A generic learning multi-agent-system approach for Spatio-Temporal-, Thermal- and Energy-Aware Scheduling

Christina Herzog
LIUPPA, University of Pau and Adour Country,
and Efficit SAS, France
Bayonne, France
Email: herzog@efficit.com

Jean-Marc Pierson
IRIT, University of Toulouse
CNRS, INPT, UPS, UT1, UT2J
Toulouse, France
Email: jean-marc.pierson@irit.fr

***Abstract*—This paper proposes an agent based approach to the scheduling of jobs in datacenters under thermal constraints. The model encompasses both temporal and spatial aspects of the temperature evolution using a unified model, taking into account the dynamics of heat production and dissipation. Agents coordinate to eventually move jobs to the best suitable place and to adapt dynamically the frequency settings of the nodes to the best combination. Several objectives of the agents are compared under different circumstances by an extensive set of experiments.**

I. Introduction

In this research work, we tackle the problem of job placement in datacenters with the aim of reducing energy consumption and guaranteeing performances while preserving constraints on heat production. Scheduling of jobs to nodes in a datacenter is not a novel problem, and the literature is replete with works addressing job placement and scheduling. Taking into account heat production and dissipation in a datacenter is more rare. Models exist for task placement under temperature constraint of the nodes or in the room, but few include both the heat recirculation that affects datacenters (spatial dispersion of heat between nodes, using a static dispersion model) and the dynamic temporal evolution of the temperature (nodes temperature increasing when jobs are present, decreasing thanks to the cooling system, either room CRAC -computer room air conditioning- and nodes fan).

Some researchers have addressed the problem using heuristics [18] having a global knowledge of the system. While these solutions give very good results, we believe that in order to scale to exascale, a distributed approach is mandatory. Multi-Agent-Systems (MAS) have proven to be effective solutions to solve multi-objectives problems in general, as well as placement and scheduling problems in particular. Since the workload distributed to the datacenter is continuously evolving, a dynamic system able to adapt the node configuration and the job placement to the load of the system and to the evolution of the temperature is necessary. Agents following techniques such as reinforcement learning [19], [20], [21], [23], and more precisely in the context of job scheduling [22], [25], [24] allowing for an adaptive and flexible solution.

The main contributions of this paper are:

- We propose a novel architecture and algorithm for multi-agent system dynamics, self adapting to the environmental conditions
- We show how the genericity of the approach can handle multiple objectives
- We show the benefit of using dynamic adaptive approach to static ones.

The rest of this paper is organised as follows: Section II discusses related works and exhibits the novelty of our work. Section III gives an overview of the problem statement and presents the system model and its dynamics. In Section V, we propose a MAS architecture that we evaluate in Section VI. Section VII concludes the paper and discusses future work.

II. Related Work

There exists a large literature using multi agent systems for job scheduling and resource allocation, sometimes together with reinforcement techniques. For instance, in [24], Wu et al. propose a MAS using reinforcement techniques for the job scheduling problem in Grid Computing. Their ultimate goal is the load balancing on the Grid, and they favour a distributed approach where a limited explicit communication is necessary between the agents so that they can share information. In our work, no explicit communication is done between agents, they communicate only with the evolution of the environment (heat), independently of the agents behaviours. Using reinforcement learning has been widely adopted for resource scheduling, and the existing works can be categorised in two kinds: One is the policy gradient learning [26], [27] and the other uses value-function-based algorithms [28], [25]. While the first showed slow convergence and scalability issues, our approach is following this latter trend where we evaluate the past environments. In [28] there is no explicit communication or interaction between the agents: the only information that agents receive is the expected response time of a job it submitted to a particular resource, serving as the reinforcement signal. In our case, agents interact directly by sending and receiving jobs to/from others. In [29] the authors describe an agent-based power distribution approach for dynamic thermal management. However they do not consider agents exhanging jobs.

Many authors have considered thermal-aware scheduling in datacenters [30]. Most of these works rely on thermal models that capture spatial or temporal impacts or a combination of both in order to construct a thermal map of the datacenter. A spatial thermal model characterises the spatial correlation of the temperatures in different servers/outlets of a datacenter. [12] first introduced the notion of heat recirculation to capture the thermal profile of a datacenter by taking the topology and heat flow into account. [11] formally defined a heat-distribution matrix via an abstract heat flow model for the optimisation of the cooling cost of a datacenter. It has been subsequently adopted in many thermal-aware scheduling research [14]. In contrast to a spatial model, a temporal thermal model captures the temperature evolution of a single server over different time intervals. [10] were the first to apply the lumped-RC model to capture the transient evolution of temperatures in processors. Sun et al. [18] proposed a holistic thermal model that captures both spatial and temporal correlations of the temperature evolution in datacenters.

Since most thermal-aware scheduling problems are NP-complete, many heuristic solutions have been proposed with the objective of minimising the cooling cost, the energy consumption, and/or the application performance. Based on the spatial thermal model, [12] proposed, among other heuristics, MinHR, which assigns each job to a server that contributes minimally to the heat recirculation in the datacenter. [17] reduced the total energy consumption of a datacenter with server consolidation while accounting for heat recirculation, like [4], [6]. For temporal thermal models, [13] applied the lumped RC model to predict the temperatures of the servers. [5] relied on the same model to maintain the temperature threshold of the system by using DVFS while maximising the throughput.

