let t be a point in time, then, for all $a^i \in V(t)$, $x^i(t) \to l$ when $t \to \infty$ if and only if $k^i(t) = l$.*

And as a consequence of the above:

COROLLARY 4.4. *Let t^* be the time when the dissemination process is finished. Then, for $t^{'} \geq t^*$, $x^i(t^{'}) \to \sum_{a^i \in V(t^*)} 2^{\rho^i}$ when $t^{'} \to \infty$, for all $a^i \in V(t^*)$.*

The properties above give us two things. First, the guarantee that the system will reach equilibrium, once agents stop receiving α values (Lemma 4.2). Second, they ensure that, *only* when the dissemination process is finished, all $x^i(t)$ will converge to the same, specific, value (Lemma 4.3 and Corollary 4.4). These properties are important because they provide a starting point for defining when the dissemination process is finished. Firstly, however, we are interested in properties guaranteeing that the end of the dissemination process is a necessary condition for the common convergence of $x^i(t)$ (i.e., Lemma 4.3). That is guaranteed by the following:

LEMMA 4.5. *Let t be a point in time, then, $k^i(t) = l$, $\forall a^i \in V(t)$, if and only if $O^i(t) = O^k(t)$, for any $a^i, a^k \in V(t)$ such that $i \neq k$.*

LEMMA 4.6. *Let t^* be the time when the dissemination process is finished. Then, $O^i(t^{'}) = O^k(t^{'})$, for any $a^i, a^k \in V(t^{'})$ such that $i \neq k$, if and only if $t^{'} \geq t^*$.*

Whereas Lemma 4.5 ensures that $k^i(t)$ in $x^i(t)$ will only be the same, for any two agents, when their $O^i(t)$ is the same, Lemma 4.6 guarantees that the latter only holds when every agent receives the α of all the others. Consequently, the system will only

reach equilibrium once the dissemination process is finished, which is the guarantee we wanted. We provide in Section 5 a theoretical analysis on the convergence of *Darma*, which proves Lemmas 4.2 and 4.3. Due to space constraints, however, we have left out of the paper the proofs for Lemmas 4.5 and 4.6. Whereas they reinforce the convergence properties we stated, we decided to concentrate mostly on analyzing the convergence aspect itself, but still providing practical evidence on the correctness of these lemmas in the evaluation.

From the previous properties, we define the final element of the Termination Model, the $d^i(t)$ function, which will provide agents with the specific value they have to look after to determine the end of the dissemination process. The convergence of $x^i(t)$ cannot be used for that purpose, since network delays could cause it to be stuck at a particular value for some time, potentially leading agents to falsely believe that the dissemination is over. Given that, $d^i(t)$ is defined as:

$$d^i(t) = \sum_{a^j \in N^i(t)} |x^i(t) - x^j(t)| \tag{17}$$

The importance of the difference function then comes from the following Corollary, a consequence of Lemma 4.3:

COROLLARY 4.7. *Let t be a point in time. If $k^i(t^{'}) = l$, for all $a^i \in V(t)$ and $t^{'} \geq t$, then, $d^i(t^{'}) \to 0$ when $t^{'} \to \infty$.*

Agents now know that, once their $d^i(t)$ reaches zero, the dissemination process is over. Even if $d^i(t)$ gets stuck at any other value, due to delays, agents will still know that the dissemination is not finished yet. Due to Lemmas 4.5 and 4.6, that is guaranteed to happen only from that moment on.

Apart from the decentralization characteristics we have discussed, $x^i(t)$ and $d^i(t)$ also provide *Darma* with support for system changes. The distributed computing systems we are dealing with will most likely change over time, in terms of agents joining and/or leaving, so situations like this must clearly be handled. We analyze these changes in the system from two perspectives: 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 would not be affected, since our Termination Model does not rely on any global information related to the number of agents. Therefore, any changes to the set of agents in the system are automatically picked up by the Termination Model once the resource management process starts, with no further actions or updates being required. 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, the convergence of the system cannot be guaranteed. More precisely, if an agent joins after at least one $d^i(t)$ converges to 0, the system will not reach equilibrium. In terms of agents leaving the system, convergence is still guaranteed, as long as the remaining topology of the system matches Definition 3.1. In this case, because the agent's $\alpha^i(t)$ is already being disseminated, it will still be considered part of the resource management process. This scenario of agents joining and leaving during the resource management process are characteristic of *unplanned* changes, like the crash of an agent. We believe that unplanned additions of resource consumers (e.g., client applications), and consequently agents, can be controlled in practice. Therefore, without any loss of flexibility, we restrict the addition of agents to happen only in between two resource management processes.

Finally, we should also expect topology changes at any moment of the system's execution. Due to the fact that our Termination Model relies only on the neighbourhood of the agents, it is able to guarantee that the system will reach equilibrium in face of changing topologies, even if it happens during the resource management process. Again, any changes in the topology should produce a system matching Definition 3.1. We demonstrate in Section 6 that all the properties of the Termination Model hold in practice, even in the face of the system changes mentioned.

4.1.4. Demand Model. As defined in the Optimization Model, $\alpha^i(t)$ indicates an agent's resource demand at time t. Such a demand is determined by both workload and consumer requirements. We then need a way of mapping them to $\alpha^i(t)$, and for that reason, the Demand Model has been defined. The problem is that such a mapping is application-specific, and the Demand Model could not employ a general model of calculating $\alpha^i(t)$. Therefore, this model is actually a hot spot of *Darma*, which should be extended for each application scenario. For that, first, a function $w^i(t)$ is defined, representing the expected workload at time t of the consumer associated with the agent a^i. In this case, clearly $w^i(t) \in \mathbb{R}$ and $w^i(t) \geq 0$. Representing consumer requirements, however, is troublesome, since that can be done in different ways. Given that, we will assume that this is an inherent knowledge available to the agent at the Demand Model, but demonstrate in the evaluation how they can be represented and put together with the workload in order to calculate $\alpha^i(t)$.

4.2. Multiagent Model

From the Mathematical Model, a Multiagent Model has been defined, comprising the architecture of the agents as well as a set of algorithms. For doing so, multiagent design methodologies, like MASINA [Aguilar et al. 2007], Prometheus [Padgham and Winikoff 2002], and Gaia [Wooldridge et al. 2000] could have been employed. However, the focus here was mostly on providing more concrete realizations to the definitions in the Mathematical Model, and also demonstrate how such definitions can be put together. Also, from a multiagent modeling perspective, *Darma* is actually simple. Notice, for example, that only one *type* of agent exists in the system, the *resource consumer*. For those reasons, we have opted for a simpler approach for modeling the agents.

Firstly, the architecture of the agents is defined, presented in Figure 3. In the architecture, each module corresponds to one of the sub-models of the Mathematical Model of *Darma*, whereas the arrows indicate the workflow within the agents once a resource management process starts, along with the information that is passed on to each module. The workflow of the agents is implemented by Algorithm 1. More precisely, when a resource management process starts, the current workload, $w^i(t)$, is passed on to the agent. Then, through the Demand Module, the agent uses $w^i(t)$ to calculate its $\alpha^i(t)$ (line 1). From that, the $\alpha^i(t)$ is passed on to the Termination module of the agent, through the `termination(`$\alpha^i(t)$`)` function (line 2).

When the Termination module is called, the code in Algorithm 2, which implements the Termination Model, is executed. It starts by initializing $x^i(t)$, $O^i(t)$, and $M^i(t)$ (lines 1, 2, and 3). After that, it performs a loop where the dissemination algorithm is invoked, through the `dissemination(`$O^i(t)$`,` $M^i(t)$`)` function (line 5), the value of $x^i(t)$ is updated (line 6), then checking for convergence (line 7). Once that happens, the multiset $M^i(t)$ is returned (line 8). We are assuming that, in the meantime, ρs and α values are being received and kept in their respective places, but we omitted the pseudocode for that. Back to Algorithm 1, $M^i(t)$ is used to calculate the agent's resource share, through the `optimization(`$M^i(t)$`)` function (line 3). Such a function uses this multiset to find $\ln \lambda$, through Equation 10, and from that, the agent's share, using Equation 8, which is finally returned (line 4). From that point, the agent itself or

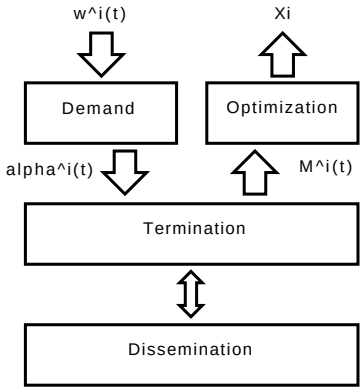


Fig. 3: Architecture of the agents in *Darma*

the consumer it represents would issue a request to the resource pool for the amount of resources X_i.

