A Decentralized Algorithm to Revisit the Debate of Centralization and Decentralization Approaches for Cloud Scheduling

Cheikhou Thiam[1], Georges Da Costa[2] and Jean-Marc Pierson[2]
[1]*Université de Thiés, Thiés, Senegal*
[2]*IRIT, Toulouse Computer Science Institute, Toulouse University, France*

Keywords: Energy, Heuristic, Virtual Machines, Cloud federation, Migration.

Abstract: In Cloud Computing, scheduling jobs is a major and difficult issue. The main objectives of cloud services providers are the efficient use of their computing resources. Existing cloud management systems are mostly based on centralized architectures and energy management mechanisms are suffering several limitations. To address these limitations, our contribution is to design, implement, and evaluate a novel cloud management system which provides a holistic energy-efficient VM management solution by integrating advanced VM management mechanisms such as underload mitigation, VM consolidation, and power management. In this paper, we introduce a distributed task scheduling algorithm for Clouds that enables to schedule VMs cooperatively and dynamically inside a federation of clouds. We evaluated our prototype through simulations, to compare our decentralized approach with a centralized one. Our results showed that the proposed scheduler is very reactive.

1 INTRODUCTION

Nowadays, most of the design and implementation of energy-efficient cloud management systems are mostly based on centralized architectures. To overcome issues bottlenecks such as overloading, under loading and impractical unique administrative management, which are normally led by conventional centralized or hierarchical schemes, the distributed scheduling scheme is emerging as a promising approach because of its capability with regards to scalability and flexibility particularly for federation of clouds. Energy management mechanisms are suffering several limitations. Many studies have been made precisely to identify the parameters with the largest impact on the total energy consumption (Khosravi et al., 2017) (Tseng et al., 2017) and to distribute dynamically the jobs (Brant et al., 2017) (Rouzaud-Cornabas, 2010) (Yazir et al., 2010) (Hermenier et al., 2010). In this paper, we introduce a distributed task scheduling algorithm for Cloud that enables to schedule and manage VMs cooperatively and dynamically in federation clouds. Our contribution is also to design, implement, and evaluate a novel cloud management system which provides a holistic energy-efficient point of view which avoids resource overprovisioning. Decentralized algorithms solve the main shortcomings of centralized algorithms such as scalability, fault tolerance and bottlenecks which can significantly degrade performance, the adequacy of the cloud computing environment and autonomy. In centralized scheduling, one cloud scheduler maintains a complete control over the clusters. All the jobs are submitted through the cloud scheduler. In contrast, in decentralized scheduling, organizations maintain (limited) control over their schedules. Jobs are submitted locally, but they can be migrated to another cluster, if the local cluster is overloaded. The possibilities of migration are, however, limited, so that migrated jobs do not overload the network and the node themselves. The aim of this article is to compare energy consumed by centralized algorithm and decentralized algorithm. In this paper, we compare both classes of scheduling algorithms, centralized and decentralized ones. This article is structured as follows : Section 1 introduces the design principles. Section 2 presents severel studies in the domain. Section 3 presents the decentralized approach. Section 4 presents results, i.e. a comparison of a centralized algorithm (Energy aware clouds scheduling using anti load-balancing algorithm (EACAB)(Thiam et al., 2014) named here Centralized) and the Decentralized algorithms. Finally, Section 5 summarizes the contribution and proposes future works.

Thiam, C., Da Costa, G. and Pierson, J.
A Decentralized Algorithm to Revisit the Debate of Centralization and Decentralization Approaches for Cloud Scheduling.
DOI: 10.5220/0006682901650172
In *Proceedings of the 7th International Conference on Smart Cities and Green ICT Systems (SMARTGREENS 2018)*, pages 165-172
ISBN: 978-989-758-292-9