III. Problem Statement

The problem statement is presented in this Section. The scheduling problem that we solve consist in determining which job to execute at which moment on which node, for a workload executed in typical datacenters to optimise various objectives.

- Platform: the platform is composed of a set of (heterogenous) compute nodes in a datacenter with air cooling. We consider datacenters with several rows of server racks organised with alternating cold and hot aisles. Cold air is provided by the CRAC unit and we consider the air temperature from the CRAC constant.
- Workload: it consists of a set of independent jobs, each characterised by an amount of work to perform (expressed in MI, Million Instructions), and by a power drawn from the nodes, considered constant during the execution of the job. This assumption is not that strong, i.e. if a job is having different phases consuming different powers, it can be split in different jobs from the model point of view.
- Objective: the optimisation problem to be solved can address different objectives. For example:
  - Minimise the energy consumption. The objective is to run the set of jobs using the minimal energy, expressed in Joules.
  - Minimise the makespan, the time when all the tasks in the workload are completed.
  - Maximise the energy efficiency, the amount of work performed using a single Joule.
- Constraint: the objective functions are subject to the constraint that the nodes temperature should always be maintained below a given threshold temperature.

IV. System Model and Dynamics

A. System Model

The model we use for the description of a datacenter and the temporal and spatial evolution of the temperature of the nodes is the same as in [18]. We consider a datacenter with N compute nodes. A compute node n_i is characterised by following parameters: thermal resistance R_i, thermal capacitance C_i, compute speed s_i, idle power consumption $P_{i_{IDLE}}$, a frequency set (FS_i) and a number of jobs $Capacity_i$ that can be handled at the same time (for instance the number of cores of the node).

The time is discretised, and each varying value will be indexed by time t. We consider these values as constant in the interval $[t, t + \Delta t)$, where Δt is the timestep. $T_i^{in}(t)$ is the inlet temperature of n_i at time t, while $T_i^{out}(t)$ is the outlet temperature of n_i at time t (this is the temperature of n_i itself). $P_i(t)$ is the power consumption of n_i at time t. $freq_i(t)$ is the frequency of node n_i at time t ($freq_i(t) \in FS_i$).

Given the above definitions, the "RC model" [10], [15], [13] gives the temperature evolution of n_i over time as:

$$\begin{aligned} T_i^{out}(t+\Delta t) = P_i(t)R_i + T_i^{in}(t) \\ + (T_i^{out}(t) - P_i(t)R_i - T_i^{in}(t)) \\ \times e^{-\frac{\Delta t}{R_i C_i}} \end{aligned}$$

To avoid overheating of the processor, T^{out} should be below T^{thresh}, a threshold temperature linked to the junction temperature of the chip [16]. We set $Pcrit_i$ as T^{thresh}/R_i, the maximum power a node n_i can use without overheating.

Another aspect that must be taken into account is the air recirculation between the nodes which causes the inlet temperature of a node to deviate from that provided by the CRAC unit, or specifically to increase due to the hot air recirculated from the outlets of other nodes in the datacenter. The work in Tang et al. [11] gives:

$$\overrightarrow{\boldsymbol{T^{in}}}(t) = \overrightarrow{\boldsymbol{T^{sup}}}(t) + \boldsymbol{D} \times \overrightarrow{\boldsymbol{P}}(t)$$

where $\overrightarrow{\boldsymbol{T^{in}}}(t)$ is an N-dimensional vector whose components are the $T_i^{in}(t)$'s, $\overrightarrow{\boldsymbol{T^{sup}}}(t)$ is an N-dimensional vector whose components are all equal to T^{sup}, $\overrightarrow{\boldsymbol{P}}(t)$ is an N-dimensional vector whose components are the $\overrightarrow{\boldsymbol{P}}_i(t)$'s, and $\boldsymbol{D}$ is an $N \times N$ air recirculation matrix, which is constant and precomputed for a given datacenter configuration as in Tang et al [11].

Combining the spatio-temporal model, we can get the outlet temperature for each node as:

$$\begin{aligned} \overrightarrow{\boldsymbol{T^{out}}}(t+\Delta t) = \overrightarrow{\boldsymbol{P}}(t) \times \boldsymbol{R} + \overrightarrow{\boldsymbol{T^{in}}}(t) + (\overrightarrow{\boldsymbol{T^{out}}}(t) \\ - \overrightarrow{\boldsymbol{P}}(t) \times \boldsymbol{R} - \overrightarrow{\boldsymbol{T^{in}}}(t)) \times \boldsymbol{F} \end{aligned}$$

where $\boldsymbol{R} = \mathrm{diag}(R_1, \ldots, R_N)$ and $\boldsymbol{F} = \mathrm{diag}(e^{-\frac{\Delta t}{R_1 C_1}}, \ldots, e^{-\frac{\Delta t}{R_N C_N}})$.