ALGORITHM 1: Core Agent's Algorithm

Require: The workload $w^i(t)$ for the current time stamp t.
1: $\alpha^i(t) \leftarrow$ demand($w^i(t)$)
2: $M^i(t) \leftarrow$ termination($\alpha^i(t)$)
3: $X_i \leftarrow$ optimization($M^i(t)$)
4: **return** X_i

ALGORITHM 2: Termination Algorithm

Require: The current $\alpha^i(t)$ of the agent.
1: $x^i(t) \leftarrow 0$
2: $O^i(t) \leftarrow \{\rho^i\}$
3: $M^i(t) \leftarrow \{\alpha^i(t)\}$
4: **repeat**
5: dissemination($O^i(t)$, $M^i(t)$)
6: $x^i(t+1) \leftarrow \frac{k^i(t)+\sum_{\alpha^j \in N^i(t)} x^j(t)}{|N^i(t)|+1}$
7: **until** $d^i(t) \approx 0$
8: **return** $M^i(t)$

In terms of dissemination, once the `dissemination(`$O^i(t)$, $M^i(t)$`)` function is invoked in Algorithm 2, the steps in Algorithm 3 are executed. First, it selects a neighbour to disseminate to, through the `pick(`$k^i(t)$`)` function (line 1). If there is such a neighbour (line 2), the appropriate agent identifiers and corresponding α values, as defined in Equations 14 and 15 of the Epidemic Model, are pushed to and pulled from it (lines 3 and 4), so as to synchronize the replicas $O^i(t)$ and $M^i(t)$. The `pick(`$k^i(t)$`)` function, whose code is presented in Algorithm 4, simply returns a random neighbour among those who have not received some of the α values that the agent has. For that, the algorithm starts by building a set with the agent's neighbours (line 1). From that,

it selects a random neighbour from this set (line 3), using $g^{ij}(t)$ to find out if it is eligible to be picked for the dissemination (line 4). If so, the neighbour is returned (line 5), otherwise, it is removed from the set (line 7), and the process is repeated until either a neighbour is picked or the set is empty. As a consequence, it is guaranteed that, if there are neighbours for which the agent is able to disseminate $\alpha^i(t)$ values, one of them will be picked.

ALGORITHM 3: Dissemination Algorithm

Require: The current $O^i(t)$ and $M^i(t)$ of the agent.
1: $a^j \leftarrow$ pick($k^i(t)$)
2: **if** a^j **then**
3: push(a^j, $\{< \rho^k, \alpha^k(t) >\}^* | \rho^k \in O^i(t) - O^j(t)$)
4: pull(a^j, $\{< \rho^k, \alpha^k(t) >\}^* | \rho^k \in O^j(t) - O^i(t)$)
5: **end if**

ALGORITHM 4: Algorithm of the pick($k^i(t)$) function

Require: The current $k^i(t)$ of the agent
1: $L = \{a^j\}^*, \forall a^j \in N^i(t)$
2: **while** $|L| > 0$ **do**
3: $a^k \leftarrow$ random(L)
4: **if** $g^{ik}(t) \neq 0$ **then**
5: **return** a^k
6: **else**
7: $L = L - \{a^k\}$
8: **end if**
9: **end while**

5. CONVERGENCE ANALYSIS

In this section we analyze the convergence of $x^i(t)$, and consequently $d^i(t)$, as stated in the Termination Model. First, we start looking into Lemma 4.2. In this analysis, we are assuming that no agent leaves the system and also that topology does not change. Later we provide directions as to how the analysis would go if incorporating these changes. We start by defining a perturbation, as follows:

Definition 5.1. A perturbation $p^i(c)$ is any change in the $x^i(t)$ value of an agent, such that:

$$p^i(c) = \frac{c}{|N^i(t)| + 1}, \tag{18}$$

where c is what has originated the perturbation.

In the above, c can either be another perturbation, propagated from one of the agent's neighbours, or a change in the α received by it (i.e., $O^i(t)$). Clearly, it all starts with a change in the $k^i(t)$ function of an agent, triggering a "chain reaction", where perturbations would start propagating from the agent itself to the others, and so on. Lets call $c^i(t)$ such an initial change in $k^i(t)$. Let $t^{'}$ be the time on which some α has been received by an agent and t such that $t^{'} \geq t$, then, it is easy to see that $c^i(t^{'})$ would be:

$$c^i(t^{'}) = k^i(t^{'}) - k^i(t) \tag{19}$$

The initial perturbation would be on a^i, and such that:

$$p^i(c^i(t^{'})) = \frac{c^i(t^{'})}{|N^i(t)|+1}. \tag{20}$$

Agent a^i would then generate a set of $|N^i(t)|$ perturbations $p^i(c)$ to each neighbour a^j. In this case, $c = p^i(c^i(t^{'}))$, and from Equation 18, we have:

$$p^j(p^i(c^i(t^{'}))) = \frac{c^i(t^{'})}{(|N^j(t)|+1)\,(|N^i(t)|+1)}. \tag{21}$$

Consequently, $\sum_{a^j \in N^i(t)} |N^j(t)|$ new perturbations would be propagated. Clearly, that process would continue for every neighbour of each a^j, and so on, thus generating a perturbation tree. Given that, we define:

Definition 5.2. Let $c^i(t)$ be an initial change as in Equation 19. We define the perturbation tree of $c^i(t)$ to be the one where $p^i(c^i(t))$ is the root and the remaining nodes are the perturbations propagated from $p^i(c^i(t))$ and by the other agents, recursively, over time.

Any node of a perturbation tree can be traced back to the initial change $c^i(t)$, thus forming a *perturbation path*. Furthermore, each node of the tree has an agent associated, from which the perturbation is being propagated. Let e be a node in the tree, $H(e)$ be a perturbation path tracing e back to the root, and $O(H(e))$ the set of the agents associated with each of the nodes in the perturbation path $H(e)$. From that, we can find the perturbation being propagated by the node with:

$$p^e(c^i(t)) = \frac{c^i(t)}{\prod_{a \in O(H(e))} |N^a(t)|+1}. \tag{22}$$

As a consequence of the above, for two nodes $e^{'}$ and $e^{''}$ in a perturbation tree, such that $e^{'}$ is a parent of $e^{''}$, we have that

$$|H(e^{''})| > |H(e^{'})|. \tag{23}$$

Because perturbations will be propagated infinitely, then

$$|H(e)| \to \infty. \tag{24}$$

Consequently, in Equation 22

$$\prod_{a \in O(H(e))} |N^a(t)| + 1 \to \infty, \tag{25}$$

and in turn

$$\lim_{|H(e)| \to \infty} p^e(c^i(t)) \to 0. \tag{26}$$

We can then consider that, at some point, agents will not propagate any perturbation anymore, following the initial change $c^i(t)$. Let T be a perturbation tree of an initial change $c^k(t)$ in an agent a^k and $A^i(T)$ be the set of all nodes in T such that the agent associated with such a node is a^i. Then, from 26, we know that:

$$\sum_{a \in A^i(T)} p^a(c^k(t)) = c^i \tag{27}$$

The total of value of $x^i(t)$, for any agent in the system, following from the initial change $c^k(t)$, is

$$x^i(t+1) = x^i(t) + \sum_{a \in A^i(T)} p^a(c^k(t)). \tag{28}$$

Consequently, an initial change will generate perturbations on each $x^i(t)$ until a certain point, where it converges. If we view a dissemination process as a finite sequence of initial changes on different agents, it is straightforward to conclude that the final value of $x^i(t)$ will be a set of sums as in Equation 28, one for each initial change. This in turn shows that, once agents stop receiving identifiers and α values, all $x^i(t)$ converge, guaranteeing equilibrium under these conditions.

When analyzing Lemma 4.2 considering topology changes, the key aspect is to realize that Equation 26 remains unaffected. In this case, only the perturbation paths will change, not the properties of the perturbation itself. As for agents leaving the system, it can be shown that nothing is lost when that happens. In other words, perturbations that would be propagated to agents that left the system will simply be redistributed and passed on to other agents, and so Equation 26 also holds in this scenario. The fact that Equation 26 holds in both cases is the important aspect here, as it guarantees that Equation 28 holds, thus ensuring correct convergence.