2 RELATED WORK

The decentralized meta-scheduling scheme allows each cluster to own a meta-scheduler that receives job submissions originated by local users, and to assign such jobs to the local resource management system, i.e., local scheduler. Besides the issue of efficiency and overhead, the Decentralized algorithm scheme brings better scalability, compared to other scheduling schemes. Distributed VM consolidation algorithms enable the natural scaling of the system when new nodes are added, which is essential for large-scale Cloud providers (Pattanaik, 2016). Another benefit of distributing VM consolidation algorithms is the improved fault tolerance by eliminating single points of failure: even if a server or controller node fails, it would not render the whole system inoperable. The dynamic scheduling of a large number of VMs as part of a large distributed infrastructure is subject to important and hard scalability problems that become even worse when VM image transfers have to be managed. Consequently, most current algorithm schedule VMs statically using a centralized control strategy. An analysis of virtual infrastructure managers (VIMs) reveals that most of them still schedule VMs in a static manner. Indeed, the advanced solutions that promote the implementation of dynamic VM scheduling (Khosravi et al., 2017) are subject to strong limitations regarding scalability, reactivity, and fault-tolerance aspects. The choice of a single master node leads to several problems. First, during the computation and the application of a schedule, this manager does not enforce QoS properties anymore, and thus cannot react quickly to QoS violations. Second, because the manipulation of VMs is costly, the time needed to apply a new schedule is particularly important: the longer the reconfiguration process, the higher the risk that the schedule may be outdated, due to the workload fluctuations, when it is eventually applied. Finally, a single master node can lead to well-known fault-tolerance issues or a node can be overloaded; a subgroup of VMs may be temporarily isolated from the master node in case of a network disconnection; QoS properties may not be ensured any more if the master node crashes. Some nodes could be overloaded which increases the energy consumed. Even if a better design of the master can help to separate each phase in order to resolve some of these issues (a multi-threaded model, for instance, can help to track workload changes so that a scheduling process can be stopped if needed), a centralized approach will always be subject to scalability, reactivity, and fault-tolerance issues. In this paper, we investigate whether a more decentralized algorithm approach can tackle the aforementioned limitations. Indeed, scheduling takes less time if the work is distributed among several nodes, and the failure of a node does not impede scheduling strongly any more. Several proposals have been made precisely to distribute dynamic VM management (Yin et al., 2016) (Yang et al., 2016) (Feller et al., 2010) (Yazir et al., 2010). However, the resulting prototypes are still partially centralized. Firstly, at least one node has access to a global view of the system. Secondly, several virtual infrastructure managers (VIMs) consider all nodes for scheduling, which limits scalability. Thirdly, several VIMs still rely on service nodes, potential single points of failure. (Quesnel and Lèbre, 2011) introduces distributed VM scheduler (DVMS), a VIM that schedules and manages VMs cooperatively and dynamically in distributed systems. Author designed it to be nonpredictive and event-driven, to work with partial views of the system, without any potential single points of failure. Our VIM thus has the same characteristics and is more reactive, more scalable, and more tolerant to nodes crashes or network disconnections. Kang and Choo (Kang and Choo, 2016) introduced an Inter Cloud Manager (ICM) job dispatching decentralized algorithm which operates well in large scale environments. Authors in (Mehrsai et al., 2017) propose a hybrid algorithm which is a coalition between both techniques (hybrid centralized and decentralized). The proposed approach can effectively adhere to strict QoS requirements, as well as handle multi-core CPU architectures, heterogeneous infrastructure and heterogeneous VMs.

3 A DECENTRALIZED TASK SCHEDULING ALGORITHM FOR CLOUD

The algorithm uses a multi-phase decentralized algorithm scheduling solution. It is comprised of a migration phase, a job submission phase responsible for job dissemination, as well as the dynamic scheduling phase responsible for iterative scheduling improvement. Furthermore, the decentralized algorithm is a collection of implementation independent interfaces and heuristics, which are used to facilitate job scheduling across distributed nodes.