The number of jobs to be scheduled is J. Job j is defined by an amount of work (number of operations) w_j, a remaining work to be done at time t (rw_j^t) and by a power p_{ij} used from the node n_i when running at full speed using the full capacity of n_i.

The power consumption of n_i at time t is:

$$P_i(t) = P_{i_{IDLE}} + (freq_i(t)^3) * \sum_{j=0}^{j=J-1} (\alpha_{i,j,t} * p_{ij}) \tag{1}$$

where $\alpha_{i,j,t}$ is the fraction of node n_i used by job j at time t.

We consider the following scenario: several jobs share the same node at any time t. A job can be suspended for some time steps and resumed later, e.g., to allow the node temperature to cool down. Job migration is possible.

B. System Dynamics

At a regular pace, the system distributes the jobs from a waiting queue to each node, using a given assignment policy. Each node has a maximum number of jobs able to be run at the same time (corresponding for instance to the case of Cloud computing and a number of vcpu available on one node). Jobs enter a node either from the system, by being stolen by that node from another node, or by being migrated from another node. A job can be active or not at a given time (see Section V).

The dynamics of the system proceed as follows: at each time step τ, each node n_i makes a decision concerning its own behaviour, handling its resources and work to do (CPU frequency increase / decrease, resource dispatch between jobs, activate/deactivate jobs, migrating/stealing jobs,...). These decisions are made based on agents' probability vectors, which are set using a learning algorithm (See Section V).

Then node n_i performs some work for its jobs. If a job k is active at a time t, then it will be executed on the node and its remaining work rw_k^τ will decrease depending on the node n_i speed s_i, $freq_i(\tau)$ (the frequency of node n_i), and $\alpha_{i,j,\tau}$ (the amount of resource node n_i gave to job j at time τ). If a job is inactive, it does not consume resources and rw_k^τ will be constant for this timestep.

The power of the nodes and their temperatures are updated according to equations in SectionIV-A, meaning the influence of the other nodes are taken into account and the temperature diffuse among nodes. If the power of node n_i is above $Pcrit_i$, then node n_i deactivates the hottest jobs until the power goes under $Pcrit_i$. This is mandatory to ensure the system is viable, and not going to burn down. It is therefore possible that at a given time τ

all jobs on one node are inactive if their execution would endanger the node.

V. Multiagent Architecture

Given a specific datacenter with N nodes, we selected an architecture with N agents where one agent maps to one node. Each agent is autonomous and can decide upon actions on its own node, and interact with others by migrating jobs to, or stealing jobs from, other nodes. The key for each agent is to learn what is the best option that will optimise not only its own behaviour but also a global objective. Taking inspiration from the work of [7], we assign to each agent a probability vector whose components give the probability of taking one action or another. In this scenario, the agents are responsible for setting and evolving their own probability vector. The question is therefore to determine what objective function an agent should attempt to optimise so that the probability vector is set also to optimise the global objective of the system.

A. *Possible actions of the agents*

The different actions one agent i can take are the following:

Action 1: activate local coolest jobs ; set node frequency $freq_i$ to its minimal value ;

Action 2: activate local hottest jobs ; set node frequency $freq_i$ to its maximal value ;

Action 3: activate randomly local jobs ; if the capacity of the node n_i is not reached, ask another node for its hottest job, and steal it ; set optimal frequency to node: this is the maximum frequency the node i can run given the current active jobs and $Pcrit_i$ (see [18])

Action 4: activate randomly local jobs ; migrate the local hottest job to the coolest node ; set optimal frequency to node.

In all Actions:

- the nodes are fully used and their resources shared evenly between the active jobs (i.e. at a given time τ, on each node n_i, $\alpha_{i,j,\tau}$ is the same for all active jobs j).
- the number of jobs activated at the beginning of the actions is the number of available slots on the node n_i (depends on $Capacity_i$).
- after the action, if $P_i(t) > Pcrit_i$, then deactivate local hottest jobs until $P_i(t) <= Pcrit_i$, recompute the share of resources, and for Action 3 and 4, set the optimal frequency.

It is obvious that some Actions are better than others at certain moment: for instance, when a node becomes too hot, it is wise to choose Action 1, or Action 4 in order to decrease the load, hence the temperature locally. Conversely, when a node becomes cold, it is wise to choose Action 2, or Action 3 to increase the local performance (performing then more work). As these situations evolve during time, and may differ from node to node, the agents should be able to learn what is the wiser Action to take over time, given the current conditions.

Intuitively, Action 1 will give no difficulty to the node (minimal frequency, cooler jobs selected first so the temperature is unlikely to overpass the $Pcrit_i$ and T^{thresh} thresholds), but the progress is slow due to the minimal frequency. Conversely, Action 2 will stress the node, using it at maximum performance and processing the hottest jobs first, i.e. it is likely that the thresholds are reached or surpassed, meaning some jobs will be obviously deactivated, making remaining jobs getting more resources, while other are stalled. Action 3 will favour a cooperative behaviour, helping other nodes while Action 4 will favour a selfish behaviour. Other Actions could easily be added, but we refrained to 4 Actions so that we can understand clearly and see the different evolutions depending on the objectives.