Now, we analyze in details the convergence stated by Lemma 4.3. Like in the previous analysis, we are assuming, for now, no agents leaving the system or topology changes. We know from Equation 28 that, following an initial change in an agent's $k^i(t)$ value, the total value that will be propagated to another agent a^k is

$$\sum_{a \in A^k(T)} p^a(c^i(t)). \tag{29}$$

For each node a in $A^k(t)$, given $H(a)$ and $O(H(a))$, as defined before, we can rewrite the equation above as

$$c^i(t) \left(\sum_{a \in A^k(T)} \frac{1}{\prod_{o \in O(H(a))} (|N^o(t)| + 1)} \right). \tag{30}$$

We represent the inner part of this equation as

$$z^{ki} = \sum_{a \in A^k(T)} \frac{1}{\prod_{o \in O(H(a))} (|N^o(t)| + 1)}, \tag{31}$$

which gives us the percentage of an initial change $c^i(t)$ that is propagated to an agent a^k. For any agent a^i, we have

$$\sum_{a^k \in V(t)} z^{ki} = 1. \tag{32}$$

In other words, the percentage that an initial change in a^i will propagate to the agents, including itself, adds up to 1. Given a set of initial changes in each agent a^i, the total value that will be received by another agent a^k is

$$\sum_{a^i \in V(t)} c^i(t) z^{ki}, \tag{33}$$

and the value of $x^k(t)$ after that will be

$$x^k(t+1) = x^k(t) + \sum_{a^i \in V(t)} c^i(t) z^{ki}. \tag{34}$$

To show that all $x^i(t)$ will converge to the same value when their $k^i(t)$ is the same, we assume that the initial change $c^i(t)$ on each agent will be such that $c^i(t) = k^i(t) = l$. Assume all $c^i(t)$ happen at time t, for $t > 0$. Since $x^i(0) = 0$, from Equation 34, we have to show that

$$\sum_{a^i \in V(t)} l z^{ki} = \sum_{a^i \in V(t)} l z^{wi} \tag{35}$$

for any $a^k, a^w \in V(t)$ such that $k \neq w$, which can be rewritten as

$$\sum_{a^i \in V(t)} z^{ki} = \sum_{a^i \in V(t)} z^{wi}. \tag{36}$$

The above can be proved if we show that the sum of all the percentage that is propagated to any agent adds up to 1. In other words

$$\sum_{a^i \in V(t)} z^{ki} = 1. \tag{37}$$

Whereas it is easy to see that Equation 32 holds, this one requires some more elicitation. For that, we can show that

$$\sum_{a^i \in V(t)} z^{ki} = \sum_{a^i \in V(t)} z^{ik}, \tag{38}$$

since we know the right-hand side of the equality is 1. Remember that the above is

$$z^{k1} + \ldots + z^{kn(t)} = z^{1k} + \ldots + z^{n(t)k}. \tag{39}$$

We will then show that, for any $a^i, a^j \in V(t)$

$$z^{ij} = z^{ji}, \tag{40}$$

which would in turn make Equation 37 to hold. Note that, for every path ending in a^j in the perturbation tree generated by a^i, the same path exists in the perturbation tree of a^j, but reversed. From that, let T^i be a perturbation tree of a^i. For each node a in $A^j(T^i)$, let $R(H(a))$ be the inverse of $H(a)$. If $O(R(H(a)))$ is the set of the agents associated with each node in $R(H(a))$, then

$$\sum_{a \in A^j(T^i)} \frac{1}{\prod_{o \in O(R(H(a)))} (|N^o(t)| + 1)} = \sum_{a \in A^j(T^i)} \frac{1}{\prod_{o \in O(H(a))} (|N^o(t)| + 1)}. \tag{41}$$

Note, however, that for every node a in $A^j(T^i)$, the path $R(H(a))$ can be rewritten simply as $H(a)$, for every node a in $A^i(T^j)$. The equation above becomes

$$\sum_{a \in A^i(T^j)} \frac{1}{\prod_{o \in O(H(a))} (|N^o(t)| + 1)} = \sum_{a \in A^j(T^i)} \frac{1}{\prod_{o \in O(H(a))} (|N^o(t)| + 1)}, \tag{42}$$

which clearly brings us to

$$z^{ij} = z^{ji}. \tag{43}$$

That in turn, shows that Equation 37 holds, and in turn Equation 36 holds too. The latter was what we wanted to show to prove that all $x^i(t)$ converge to the same value when $k^i(t) = l$ for all $a^i \in V(t)$.

To prove that this happens *only* when $k^i(t) = l$, we consider that this only holds for some of the agents in the system. For any other agent a^k such that this condition does not hold, we will have $c^k(t) = k^k(t) = l^k$. We have to show that

$$\sum_{a^k \in K} l^k z^{jk} + \sum_{a^i \in I} l z^{ji} = \sum_{a^k \in K} l^k z^{wk} + \sum_{a^i \in I} l z^{wi} \tag{44}$$

does not hold, for any $a^j, a^w \in V(t)$, where K and I are sets of agents such that, for all $a^k \in K$, $k^k(t) = l^k$ and, for all $a^i \in I$, $k^i(t) = l$ and $l > l^k$. Clearly, $K \cup I = V(t)$ and $K \cap I = \emptyset$. We can rewrite the above as

$$\sum_{a^k \in K} l^k \left(z^{jk} - z^{wk}\right) = l \left(\sum_{a^i \in I} z^{wi} - \sum_{a^i \in I} z^{ji} \right). \tag{45}$$

From Equation 37, we know that

$$\sum_{a^k \in k} z^{jk} + \sum_{a^i \in I} z^{ji} = 1, \tag{46}$$

since $K \cup I = V(t)$. We can in turn rewrite the right-hand side of Equation 45 as

$$\begin{gathered} \sum_{a^k \in K} l^k \left(z^{jk} - z^{wk}\right) = l \left(\left(1 - \sum_{a^k \in K} z^{wk}\right) - \left(1 - \sum_{a^k \in K} z^{jk}\right) \right) \\ \sum_{a^k \in K} l^k \left(z^{jk} - z^{wk}\right) = l \left(\sum_{a^k \in K} z^{jk} - z^{wk} \right). \end{gathered} \tag{47}$$

Expanding the sums, we have

$$\begin{gathered} l^{K_1} \left(z^{jK_1} - z^{wK_1}\right) + \ldots + l^{K_{|K|}} \left(z^{jK_{|K|}} - z^{wK_{|K|}}\right) = \\ l \left(z^{jK_1} - z^{wK_1}\right) + \ldots + l \left(z^{jK_{|K|}} - z^{wK_{|K|}}\right). \end{gathered} \tag{48}$$

However, $l > l^{K_i}$, and consequently the equality above could not hold. Therefore, unless $k^i(t) = l$, for all $a^i \in V(t)$, their a^i will not converge to the same value.

To analyze the convergence stated in Lemma 4.3 for agents leaving the system and topology changes, a key aspect is that, as discussed before, Equation 26 holds under these conditions, and consequently so does Equation 37. With that, the other properties that follow from Equation 37 hold in turn, thus causing $x^i(t)$ to converge as stated by the lemma, as our analysis showed.

6. EVALUATION

In this section we present a set of experiments we performed to validate *Darma*. For that, we modelled a data center scenario, on which a number of servers is available to be allocated to some Application Environments (AEs) [Bennani and Menascé 2005]. Each AE processes one type of transaction and, in this case, is represented by an agent of our approach. In the following sub-sections we present how such a scenario is modelled into *Darma* and also results of experiments performed.

6.1. Modelling

Each AE has a time-varying workload, stated in terms of number of requests per second. That then maps to the $w^i(t)$ function of our Demand Model. For our experiments, the values for the workload have been based on the analytical access data of different web applications.

All AEs have a Target Response Time (TRT) as a requirement for their transactions. From that, we derive the Expected Average Response Time (EART) of an AE. That represents the expected response time of the AE's transaction, given some workload and a number of servers allocated to the AE. Based on [Bennani and Menascé 2005], we denote by $r^i(s, w)$ the EART of AE i, as follows:

$$r^i(s, w) = \frac{w.c^i}{s}. \tag{49}$$

In the above, s is the number of servers allocated to AE i, w is the workload, and c^i is the CPU time of the type of transaction processed by AE i, in seconds. We denote by $q^i(w)$ the amount of servers required by AE i so that its TRT will be met. We define it as follows:

$$q^i(w) = \frac{w.c^i}{T^i}, \tag{50}$$

where w and c^i are as in Equation 49, whereas T^i is the AE's TRT. Now, only the actual method for calculating $\alpha^i(t)$ remains to be defined, as follows:

$$\alpha^i(t) = -\frac{\ln(1-H)}{q^i(w^i(t))}, \tag{51}$$

where H is a constant defining the value of the agents utility when the EART of its AE reaches the TRT, i.e., a value close to 1. This models the fact that agents are "happy" with an amount of servers that causes their TRT to be met, but also becoming "happier" if more resources are given. To obtain Equation 51, we would start with:

$$\begin{aligned} u^i(q^i(w^i(t))) &= H \\ 1 - e^{-\left(\alpha^i(t) q^i(w^i(t))\right)} &= H, \end{aligned} \tag{52}$$

which represents what we have just discussed. One would then develop the equation above, finally isolating $\alpha^i(t)$.