3.1 Algorithm Statements

There are in total N sites and in each site a set H_i of nH_i nodes distributed in a cloud data center system with the same start time U. $H_{i,j}$ is the node j in site i. Each time when one node (initiator) attempts to

(re)assign a task to another node (or the same node) for execution, the initiator is called the requester node, and the node receiving such a request is called the responder node. Each task $VM_{i,j,k}$ i $\in [1,2,...,N]$ and $j \in [1,2,...,nH_i]$ is sent from node $H_{i,j}$. The decentralized algorithm is comprised of two phases, namely the job submission phase and the dynamic scheduling phase, which work together to ensure both a quick job distribution and an optimized rescheduling effect.

- Job submission phase. This phase is the first phase of the algorithm. Each time a node j, receives a $VM_{i,j,k}$ submitted by its local user, node j behaves as a requester node $H^{req}_{i,j}$ and generates a request message $VM^{req}_{i,j,k}$ for $VM_{i,j,k}$. VM characteristic information including estimated execution time $D_{i,j,k}$ and requested amount of Processor Elements (PEs) $VM^{pe}_{i,j,k}$ will be appended to the generated request message. Afterwards, request message $VM^{req}_{i,j,k}$ is replicated and disseminated to each of the discovered remote nodes asking for the job delegation possibilities. All nodes receiving the job delegation request message $VM^{req}_{i,j,k}$, including the requester node j itself, are considered as responder nodes. Each responder node $H^{rep}_{i',j'}$ needs to send an accept message $VM^{ack}_{i,j,k}$ whether the node j' is able and willing to execute the received $VM_{i,j,k}$. A $VM^{req}_{i,j,k}$ takes various factors, such as the current state of the node which allows it to execute the VM using the most efficient amount of energy relative to other nodes. If yes, each candidate, considered as responder node, computes an estimated completion time according to its current scheduling, mainly the energy consumed and resource status and delivers the information by means of an accept message $VM^{ack}_{i,j,k}$. In addition, the estimated response time and the estimated energy consumed if $VM_{i,j,k}$ is executed by node j' in site i' are also appended to the generated accept message, which can be utilized by the requester node for responder node evaluation and selection. Each time a request message $VM^{req}_{i,j,k}$ is generated by the requester node and disseminated to contactable remote nodes, the requester node waits and collects all received $VM^{ack}_{i,j,k}$, generates an assign message $VM^{ass}_{i,j,k}$ and send it to a proper remote node $H^{ass}_{i',j'}$ (assignee node) to which the $VM_{i,j,k}$ is delegated. An arbitrary node j in site i, due to the effect of the job submission phase, has received a set of jobs from either local users' submissions or remote nodes' delegations. A rescheduling process helps this approach to adapt to the changes of both underlying resources and arriving jobs.
- Dynamic scheduling phase. This phase solves some problems related to the ever changing data center infrastructure during VM submission phase. It allows for example a redistribution algorithm for a VM that is in a long tail and thus a node can not be executed instantly. Suppose the arbitrary node $H_{i,j}$ finds that many customers applications (VMs) wait to be served, such selected customers will be sent to others nodes (rescheduling) to improve their own VM makespans and the overall resource utilization, therefore reducing the energy consumed of the overall cloud data center. The number of to-reschedule VMs is computed by considering the status of node $H_{i,j}$ itself, such as the length of local waiting customers and current node overhead. Afterwards, the algorithm finds a set of contactable remote nodes for VM rescheduling and re-allocation. $H_{i,j}$ sends an inform message $VM^{info}_{i,j,k}$ for each to-schedule $VM_{i,j,k}$, and disseminates those inform messages to all remote nodes discovered for negotiating VM rescheduling possibilities. Each generated message $VM^{info}_{i,j,k}$, contains firstly the same information as the aforementioned request message, including estimated execution time $D_{i,j,k}$, requested amount of PEs $VM^{pe}_{i,j,k}$, and job characteristic profile. Furthermore, each inform message also includes the already made schedule for $VM_{i,j,k}$, on the current node $H_{i,j}$, i.e., the estimated VM finish time and the estimated energy consumed, which will be used for offer comparison by the contacted remote nodes later. It is noteworthy that the algorithm can be triggered by customized events as well depending on each node local setting. The selection of the node that will receive the task is the same as during the submission phase except that the initiator is no longer a candidate node.