For the dynamics of the system evolution, we distinguish between two time scales. τ is the timestep the system will operate (nodes performing job computation, and adjusting according to the decision set by the agents). t will be the time period at which the agents operate, meaning observing their objectives and setting the actions performed by their node during the next observation period. It is important that $t >> \tau$, so that an agent can actually see the impact of a setting during a long enough observation period. An agent changes therefore its probability vector at each period t (meaning after a number of τ timesteps), and the agent will learn after each t time period.

The pseudo-code of the algorithm for each agent is finally given in Algorithm 1. Every t timesteps an agent i computes the target objective (see Section V-C for the different objectives), and adapts its policy thanks to the learning algorithm (see Section V-D for the details of the learning algorithm). Then, depending on the policy learnt, the agent activates jobs and updates the frequency of the node (one of the four Actions previously stated). Finally, it computes the amount of resources it can give to each job, and updates the power of the node. If the power is exceeding the critical power $Pcrit_i$, the agent deactivates the hotter job, recomputes the resources allocated to remaining jobs and updates the power until the power is acceptable.

Algorithm 1 Pseudo-code for agent i

```
procedure AGENTBEHAVIOUR(i)
    for Ever do
        if $time\%t == 0$ then
            obj = ComputeObjective()
            policy = AdaptNewBehaviour(obj)
        end if
        UpdateJobs(policy)
        UpdateFrequency(policy)
        AdjustJobsAlphaToNode()
        P = UpdatePowerNode()
        while $P > Pcrit_i$ do
            DeactivateHotterJob()
            AdjustJobsAlphaToNode()
            P = UpdatePowerNode()
        end while
    end for
end procedure
```

B. State of the system and Global Objective

The state of the full system, z_t, at time t is:

$$z_t = \{(j_0, rw_0^t), ...(j_k, rw_k^t), ...(j_{J-1}, rw_{J-1}^t)\}$$

where j_k identifies job k, rw_k^t identifies the remaining work of job k at time t.

The objective $G(z_t)$ of the system is to maximise, at each time t, its energy efficiency, meaning making the maximum work with the less overall energy.

More precisely, with z_t the state of the system at time t ($t > 0$), then:

$$G(z_t) = \frac{\sum_{k=0}^{J-1}(rw_k^t - rw_k^{t-1})}{\sum_{tt=t-1}^{t}\sum_{i=0}^{N-1} P_i(tt)} \tag{2}$$

Intuitively, this equation gives the energy efficiency of the system during the last time period t, i.e. during the last τ timesteps. Recall that time t is a window of time, not a single timestep from the point of view of the nodes and the jobs.

C. Agent Objectives

In this research, we investigated several types of agent objectives. Each was used exactly in the same manner, the same configuration, the same learning algorithm. Only the objectives the agents are trying to optimise are different and will give different system performances.

1) The first agent objective is the global objective given in equation (2). With this objective, each agent attempts to optimise directly the global objective. One can see that, by definition, if all agents (each responsible for one node) succeed in optimising their own objectives, then the global objective will be optimised. However this has been shown suboptimal for large systems because of the mutual influences of the agents (in our case: sending and receiving jobs, heat dispersion,...) and provides good solutions only for small systems [8], [9].
2) The second agent objective is the difference objective, that aims at isolating the impact of that agent on the system [8], [9]. This is done by computing the difference between the global objective and the global objective that would be achieved if the agent would be removed from the system ($G(z_t^{-i})$). The resulting objective D_i for agent i is:

$$\begin{aligned} D_i(t) &= G(z_t) - G(z_t^{-i}) \\ &= \frac{\sum_{k=0}^{J-1}(rw_k^t - rw_k^{t-1})}{\sum_{tt=t-1}^{t}\sum_{l=0}^{N-1} P_l(tt)} \\ &\quad - \frac{\sum_{k=0}^{J-1}((rw_k^t - rw_k^{t-1}) * e_{ik}^t)}{\sum_{tt=t-1}^{t}\sum_{l=0}^{N-1} P_l^*(tt)} \end{aligned}$$

where $e_{ik}^t = 0$ if job k is active on node n_i at time t, 1 otherwise, and $P_l^*(tt) = P_l(tt)$ for all $l != i$, and $P_i^*(tt) = 0$.
3) The third agent objective is the selfish objective, where the agents are only concerned by optimising their own energy efficiency, meaning:

$$\begin{aligned} &S_i(t) \\ &= \frac{\sum_{k=0}^{J-1}((rw_k^t - rw_k^{t-1}) * (1 - e_{ik}^t))}{\sum_{tt=t-1}^{t} P_i(tt)} \end{aligned}$$

D. Agent Learning

As mentioned above, we have two time scales. The agents perform their operations at each time period $t = 40\tau$. An Action that has been decided by an agent is therefore valid for 40τ timesteps from the point of view of the nodes they are responsible for. It means the probability vector are fixed for this time period. At the end of each time period t, the objectives functions are computed and recorded in the agents' training sets. In order to be able to compare the system behaviour due to the decisions of the agents, and therefore the performance of the individual probability vectors, all jobs are considered not scheduled after each period t. During an initial phase (lasting 20 periods in our experiment), the probability vectors are set completely randomly. This is the initial training set of the agents. After this initial phase, the agents use the following learning algorithm to set their probability vectors that will be valid for the next time period t.