6.2. Experiments

Using the data center model presented, we performed a series of experiments to evaluate *Darma*. We focused on the three main aspects of *Darma*: the allocations found, the dissemination of the α values, and the convergence of the agents' $x^i(t)$ and $d^i(t)$. In these experiments, we assumed that the resource management process is performed in discrete points in time. In a real-world scenario, this could represent different hours of the day, on which a re-allocation of the servers would take place. The specific frequency on which the resource management process would happen is to be decided by system administrators, and could thus vary from system to system. For our experiments, we will simply refer to iterations to the moments on which the resource management process takes place.

The values used for ρ^i were such that $\rho^i \in \{1, \ldots, |V(t)|\}$. Also, the topology of the systems being simulated was generated randomly, differing from one experiment to another. For most of the experiments, the topologies generated had nodes with degrees ranging from 3 to 5. Exceptions to that will be mentioned in due course. The results of these experiments are presented next.

6.2.1. Allocation. To evaluate the allocations, we assumed scenarios where at most six AEs are deployed in the data center. The point here is to show the optimal aspects of the allocations found, and their outcome, and so a small-scale system has been used. Experiments involving larger-scale systems will be presented when discussing the dissemination and convergence aspects. For each of the allocation experiments, one simulation was executed, each simulation being performed for 20 iterations. The values used for the CPU times, i.e., c^i in Equation 50, were based on [Bennani and Menascé 2005] and are presented in Table I.

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

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

The first allocation experiment assumes a static system and also that the number of servers in the pool (145) is always greater than the total demand. Also, the workloads plotted in Figure 4 have been used. Based on these workloads, the $\alpha^i(t)$ of each agent was as in Figure 5. As illustrated, each $\alpha^i(t)$ varies exactly the opposite to the workload, thus matching the properties of the agents' utility function. That caused the shares of each AE to be as in Figure 6, and, in turn, their EARTs to be as in Figure 7. In the figure, the dashed line represents the TRT of an AE. Notice that the EART is always smaller than the corresponding TRT. Given the amount of servers in the pool, this shows that *Darma* behaves correctly in this scenario. In this case, the aggregate utility was always such that $U(X) \approx 6$.

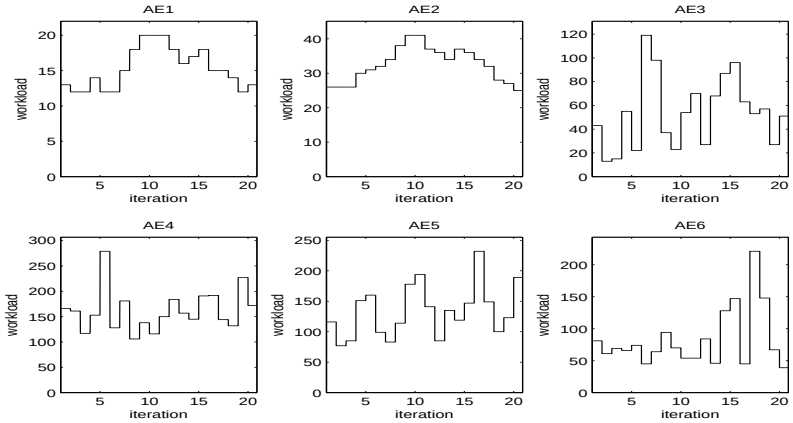


Fig. 4: Workload of each AE in the first allocation experiment

In a second scenario, we have simulated AEs joining and leaving the system, in between two resource management processes. For this scenario, the workload presented in Figure 8 has been used. Notice that, despite having such system changes, *Darma* is able to handle the situation smoothly. That is shown in Figure 9, which illustrates the

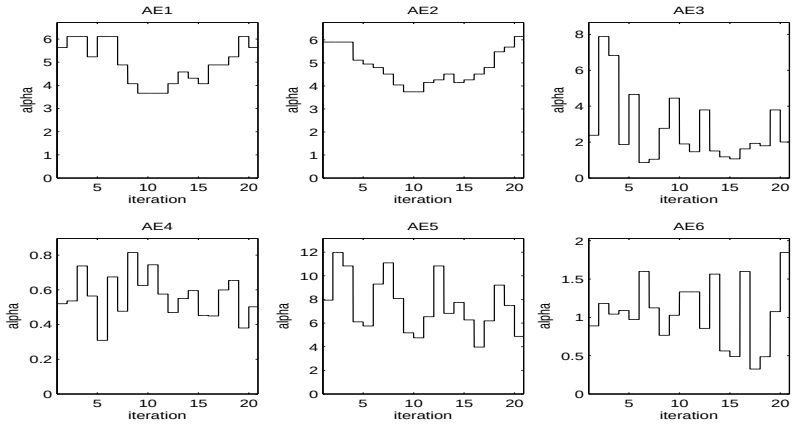


Fig. 5: $\alpha^i(t)$ of each AE in the first allocation scenario

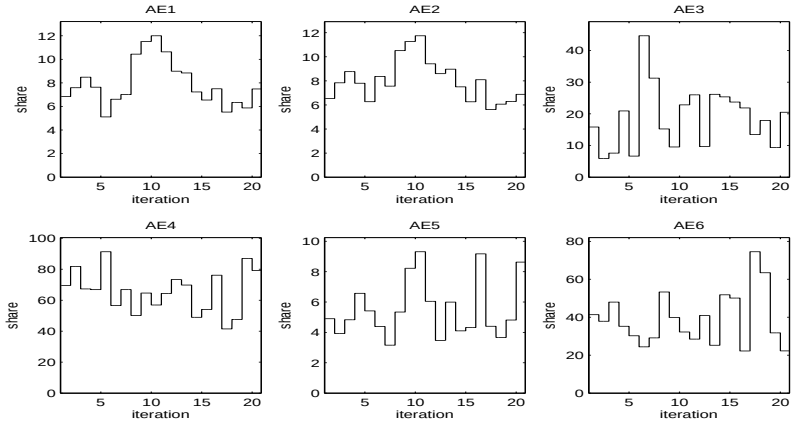


Fig. 6: Shares of each AE in the first allocation scenario

response times of each AE over time. As one can see, all TRTs have been met, regardless of having a varying number of AEs in the system over time. *Darma* is flexible enough to capture such changes, and allocate the resources based on that. The figures for the $\alpha^i(t)$ and shares of each agent have been omitted, as they present results similar to what has been discussed.

Finally, in a third scenario, we have set up a situation of overload. In other words, at some points in time, we allowed the total demand to be greater than the number of servers in the datacenter. For that, the same workloads of the first scenario (Figure 4) have been used, and the number of servers in the datacenter has been set to 100. The overload is represented in Figure 10a, where the total server demand is plotted over time, with the solid line representing the number of servers. During overload, it is clear that not all TRTs can be met, and so, the aggregate utility $U(X)$ will not reach its maximum value, $U(X) \approx 6$ in this case. That is illustrated in Figure 10b, which demonstrates that the value of $U(X)$ drops in moments with, or at the verge of, overload. Still, *Darma* finds a way of allocating the servers in a way that $U(X)$ is maximized, ensuring optimal resource usage.