3.2 Scenario

Assuming that a cloud is comprised of interconnected nodes $H_{i,j}$, i $\in [1,2,...,N]$ and j $\in [1,2,...,nH_i]$, $VM_{i,j,k}$ is submitted to node $H_{i,j}$. The Decentralized algorithm then leads to the following phases and steps for job assignment and dynamic scheduling. When a new VM arrives there is a VM submission phase.

- VM Submission Phase.
  - Step 1 : $VM_{i,j,k}$ is submitted to node $H^{req}_{i,j}$ together with its characteristic data.
  - Step 2 : The initiator node $H^{req}_{i,j}$ generates a request message according to the retrieved execution requirement from $VM_{i,j,k}$, and broadcasts such a message to the cloud by the em-

ployed lightweight decentralized algorithm information system.

- Step 3 : The initiator node $H_{i,j}^{req}$ then waits for some time to receive returned accept messages from other nodes of the data center.
- Step 4 : Nodes receiving the broadcast request message check if the required VM profile can be matched by the local resources.
- Step 5 : If yes, each candidate node computes an estimated VM response time according to its current scheduling, their loads (load increases while staying between the under-loaded threshold and the over-loaded threshold) and resource status, and sends the information back to the initiator node $H_{i,j}^{req}$ by means of an accept message.
- Step 6 : The initiator node $H_{i,j}^{req}$ evaluates received accept messages and selects the candidate node which fulfilled requirements and minimize the energy consumption. The selected node, referred to the assignee node, is assigned the VM by means of an assign message. A node $H_{i',j'}^{ass}$ is selected as the candidate node for $VM_{i,j,k}$ delegation.

Regularly the system tries to schedule the waiting VM.

- Dynamic Scheduling Phase.
  - Step 7 : The assignee node $H_{i',j'}^{ass}$ takes the assigned $VM_{i,j,k}$ and puts it into its local list of VMs.
  - Step 8 : The assignee node $H_{i',j'}^{ass}$ periodically picks jobs from its local list of VMs, which have a long enough waiting time and have not been selected recently. Afterward, for each selected VM, the assignee node $H_{i',j'}^{ass}$ generates an inform message, which contains both VM estimated completion time and energy estimated for VM execution. In our case, $VM_{i,j,k}$ is selected with an assumed long waiting time.
  - Step 9 : The inform message of $VM_{i,j,k}$ is sent over the network using a low-overhead walking protocol (the walk starts at an initial node and at each step selects randomly a minimum number of neighbors).
  - Step 10 : Nodes receiving the aforementioned inform message check if the local resource and scheduling status could match the requirement of $VM_{i,j,k}$; furthermore, they also need to check whether the estimated VM response time is short enough when compared to $VM_{i,j,k}$ current response time upon the assignee node $H_{i',j'}^{ass}$.
  - Step 11 : If the evaluation result from above step is positive, an accept message will be generated and delivered to the assignee node $H_{i',j'}^{ass}$.
  - Step 12 : The assignee node $H_{i',j'}^{ass}$ evaluates the received accept messages according to the promised VM response time. As a result, node $H_{i',j'}^{re-ass}$ is selected, and $VM_{i,j,k}$ is re-assigned by means of an assign message.
  - Step 13 : To enable tracking of VMs for the purpose of node crash tolerance, each VM reassignment is logged and notified to the initiator node $H_{i,j}^{req}$.
  - Step 14 : To enable node weighting for future scheduling, VM completion status is sent back to the original assignee node $H_{i',j'}^{ass}$ and initiator node $H_{i,j}^{req}$.