The learning algorithm is inspired from the work of [7], with adjustments to our case. The idea of updating a probability vector according to a given objective distance is taken from [7]. However, their approach only considered the allocation of jobs to resources while we reconsider the placement (migrating jobs) and the node configuration (frequency setting) during the life of the platform. Moreover the details of the parameters are given in our work, facilitating the reproducibility of our approach. First the algorithm proceeds by generating a set of 10 candidate probability vectors, following a Gaussian distribution about the current probability vector, using a mean of 0.2 and a standard-deviation of 0.1. For each candidate vector $\overrightarrow{p_T}$, an expected resulting objective $O(\overrightarrow{p_T})$ is estimated using the last items in the agents training set. Our experiments showed that using 20 items in the history is enough, because older values from the history have a too little impact. The objective values from the history are weighted by both how long ago this value was recorded (to favour fresh data, more adapted to the current environmental conditions of the agents) and the distance between the candidate vector and the current probability vector (in order to keep close enough probability vector from one period to the next).

$$O(\overrightarrow{p_T}) = \frac{\sum_{t=T-20}^{t=T} O_t e^{-\alpha_T (T-t)} e^{-\alpha_D \|\overrightarrow{p_T} - \overrightarrow{p_t}\|}}{\sum_{t=T-20}^{t=T} e^{-\alpha_T (T-t)} e^{-\alpha_D \|\overrightarrow{p_T} - \overrightarrow{p_t}\|}}$$

where T is the current learning period, t is the period that resulted in objective O_t, $\overrightarrow{p_T}$ is the considered candidate vector, p_t is the probability vector that resulted in objective O_t. α_T and α_D are system parameters, set to 0.1 and 10, respectively. Depending on the agents objective chosen for the experiment, O_t is given by G, D_i, or S_i. The new probability vector is simply the one with the best value (i.e. maximal values if we want to maximise an objective, minimal otherwise) among all candidates.

VI. Evaluation

A. Experimental settings

The described MAS has been implemented in Netlogo 5.3.1 [1]. This simulation framework allows to implement agents and to follow their evolution. It has been used in many different fields, from sustainability, economics, agriculture to biology [3], [2].

The number of nodes is $N = 20$ (homogeneous nodes), the number of jobs is $J = 1000$. Following the works of [18], the parameters of the simulation and the thresholds are set as following: the heat dispersion matrix is the same as in [6]; the parameters for all nodes are the same: the frequency set is $\forall i, FS_i = (0.60.7330.8661)$, $R_i = 0.7$, $C_i = 2.06$, $Capacity_i = 10$, $Tthread = 80°C$, $Pcrit_i = 80/0.7 = 114$ Watts, $P_{i_{IDLE}} = 20$ Watts, and $T^{sup} = 25°C$. The work w_k for job k is set randomly following a uniform distribution in $[100, 500]$. The dynamic power p_{ik} of job k is set randomly in $[30, 94]$, so that a job can run on any node n_i, if it is alone on that node. The processing power of all nodes is set to 1 (1 unit of work per time step), and the nodes don't shut down.

The scheduling frequency is set to 1, i.e. at each timestep τ, an assignment policy is started to distribute the jobs. With this scheduling frequency, a node is never let without jobs to process and its full capacity is used at all moments, at least until no more jobs are to be processed. In this paper, the jobs are assigned randomly to the nodes.

The learning frequency is set to 40τ. The different possible Actions have some impacts that will show their benefit after a short time. For instance, when a node policy is set to Action 4, then for τ timesteps it will migrate its hottest job to another node. Since the capacity of a node is limited to $Capacity_i$ (set to 10 in our setting), then after 10 timesteps the node has no more jobs (except if another node with also Action 4 sends it some new job), and it can cool down (making it an even better candidate to receive a new job, if another node follows Action 4). The same holds for the other Actions in our settings.

B. Evaluation Metrics

An experiment finishes when all the N jobs have been processed. For each random seed, the same set of jobs is processed using different objectives. To the objectives described in Section V-C, we added the following ones where agents have static behaviour, for comparison purpose:

- minimum frequency: the agent sets the node frequency to its minimum
- maximum frequency: the agent sets the node frequency to its maximum
- random behaviour: the agent sets the action randomly among the four Actions given in Section V-A.