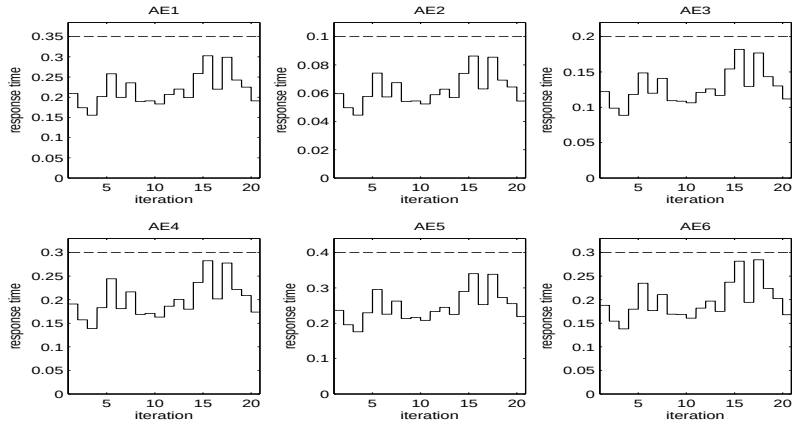


Fig. 7: EARTs of each AE in the first allocation scenario

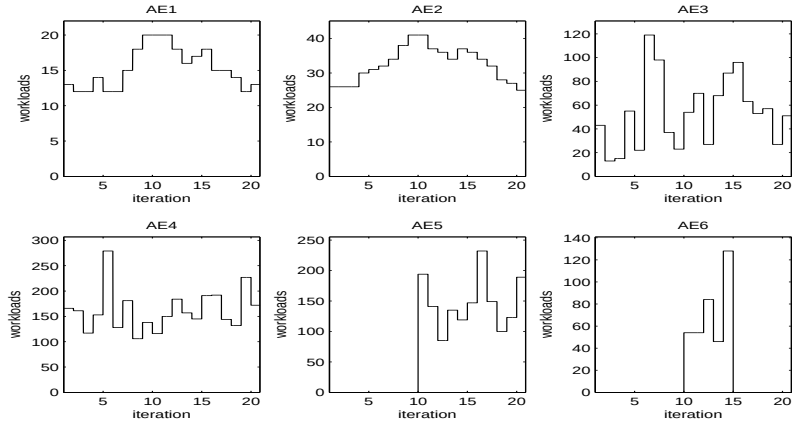


Fig. 8: Workload of each AE in the second allocation scenario

6.2.2. Dissemination. Now we present results of the experiments concerning the dissemination aspect of *Darma*. To perform these experiments, we introduced the idea of a *cycle*. A cycle is one step from each agent, sequentially, where a step means an agent executing lines 5 to 7, in Algorithm 2, once. The order on which the agents are stepped is random and thus differ from one cycle to another. Finally, for each system size being simulated, six runs were performed. From that, the results for all runs were averaged.

Based on that, we observed first how the number of cycles varies as the number of agents in the system increases. The results for such are presented in Figure 11a, demonstrating that the number of cycles grows logarithmically. This behaviour helps providing more scalability to our solution. Another factor that impacts scalability is CPU usage. In our particular case, this boils down to how much CPU time it takes to step one agent, which relies heavily on updating the value of $k^i(t)$. Given the nature of this function, one would expect the CPU time in question to grow exponentially with system size, which is confirmed in Figure 11b. Despite that, notice that not only the CPU times are small, but also, the logarithmic characteristic of the number of cycles actually reduces the steepness of such a curve on a system-wide perspective. With true

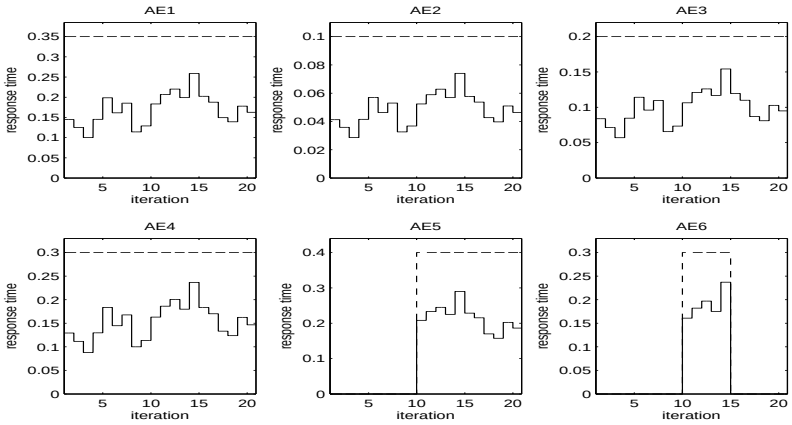
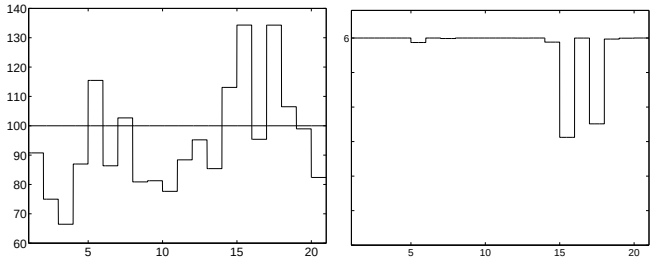


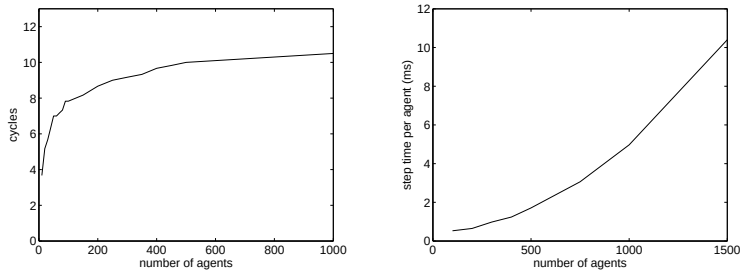
Fig. 9: EARTs of each AE in the second allocation scenario



(a) Total resource demand (b) Aggregate utility $U(X)$

Fig. 10: Simulation results for an overload scenario

parallelism, both factors would enable *Darma* to deliver a quick performance. These CPU values were obtained on a Pentium Dual Core, 2.8 GHz, with 1GB of RAM.



(a) Number of cycles to deliver all α as system size increases (b) CPU usage per agent, per step

Fig. 11: Performance of the dissemination

To provide a better picture of *Darma*'s dissemination algorithm, another experiment on the number of cycles has been performed, now for different ranges of degrees. The

results, for systems with 100, 200, 300, 400, and 500 agents, are presented in Figure 12. As the results show, the number of cycles to deliver all $\alpha^i(t)$ decreases as the system gets more connected; a consequence of having more "ways" through which data can be disseminated. At some point, increasing the range of degrees brings little improvement, potentially reaching a plateau.

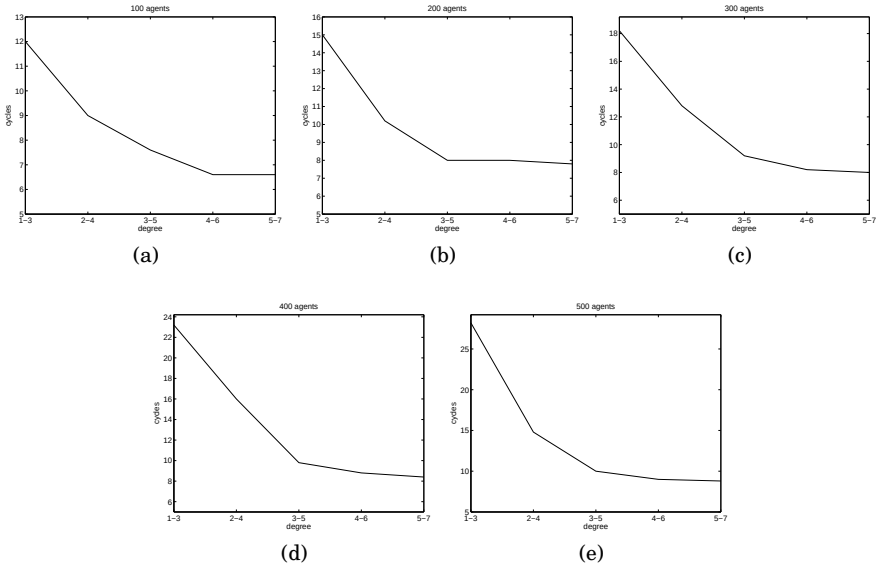


Fig. 12: Number of cycles to deliver all α, for different system sizes, with a varying degree range

As discussed, however, the results on the number of cycles are a consequence of different runs for each system size, and consequently averaged. It would then be meaningful to visualize how the number of cycles actually varies in each of these runs. Given that, in Figure 13, we present the standard deviation on the number of cycles to deliver all $\alpha^i(t)$, for the results illustrated in Figure 11a. As one can see, in all cases, the standard deviation is low, which, in turn, indicates a steady performance of *Darma*. That is confirmed by the results presented in Figure 14, which presents the standard deviation for the results presented in Figure 12.