  Regularly the scheduling restarts to optimize the metrics.
- Migration Phase. Two steps :
  - Step 1 : For a node $H_{i,j}$, if $c_{i,j} > \varepsilon$ the node is considered as being overloaded (with $c_{i,j}$ representing the load of the node j in cluster i); Node $H_{i,j}$ requires migration of one or more of its VMs. The algorithm must determine what part of the current load should be sent to other nodes. The algorithm seeks, for all nodes in all sites, one or more nodes, whose charge $\gamma < c_{i',j'} < \varepsilon$, which can receive the VM to migrate. If such node is found, the execution of the VM is stopped at the source node $H_{i,j}$ and continues at the destinations node. If the decentralized algorithm is to add one or more nodes, the algorithm must determine what part of the current load should be sent to the added nodes. Obviously, the load to be migrated should come from nodes undergoing excessive demand for resources.
  - Step 2 : if $c_{i,j} < \gamma$ the node $H_{i,j}$ is considered as being underloaded; Node $H_{i,j}$ requires migration of its VMs. The algorithm seeks, for all nodes in all sites, one or more nodes whose charge $\gamma < c_{i',j'} < \varepsilon$ which can receive the load to migrate. If such nodes is found, the execution of all VMs is stopped at the source node $H_{i,j}$ and continues at the destinations node.

3.3 Algorithms

Our Energy Efficient Decentralized Scheduling for Cloud adopts the promised VM response time and the node load as main criterions to evaluate the nodes' capabilities. Participating nodes need to calculate their

actual load and estimated response time for a concerned VM and bid for the VM delegation using the calculated and promised VM response time. Each responder node computes an estimated completion time according to its current scheduling and resource status, calculates the necessary energy, and delivers the information by means of an accept message. In addition, the node load is also added to the generated message, which can be utilized by the requester node for responder node evaluation and selection. The selected node is the best candidate node based on several parameters, such as the promised time to complete, energy consumed, the node load between under-load threshold and over-load threshold, node weight due to historical interaction records, etc. Furthermore, during the execution of the VMs, all the time the system checks if there are underloaded or overloaded nodes. A self-healing is designed similar to the intelligent feedback loop, and supports not only scheduling but also re-scheduling activities. It enables Decentralized algorithm with self-healing capabilities to allow VMs that wait a long time in a node to be re-scheduled. The dynamic scheduling phase, but also the possibility to switch off or switch on nodes, and especially the cooperation between nodes help to maintain system efficiency. These properties of Decentralized algorithm constitute its self-healing system. Decentralized algorithm provides task scheduling strategy, which dynamically migrate VMs among computing nodes, transferring VMs from underloaded nodes to loaded but not overloaded nodes. It balances load of computing nodes as far as possible in order to reduce program running time. The Decentralized algorithm behaves globally as follows:

- If total VM load on the node j of site $i > \varepsilon$, node is over-loaded. Let ε be the over-load threshold.
- If total VM load on the node j of site $i < \gamma$, node is under-loaded. Let γ be the under-load threshold.

The algorithm is described in Algorithm 4. For the sake of simplicity, corner cases such as all nodes over-loaded are not included. Selection policies take into account migration cost. First the algorithm computes the current load $c_{i,j}$ (line 6) of the host. Then it calls the CheckLoad procedure (see Algorithm 3) that will either migrate all VMs (if node is underloaded, in order to switch $H_{i,j}$ off) or some VMs (if node is overloaded, so that it becomes again in normal mode). Then the algorithm checks if some VM are still waiting for execution. If so, it selects the one waiting for the longest period (see Algorithm 1) and migrate it to the first host where it can enter, hosts being sorted by their maximum power consumption P^{max}.

Algorithm 1: Find VM with longest remaining waiting time: FindVmWait.

Input: $VM_{i,j}$ // set of VM of node j of site i
Output: $VM_{i,j,k}$ // $VM_{i,j,k}$ in node j of site i

```
1:  function FINDVMWAIT(x)
2:      for VM_{i,j,k} in VM_{i,j} do
3:          D = 0
4:          if (D^W_{i,j,k} > D) then // D^W_{i,j,k} is the waiting
    time of VM k in node j of site i
5:              D= D^W_{i,j,k}
6:              k' = k
7:          end if
8:      end for
9:      Return VM_{i,j,k'}
10: end function
```

Algorithm 2: Find VM with longest remaining execution time: FindVmExec.