In order to compare the performance of the different objectives, we decided to use the following metrics:

- The makespan M of the system, meaning the time when all jobs have completed.
- The energy $E(M)$ of the system, meaning the energy used to perform all the jobs, with $E(t) = \sum_{tt=1}^{t} \sum_{i=1}^{i=N} P_i(tt)$. E(t) is the energy consumed until time t.
- The energy efficiency $EE(M)$ of the system, computed as the total amount of work performed divided by the energy used to process it at the end of the simulation, with $EE(t) = \frac{\sum_{k=1}^{k=J}(w_k - rw_k)}{E(t)}$. EE(t) is the energy efficiency achieved until time t.

The results presented in this Section are averaged over 10 executions. The standard-deviations of the reported values are very small (less than 2%).

C. Results

1) Objective evolution: Figure 1 shows the evolution of the global objective given in equation (2) during the experiments (Y axis) while the X-axis is the time in terms of time period t, i.e. a measure is taken at each learning phase (and not at every single step τ). The legends OGE, ODE, OSE stand for Optimise G objectivE (G), Optimise D objectivE (D_i) and Optimise Selfish objectivE (S_i), respectively. We can see that the worst is the *MaxFreq* while the best is *MinFreq*. Since there is no time limit to finish the jobs, it is obvious from equation (1) linking power (thus energy) to frequency (thus performance) that the best strategy is to minimise the frequency. Besides, the OGE strategy for agents, optimising directly the global objective G, gives only small benefit compared to the random strategy. Finally, when the agents optimise D_i and S_i, they outperforms G significantly. With more jobs to handle (remember here the number of jobs is fixed to 1000), we can foresee that ODE and OSE strategies will continue to increase the energy efficiency of the system.

Another observation is that, with all strategies, the global objective G is decreasing at the very end of the experiment. This is normal since at the end, the work to be done is limited, and the nodes maybe empty, still consuming P_{IDLE} but not producing any work.

On Figures 2 and 3, one can see the final values of the makespan M and the energy efficiency EE. The makespan M (shown here in terms of timesteps τ) is obviously better with *maxfreq* than other strategies, since, when possible, the fastest frequency is selected. Conversely, the energy E (not shown here) is lower with the minimum frequency. ODE gives a slightly better makespan, but consumes a little more energy than OSE. Compared to OGE, both ODE and OSE give higher makespan and lower energy. This translates to better energy efficiencies EE for ODE and OSE compared to OGE, meaning the agents altogether use the energy in a better way when having a selfish behaviour (OSE) and when isolating their contribution to the global energy (ODE), instead of directly trying to optimise the global objective G.

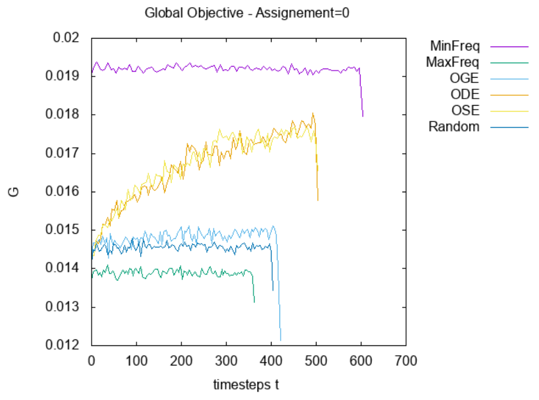

Fig. 1: Evolution of the global objectif G during the experiments when agents follow the six different objectives

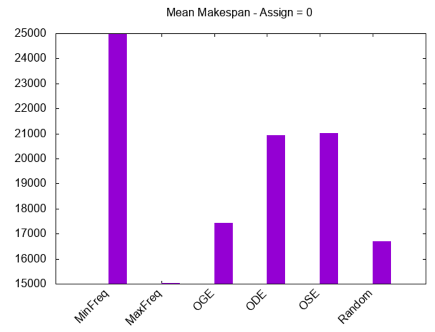

Fig. 2: Makespan using the different strategies for agents

2) Impact of learning frequency: To better understand the impact of the learning strategy, we repeated the experiments, changing only the learning frequency to 20τ and 200τ, instead of the original 40τ (used in previous section VI-C1). Figure 4 shows the evolution of the global objective G using these frequencies. Please note that the x-axis represents the periods t, not the single timesteps τ (that's why the duration, expressed in periods, changes that much). Using a high learning frequency (20τ) we can see that the global objective is reaching higher values faster than using smaller learning

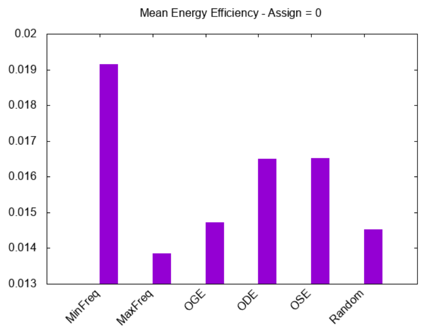

Fig. 3: Energy Efficiency using the different strategies for agents

frequencies, whatever the agents' objective, but it is more noticeable with ODE and OSE. Indeed when the learning frequency is small, the agents take a lot of time to learn the optimal behaviour. Using the higher learning frequency, the global objectives reaches a limit and seems to increase only slowly after the initial increase.