In another dissemination experiment, we have run simulations using a random approach for picking neighbours, instead of our approach based on $g^{ij}(t)$. We analyzed the results from two different perspectives. In the first one, we were interested in comparing the number of cycles that each approach would take to disseminate all α values. Our initial belief was that our approach would take less cycles to do so, which is confirmed by the results shown in Figure 15a. More precisely, it performed between 17% to 29% better, which is illustrated in Figure 15b. Simulations using the random approach, with different degree ranges and system sizes, have also been performed, with their results presented in Figure 16. Again, our approach needs always less cycles to deliver all $\alpha^i(t)$, thus demonstrating its better performance of over a wider range of scenarios.

In a second perspective, we focused on the communication costs caused by $k^i(t)$ and $x^i(t)$ in both approaches. The cost associated with $k^i(t)$, for our approach, comes from

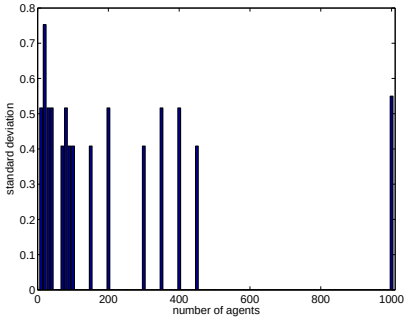


Fig. 13: Standard deviation on the number of cycles to deliver all $\alpha^i(t)$

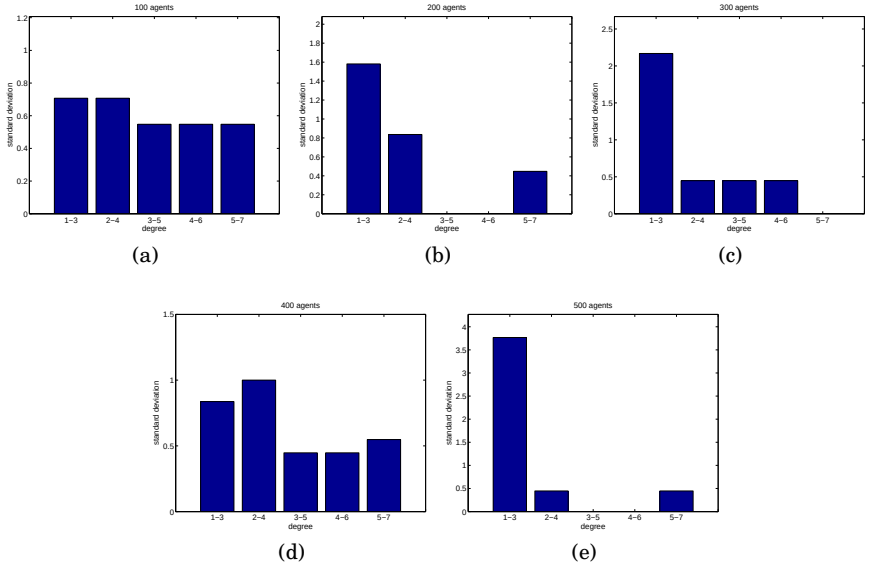
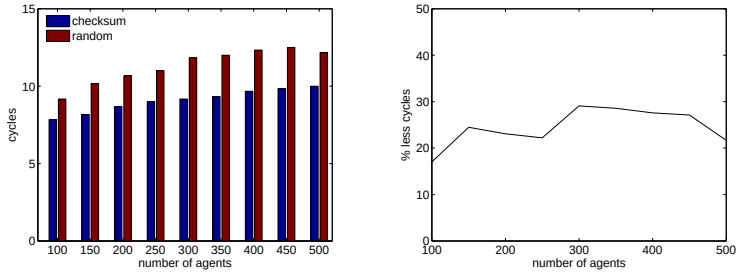


Fig. 14: Standard deviation on the number of cycles to deliver all $\alpha^i(t)$ for different system sizes and range of degrees

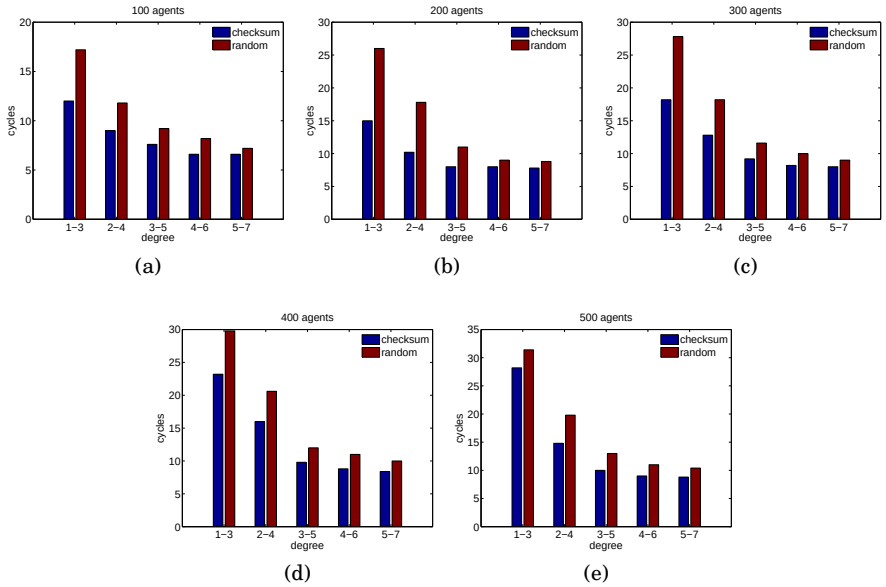
agents needing to obtain it from neighbours frequently, so as to be able to use the $g^{ij}(t)$ function (line 4 of Algorithm 4). For the random approach, the cost is similar, except that it happens only once every time the `pick()` function is invoked in the agents' algorithm. As for $x^i(t)$, the cost, for both approaches, is associated with agents having to update their own $x^i(t)$, which requires their neighbours' in turn (line 4 in Algorithm 2). Given that, we would expect our approach to have an overhead of $k^i(t)$ as low as the overhead for the random approach, at best, which is confirmed by the results presented in Figure 17a. However, because the random approach always takes more cycles to deliver all α values, the total overhead, i.e., the overhead for $k^i(t)$ plus the one for $x^i(t)$, favours our checksum approach. This is illustrated in Figure 17b.

6.2.3. Convergence. The last aspect to be evaluated from our solution is the convergence one. The idea of cycles, as introduced in the dissemination experiments, has also been used for analyzing convergence. For each experiment, a single run of the system



(a) Number of cycles to deliver all α (b) Percentage referring to how many less cycles our approach needs

Fig. 15: Number of cycles that each approach takes to deliver all α values



(a) (b) (c)

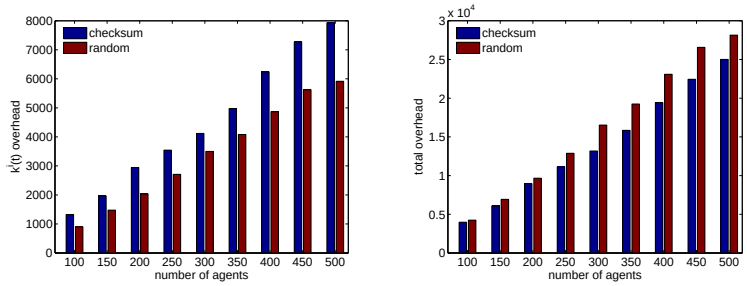
(d) (e)

Fig. 16: Comparison of *Darma*'s dissemination approach against a purely random one, for different system sizes and varying degrees

being simulated was performed. The idea with these experiments was to demonstrate, under different scenarios, the actual convergence behaviour of $x^i(t)$ and $d^i(t)$, and also that both converge to their proper values, as specified in the Termination Model.

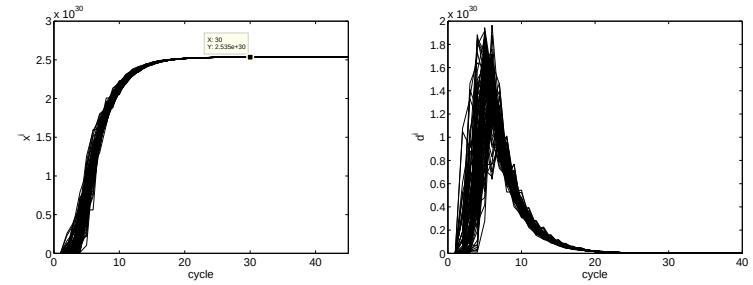
Firstly, we analyzed if $x^i(t)$ and $d^i(t)$ converge properly when the number of agents in the system as well as its topology is fixed over time. We simulated such a scenario with 100 agents, so as to ease the understanding of some of the figures we are to present. Given this setting, the results for the $x^i(t)$ and $d^i(t)$ of each agent were as presented in Figure 18. As one can see, both values converge as stated in the corresponding lemmas and the convergence analysis.