Input: $VM_{i,j}$ //set of VM of node j of site i
Output: $VM_{i,j,k}$ // $VM_{i,j,k}$ in node j of site i

```
1:  function FINDVMEXEC(VM_{i,j})
2:      for VM_{i,j,k} in VM_{i,j} do
3:          D = 0
4:          if (D^Exec_{i,j,k} > D) then // D^Exec_{i,j,k} is the execu-
    tion time of VM k in node j of site i
5:              D= D^Exec_{i,j,k}
6:              k' = k
7:          end if
8:      end for
9:      Return VM_{i,j,k'}
10: end function
```

The procedure CheckLoad() (described in Algorithm 3) is composed of two parts. In case of underload (line 4 to 10), it looks for potential destinations for all present VMs. When many potential destinations exist, the one with the minimum migration cost is selected. In case of overload, the algorithm repeatedly looks for the VM with maximum remaining execution time (see Algorithm 2), and migrates this VM to the first node that could host it (sorted by P^{max}), until the load is under the overload threshold ε. In both cases, if no destination is found, the VM remains where it is (and the node remains either underloaded or overloaded).

3.4 Analysis of the Algorithm

The decentralized algorithm is inspired by the motivation of enabling cloud scheduling for the scope of the overall cloud, instead of each single node. In contrast to conventional cloud scheduling solutions, this

Algorithm 3: Migration of VM based on sorted nodes.

procedure CHECKLOAD($c_{i,j}$)
Input: $H_{i,j}$ // the host running this procedure
  set $Potential = \emptyset$
  set $H = \{H_{i',j'} / i' \in [1,2,...,N], j' \in [1,2,...,nH_i], i' \neq i, j' \neq j\}$, sorted by P^{max}
  if $c_{i,j} < \gamma$ **then**
    for ($H_{i',j'}$ in H) **do**
      if $c_{i,j} + c_{i',j'} < \varepsilon$ **then**
        Add $H_{i',j'}$ to Potential
      end if
    end for
    Migrate all VMs from $H_{i,j}$ to the node in Potential with minimum migration cost
  else
    while ($c_{i,j} > \varepsilon$) **do**
      $VM_{i,j,k} = FindVMExec(VM_{i,j})$
      $l_{i,j,k} = VM_{i,j,k}$ load
      for ($H_{i',j'}$ in H) **do**
        if $c_{i',j'} + l_{i,j,k} < \varepsilon$ **then**
          Migrate $VM_{i,j,k}$ from $H_{i,j}$ to $H_{i',j'}$
        end if
      end for
    end while
  end if
end procedure

Algorithm 4: Decentralized scheduling.

Input: $H_{i,j}$ // the host running this procedure
// P^{min}, *minimum power*
// P^{max}, *maximum power*
procedure DECENTRALIZEDSCHEDULING
  for Regularly **do**
    set $H = \{H_{i',j'} / i' \in [1,2,...,N], j' \in [1,2,...,nH_i], i' \neq i, j' \neq j\}$, sorted by P^{max}
    Compute $c_{i,j}$ // it is the load of $H_{i,j}$
    CheckLoad($c_{i,j}$)
    if (notempty $VM_{i,j}^{wait}$) **then**
      $VM_{i,j,k}$=FindVMWait($VM_{i,j}^{wait}$)
      $l_{i,j,k} = VM_{i,j,k}$ load
      for $H_{i',j'}$ in H **do**
        if $(c_{i',j'} + l_{i,j,k}) < \varepsilon$ **then**
          Migrate $VM_{i,j,k}$ from $H_{i,j}$ to $H_{i',j'}$
        end if
      end for
    end if
  end for
end procedure