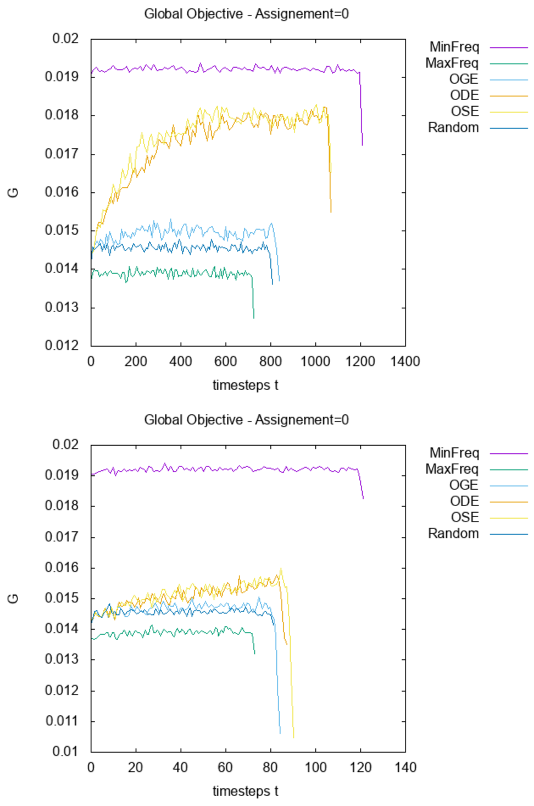

Fig. 4: Evolution of the global objectif G during the experiments when agents follow the six different objectives, using learning frequency 20 (top) and 200 (down)

The makespan, energy, energy efficiency metrics have also been computed. By lack of space, only the conclusions are given here. The makespan is reduced using a lower learning frequency (200τ). This is probably due to the fact that the jobs progress without being interrupted by new learning phases, and altogether the jobs finish earlier. The energy used is much higher, leading to a better energy efficiency with a higher frequency (20τ).

3) Impact of scheduling frequency: We tested several different scheduling frequencies, besides 1, the default in the previous experiments. The evolution of G with different scheduling frequencies (1, 20 and 40 timesteps) has been studied. With a scheduling every 40 timesteps, it is synchronised with the learning frequency, and the scheduling is therefore done only once per time period. With 20, it is done twice. However the differences are almost not visible when using the random assignment. The same goes with the other metrics for makespan, energy, energy efficiency.

4) Scalability issues and complexity analysis: The experimental setup represents a small size datacenter, with 20 nodes. Experiments extended to 50 nodes, for which a heat dispersion matrix can be found [6], have shown the same results' tendencies, taking more time. A typical experiment with 20 nodes and 1000 jobs lasts no longer than 1 minute (with plotting enabled in NetLogo), but repeating experiments for each case (different seeds, objectives, scheduling and learning frequencies) multiplies the total time of experiments. Larger scale experiments suffer from the lack of available heat dispersion matrix and without it the model would be limited. Considering the jobs, they have only an impact on the length of the experiment, since at the same instant only $Capacity_i$ jobs can be processed per node n_i, limiting the parallelism of job execution. At each timestep, the complexity of job execution is therefore $O(Max_i Capacity_i)$ on each node. Finally, at each timestep, each agent is potentially searching a job from another node, or transferring a job to another node, leading to a $O(N^2)$ complexity in the worst case (when each agent is acting according to Action 3 or Action 4).

VII. Conclusion and Future Works

In this paper we proposed a multi-agent-system for the scheduling of independent jobs. The scheduling takes into account the heat produced and disseminated in the datacenter room. Agents attached to nodes choose the best action to perform among a set of predefined actions, depending on the node current environment and global objective.

A learning mechanism allows the agents to adapt to evolving conditions. In general, better results are obtained when the agents do not try to optimise directly the global objective (OGE) of energy efficiency, but when they act either in a selfish way (OSE strategy), or by taking into account the benefit to the system of their own behaviour (ODE strategy). More importantly, we developed a multi-agent simulation tool that can be reused to test other strategies or other environmental conditions.

As future works, we will add a deadline to the jobs, so that the order of processing them has an impact. We will have to redefine the global objective accordingly so that it takes into account also the number of missed deadlines, if any. Second, we plan to attach one agent to each job, so that such agents can also control the progress of the jobs, independently of the nodes. They will be conflicting objectives with the nodes, and the question of which global objective should be optimised is still to be understood.

References

[1] U. Wilensky, Netlogo itself 1999. NetLogo. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.

[2] U. Wilensky, W. Rand, An Introduction to Agent-Based Modeling Modeling. Natural, Social, and Engineered Complex Systems with NetLogo, MIT Press., 2015.

[3] C. Herzog, JM. Pierson, L. Lefvre. A Multi Agent System for Understanding the Impact of Technology Transfer Offices in Green-IT Principles and Practice of Multi-Agent Systems (PRIMA 2016), Springer-Verlag, LNAI 9862, p. 323-332.