In a second scenario, we analyzed how both $x^i(t)$ and $d^i(t)$ converge when the topology of the system changes over time. For that, we have set up a scenario with 100



(a) Overhead for $k^i(t)$ (b) Overhead for $k^i(t)$ and $x^i(t)$ combined

Fig. 17: Overhead in each approach

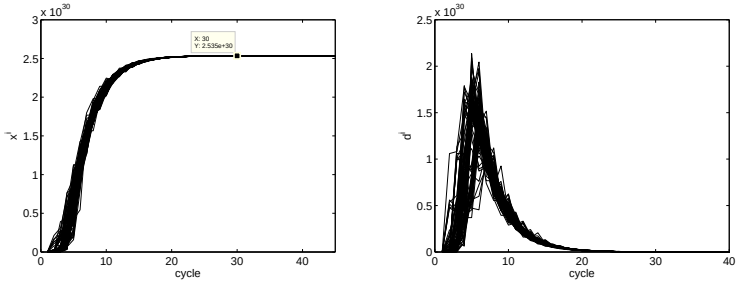


(a) (b)

Fig. 18: Convergence of $x^i(t)$ and $d^i(t)$ for 100 agents

agents, where system topology changes at cycles 3, 10, 15, and 20. In all these cases, the new topology is totally different from the previous one, but still in accordance with Definition 3.1. The results are presented in Figure 19. Note that, as in the previous scenario, $x^i(t)$ and $d^i(t)$ converge to their expected values, given the number of agents used. In particular, note that $x^i(t)$ converges to the same value as the one presented in Figure 18a, when no topology changes happened. An example of the effects caused by topology changes can be seen in Figure 20. As highlighted in the figure, one of the agents has a decrease on its $x^i(t)$. That can only happen during topology changes or agents leaving the system, since $x^i(t)$ is strictly increasing in normal scenarios. Still, once topology has changed, $x^i(t)$ follows its normal crescent behaviour, eventually reaching its proper value.

In the last of the convergence experiments, we have analyzed how $x^i(t)$ and $d^i(t)$ converge when agents leave the system. For that, we have simulated a system with 150 agents, where one third of them crash at cycle 5. The results of such an experiment are presented in Figure 21. Note that, even though the system lost one third of the agents, the convergence of $x^i(t)$ and $d^i(t)$ has not been compromised. The precise effects of agents leaving the system can be viewed in Figure 22, where the convergence curve of a particular agent's $x^i(t)$ is shown. Note that, $x^i(t)$ decreases from cycles 5 to 6, since some of the agent's neighbours are no longer in the system, consequently decreasing the value of the $\sum_{a^j \in N^i(t)} x^j(t)$ sum in Equation 16. Still, *Darma* ensures that convergence will be reached.



(a) (b)

Fig. 19: Convergence of $x^i(t)$ and $d^i(t)$ for 100 agents, under changing topologies

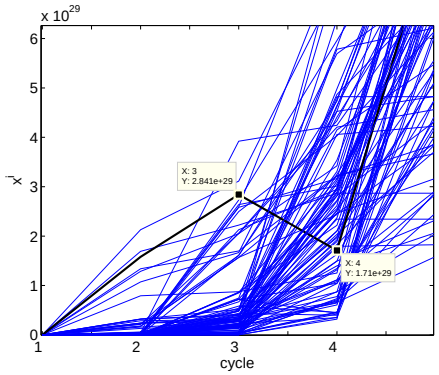
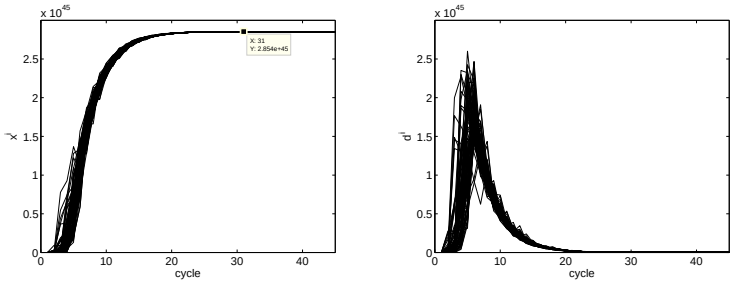


Fig. 20: Convergence of $x^i(t)$ under changing topologies, zoomed at around cycle 3



(a) (b)

Fig. 21: Convergence of $x^i(t)$ and $d^i(t)$ for 150 agents, where one third of them crash at cycle 5

7. CONCLUSIONS

In this paper we presented *Darma*, an approach for performing resource management in shared resource pools in a decentralized, adaptive, and optimal fashion. More precisely, *Darma* is focused on delivering optimal resource usage, with a best-effort approach towards satisfying individual consumer requirements. For resource

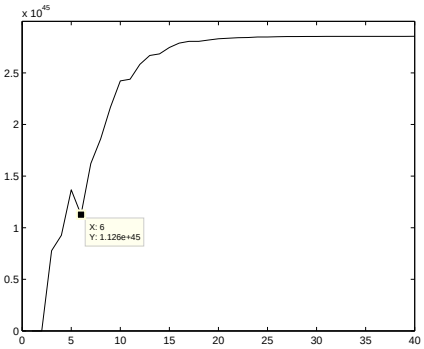


Fig. 22: Convergence of $x^i(t)$ for a particular agent, in a system with 150 agents where one third of them crash at cycle 5

pool providers, this is an important property, since it guarantees that resources are always used to their best, regardless of the situation, and allocations are pushed towards overall satisfaction. As we have showed, *Darma* relies on a set of mathematical models, which formalize the entire resource management process, regulating the interaction amongst agents in the system. These models are then implemented by a set of algorithms, executed by each agent participating in the resource management process. As discussed, *Darma* stands out due to its absolute decentralization. To the best of our knowledge, that makes it the first solution contemplating this feature in the context of optimal resource usage in shared pools.

A theoretical analysis on the mathematical models was performed, proving the correctness of its core convergence properties. Apart from that, an extensive set of experiments to validate our approach has been presented. From the optimization perspective, we showed that, in most of the cases, *Darma* is able to meet the requirements of all consumers, and still ensure optimal resource usage. In overload scenarios, or at the verge of overload, not all requirements might be met, but optimal resource usage is guaranteed. The addition and removal of consumers in between two resource management processes has also been shown to be supported by *Darma*, giving it a good degree of flexibility for real-world settings.

From the dissemination perspective, we demonstrated that our approach scales logarithmically as system size reaches hundreds to thousands of agents, in terms of the number of cycles required to disseminate all α values. Despite the exponential characteristic presented by the solution in terms of CPU time, the results demonstrate that *Darma* is expected to deliver a quick performance in a real-world setting. We have also showed that our checksum approach for choosing neighbours performs better than a purely random one, with a performance improvement ranging from 17% to 29%, in terms of cycles needed to disseminate all $\alpha^i(t)$. Also, by analyzing the total overhead caused by both approaches, we have found that the checksum approach still provided better benefits than a pure random one, thus further demonstrating its effectiveness.

The convergence aspects of *Darma* have been shown to behave as expected, in face of different scenarios. More precisely, we showed that both $x^i(t)$ and $d^i(t)$ converge to their expected values not only in a static system, but also given topology changes and agents abruptly leaving the system during the resource management process. All that clearly shows the potential of *Darma* to perform well in real-world settings, where these changes might happen unpredictably. Even though the addition of agents is not supported in this particular case, in practical terms, it is enough to allow them to

happen in between two resource management processes, which *Darma* supports, as we have showed.

As future work, we will be focusing on applying *Darma* to other shared pool scenarios, in particular for bandwidth control in cloud services. The goal with that is to further demonstrate the applicability of *Darma*, as well as the problem formulation used, in a wider range of application domains. Looking into how to address malicious agents, i.e., identifying and preventing them, is another interesting topic of study, and integrating that into *Darma* would surely be fruitful.

REFERENCES

AGUILAR, J., CERRADA, M., AND HIDROBO, F. 2007. A methodology to specify multiagent systems. In *Proceedings of the 1st KES International Symposium on Agent and Multi-Agent Systems*. Springer-Verlag, Berlin, Heidelberg, 92–101.

ARON, M., DRUSCHEL, P., AND ZWAENEPOEL, W. 2000. 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*. Vol. 28. ACM Press, New York, NY, USA, 90–101.

BABAOGLU, O. AND JELASITY, M. 2008. Self-* properties through gossiping. *Philosophical Transactions of the Royal Society 366*, 3747–3757.

BAI, X., MARINESCU, D. C., BÖLÖNI, L., SIEGEL, H. J., DALEY, R. A., AND WANG, I. J. 2008. A macroeconomic model for resource allocation in large-scale distributed systems. *Journal of Parallel Distributed Computing 68,* 2, 182–199.

BANGA, G., DRUSCHEL, P., AND MOGUL, J. C. 1999. 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, Berkeley, CA, USA, 45–58.

BATOUMA, N. AND SOURROUILLE, J.-L. 2010. Decentralized resource management using a borrowing schema. In *ACS/IEEE International Conference on Computer Systems and Applications*.

BENNANI, M. N. AND MENASCÉ, D. A. 2005. Resource allocation for autonomic data centers using analytic performance models. In *Proceedings of the Second International Conference on Autonomic Computing*. IEEE Computer Society, Washington, DC, USA, 229–240.

BOUTILIER, C., DAS, R., KEPHART, J., TESAURO, G., AND WALSH, W. 2003. Cooperative netotiation in autonomic systems using incremental utility elicitation. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*. 89–97.

BYDE, A., SALLÉ, M., AND BARTOLINI, C. 2003. Market-based resource allocation for utility data centers. Tech. rep., Hewlett-Packard. September.

CHECHETKA, A. AND SYCARA, K. 2006. No-commitment branch and bound search for distributed constraint optimization. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*. ACM, New York, NY, USA, 1427–1429.

CHEN, M., PONEC, M., SENGUPTA, S., LI, J., AND CHOU, P. A. 2008. Utility maximization in peer-to-peer systems. In *Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and Modeling of Computer Systems*. ACM, New York, NY, USA, 169–180.

DEMERS, A., GREENE, D., HAUSER, C., IRISH, W., LARSON, J., SHENKER, S., STURGIS, H., SWINEHART, D., AND TERRY, D. 1987. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of Distributed Computing*. ACM Press, New York, NY, USA, 1–12.

GMACH, D., ROLIA, J., CHERKASOVA, L., BELROSE, G., TURICCHI, T., AND KEMPER, A. 2008. An integrated approach to resource pool management: Policies, efficiency and quality metrics. In OSDI '99: Proceedings of the third symposium on Operating systems design and implementation. *Proceedings of the 2008 IEEE International Conference on Dependable Systems and Networks*, 326–335.

GUITART, J., CARRERA, D., BELTRAN, V., TORRES, J., AND AYGUADÉ, E. 2008. Dynamic CPU provisioning for self-managed secure web applications in smp hosting platforms. *Computer Networks 52,* 7, 1390–1409.

JELASITY, M., MONTRESOR, A., AND BABAOGLU, O. 2005. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems 23,* 3, 219–252.

JOHANSSON, B., ADAM, C., JOHANSSON, M., AND STADLER, R. 2006. 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, 1990–1995.

KEMPE, D., DOBRA, A., AND GEHRKE, J. 2003. Gossip-based computation of aggregate information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society, Washington, DC, USA.

KEPHART, J. O. AND CHESS, D. M. 2003. The vision of autonomic computing. *Computer 36,* 1, 41–50.

KEPHART, J. O. AND DAS, R. 2007. Achieving self-management via utility functions. *IEEE Internet Computing 11,* 1, 40–48.

KERMARREC, A. M. AND VAN STEEN, M. 2007. Gossiping in distributed systems. *Operating Systems Review 41,* 5, 2–7.

LEWIS, P. R., MARROW, P., AND YAO, X. 2008. Evolutionary market agents for resource allocation in decentralised systems. In *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature*. Springer-Verlag, Berlin, Heidelberg, 1071–1080.

LOUREIRO, E., NIXON, P., AND DOBSON, S. 2008. A fine-grained model for adaptive on-demand provisioning of CPU shares in data centers. In *Proceedings of the 3rd International Workshop on Self-Organizing Systems*. Springer-Verlag, 57–108.

LOUREIRO, E., NIXON, P., AND DOBSON, S. 2009. Decentralized utility maximization for adaptive management of shared resource pools. In *2009 International Conference on Intelligent Networking and Collaborative Systems*.

LOUREIRO, E., NIXON, P., AND DOBSON, S. 2010. Adaptive management of shared resource pools with decentralized optimization and epidemics. In *Proceedings of the 18th Euromicro Conference on Parallel, Distributed and Network-Based Processing*. IEEE Computer Society, Washington, DC, USA, 51–58.

MAHESWARAN, R. AND BAŞAR, T. 2003. Nash equilibrium and decentralized negotiation in auctioning divisible resources. *Group Decision and Negotiation 12,* 5, 361–395.

MASUISHI, T., KURIYAMA, H., OOKI, Y., AND MORI, K. 2005. Autonomous decentralized resource allocation for tracking dynamic load change. In *Proceedings of the 2005 International Symposium on Autonomous Decentralized Systems*. IEEE Computer Society, 277–283.

NEDIC, A. AND OZDAGLAR, A. 2009. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control 54,* 1, 48–61.

NOWICKI, T., SQUILLANTE, M. S., AND WU, C. W. 2005. Fundamentals of dynamic decentralized optimization in autonomic computing systems. In *Self-star Properties in Complex Information Systems*. Springer-Verlag, 204–218.

PADALA, P., SHIN, K. G., ZHU, X., UYSAL, M., WANG, Z., SINGHAL, S., MERCHANT, A., AND SALEM, K. 2007. Adaptive control of virtualized resources in utility computing environments. In *Proceedings of the 2007 European Conference on Computer Systems*. ACM Press, New York, NY, USA, 289–302.

PADGHAM, L. AND WINIKOFF, M. 2002. Prometheus: a methodology for developing intelligent agents. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*. ACM, New York, NY, USA, 37–38.

PALOMAR, D. P. AND CHIANG, M. 2006. A tutorial on decomposition methods for network utility maximization. *IEEE Journal on Selected Areas in Communications 24,* 8, 1439–1451.

PATON, N. W., DE ARAGÃO, M. A. T., LEE, K., FERNANDES, A. A. A., AND SAKELLARIOU, R. 2009. Optimizing utility in cloud computing through autonomic workload execution. *Bulletin of the Technical Committee on Data Engineering 32,* 1, 51–58.

PETCU, A. AND FALTINGS, B. 2005. A scalable method for multiagent constraint optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 266–271.

PIOVESAN, J. L., ABDALLAH, C. T., AND TANNER, H. G. 2008. A hybrid framework for resource allocation among multiple agents moving on discrete environments. *Asian Journal of Control 10,* 2, 171–186.

RAGHAVAN, B., VISHWANATH, K., RAMABHADRAN, S., YOCUM, K., AND SNOEREN, A. C. 2007. Cloud control with distributed rate limiting. *Computer Communication Review 37,* 4, 337–348.

ROLIA, J., CHERKASOVA, L., ARLITT, M., AND MACHIRAJU, V. 2006. Supporting application quality of service in shared resource pools. *Communications of the ACM 49,* 3, 55–60.

SAMAAN, N. 2008. Achieving self-management in a distributed system of autonomic but social entities. In *Proceedings of the 3rd IEEE International Workshop on Modelling Autonomic Communications Environments*. Springer-Verlag, Berlin, Heidelberg, 90–101.

TESAURO, G. AND KEPHART, J. O. 2004. Utility functions in autonomic systems. In *Proceedings of the First International Conference on Autonomic Computing*. IEEE Computer Society, Washington, DC, USA, 70–77.

TESAURO, G., WALSH, W. E., AND KEPHART, J. O. 2005. Utility-function-driven resource allocation in autonomic systems. In *Proceedings of the Second International Conference on Autonomic Computing*. IEEE Computer Society, Washington, DC, USA, 342–343.

URGAONKAR, B., SHENOY, P., CHANDRA, A., GOYAL, P., AND WOOD, T. 2008. Agile dynamic provisioning of multi-tier internet applications. *ACM Transactions on Autonomous and Adaptive Systems 3,* 1, 1–39.

WANG, X., DU, Z., CHEN, Y., AND LI, S. 2008. Virtualization-based autonomic resource management for multi-tier web applications in shared data center. *Journal of Systems and Software 81,* 9, 1591–1608.

WOOLDRIDGE, M., JENNINGS, N. R., AND KINNY, D. 2000. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems 3,* 3, 285–312.

XU, J., ZHAO, M., FORTES, J., CARPENTER, R., AND YOUSIF, M. 2008. Autonomic resource management in virtualized data centers using fuzzy logic-based approaches. *Cluster Computing 11,* 3, 213–227.