[4] H. Sun, P. Stolf, JM. Pierson, and G. Da Costa. Energy-efficient and thermal-aware resource management for heterogeneous datacenters. *Sustainable Computing: Informatics and Systems*, 4(4):292–306, 2014.

[5] D. Rajan and P. S. Yu. Temperature-aware scheduling: When is system-throttling good enough? International Conference on Web-Age Information Management (WAIM), 2008.

[6] T. Mukherjee, A. Banerjee, G. Varsamopoulos, S. K. S. Gupta, and S. Rungta. Spatio-temporal thermal-aware job scheduling to minimize energy consumption in virtualized heterogeneous data centers. *Computer Networks*, 53(17):2888–2904, 2009.

[7] Tumer, K. and Lawson, J., Coordinating Learning Agents for Multiple Resource Job Scheduling Adaptive and Learning Agents: Second Workshop, ALA 2009, Held as Part of the AAMAS 2009 Conference, 2009. Revised Selected Papers, 2010, pp123–140

[8] A. K. Agogino and K. Tumer, Analyzing and visualizing multiagent rewards in dynamic and stochastic domains, Journal Autonomous Agents and Multi-Agent Systems, 17(2), 2008

[9] K. Tumer, A. K. Agogino, and D. H. Wolpert. Learning sequences of actions in collectives of autonomous agents. 1st Int. joint conference on Autonomous agents and multiagent systems (AAMAS '02). ACM

[10] K. Skadron, T. Abdelzaher, and M. R. Stan. Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management. International Symposium on High-Performance Computer Architecture (HPCA), 2002.

[11] Q. Tang, S. K. S. Gupta, and G. Varsamopoulos. Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach. *IEEE Transactions on Parallel and Distributed Systems*, 19(11):1458–1472, 2008.

[12] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making scheduling "cool": temperature-aware workload placement in data centers. USENIX Conference, 2005.

[13] L. Wang, S. U. Khan, and J. Dayal. Thermal aware workload placement with task-temperature profiles in a data center. *J. of Supercomputing*, 61(3):780–803, 2012.

[14] A. Sansottera and P. Cremonesi. Cooling-aware workload placement with performance constraints. *Performance Evaluation*, 68(11):1232-1246, 2011.

[15] S. Zhang and K. S. Chatha. Approximation algorithm for the temperature-aware scheduling problem. IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Jose, USA, 2007.

[16] K. Ebrahimi, G. F. Jones, and A. S. Fleischer. A review of data center cooling technology, operating conditions and the corresponding low-grade waste heat recovery opportunities. *Renewable and Sustainable Energy Reviews*, 31(C):622-638, 2014.

[17] E. Pakbaznia and M. Pedram. Minimizing data center cooling and server power costs. ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED), 2009.

[18] H. Sun, P. Stolf, and J.-M. Pierson. Spatio-temporal thermal-aware scheduling for homogeneous high-performance computing datacenters. *Future Gener. Comput. Syst.*, 71C:157-170, 2017.

[19] J. Bredin, M., Rajiv T., C. Imer, T. Basar, D. Kotz and D. Rus Game-theoretic formulation of multi-agent resource allocation International Conference on Autonomous Agents, pp 349-356, 2000

[20] L. Pack Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: a survey. J. Artif. Int. Res. 4, 1, 1996, 237-285.

[21] R. S. Sutton and A. G. Barto Reinforcement Learning: An Introduction The MIT Press, 1998

[22] Busoniu, L., Babuska, R. and De Schutter, B. A Comprehensive Survey of Multiagent Reinforcement Learning Trans. Sys. Man Cyber Part C, 2008

[23] Reinforcement Learning: State-of-the-Art Editors: M. Wiering, Martijn van Otterlo, Springer, 2012

[24] J. Wu, X. Xu, P. Zhang and C. Liu A novel multi-agent reinforcement learning approach for job scheduling in Grid computing Future Generation Computer Systems, 27(5), 2011, 430-439

[25] G. Tesauro, N.K. Jong, R. Das, Mohamed, and N. Bennani A hybrid reinforcement learning approach to autonomic resource allocation ICAC 06, 2006, pp. 65-73.

[26] Sherief Abdallah and Victor Lesser. Learning the task allocation game. Proceedings of the 5th international joint conference on Autonomous agents and multiagent systems (AAMAS '06). ACM, pp850-857

[27] Zhang, C., Lesser, V. R and Shenoy, P. J. A Multi-Agent Learning Approach to Online Distributed Resource Allocation IJCAI, volume 9, pp 361–366, 2009

[28] Galstyan, A., Czajkowski, K. and Lerman, K. Resource Allocation in the Grid with Learning Agents Journal of Grid Computing, 2005, Volume 3, pp 91–100, Jun 2005

[29] T. Ebi, M. Faruque, and J. Henkel TAPE: thermal-aware agent-based power economy multi/many-core architectures IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2009, pp. 302-309

[30] H. F. Sheikh, I. Ahmad, Z. Wang and S. Ranka An overview and classification of thermal-aware scheduling techniques for multi-core processing systems Sustainable Computing: Informatics and Systems, Volume 2 (3), pp 151–169, 2012