algorithm is designed to deliver relevant scheduling events, such as VM submission and scheduled information, to as many neighboring remote nodes as possible. Moreover, the algorithm enables the possibility of dynamic rescheduling in order to adapt to cloud data center characteristics such as instantaneity and volatility. Our decentralized algorithm also uses consolidation in order to minimize energy consumed. The algorithm is comprised of three phases, namely the VM submission phase, dynamic rescheduling phase and migration phase. The VM response time and the energy are used as critical criteria to evaluate the energy consumed, the performance of scheduling and rescheduling. The decentralized algorithm is able to work together with different kinds of local scheduling algorithms.

4 RESULTS

We run the simulation using a Grid5000 workload and measure the amount of energy consumed by the placement and the average execution times for both algorithms (i.e. centralized and Decentralized algorithm). To derive the actual energy savings, the amount of energy spent for VM placement was estimated by the formula defined below.
Nodes have two different power states : Switched on and switched off. While switched on, power consumption is linear in function of load between $P_{i,j}^{min}$ and $P_{i,j}^{max}$.

We use the classical linear model of power consumption in function of load :
$\forall i, j\ P_{i,j} = P_{i,j}^{min} + C_{i,j}(P_{i,j}^{max}\text{-}P_{i,j}^{min})$
if node j is switched on, $P_{i,j} = 0$ otherwise.
Therefore the total power consumption of the system is:
$P = \sum_{i=1}^{N} \sum_{j=1}^{H} P_{i,j}$
To obtain energy consumed during a time slice, instantaneous power is integrated over time $\int_{t1}^{t2} P(t)dt$. Total energy is then obtained by summing all the energy of those time slices.

The final simulation results show the gain in energy and makespan does not depend on the numbers of jobs but mostly of the distribution of jobs between nodes. The final simulation results are depicted in table 1. The gain in energy and makespan don't depend on the number of jobs but mostly of the distribution of jobs between nodes.

Figure 1 demonstrates the number of migrations incurred by 4500 jobs. An obvious observation is that migrations are beneficial to save energy. The second phase is the dynamic scheduling phase. If at any time the tasks are finished and there are several

Table 1: Simulation results (StDev= standard deviation).

Job	Approach	Makespan	Makespan gain	Makespan StDev	Energy	Energy gain	energy StDev
500	Cent	48770		6491	16358.5		10363419
	Dec	48984	-0.44	5574	16252.9	0.65	10475850
1000	Cent	102089		40225	34473.4		56793603
	Dec	100859	1.20	40676	31628.8	8.25	51366829
1500	Cent	117143		24348	45480.4		104549039
	Dec	130734	-11.6	61778	47263.9	-3.9	118552002
2000	Cent	153386		51215	58819.5		154963019
	Dec	143047	6.74	39176	63349.97	7.7	180703187
2500	Cent	143977		37093	56716.3		129407887
	Dec	143541	0.3	36334	56108.85	1.07	116130251
3000	Cent	250457		237465	61635.4		167952610
	Dec	211032	15.72	191168	56993.7	7.5	133578369
3500	Cent	235614		215901	63370.7		164512761
	Dec	202696	13.97	162140	55093.6	13.06	130973378
4000	Cent	221526		192048	54071.2		116554623
	Dec	233452	5.38	223754	60270.2	11.5	157261329
4500	Cent	241048		222470	57983.4		131320566
	Dec	207189	14.05	173065	59478.3	2.57	175986327

machines under loaded, they can be consolidated. This is not the case for algorithms which never calls into question the allocation once the jobs are running. The impact of migration, however, may not be large enough to dominate the total energy gain when the number of jobs is less than 300. We performed series of tests, comparing Centralized and Decentralized algorithm.

The Centralized algorithm (EACAB) (Thiam et al., 2014) works by associating a credit value with each node. The credit of a node depends on its affinity to its jobs, its current workload and its communication behavior. Energy savings are achieved by continuous consolidation of VMs according to current utilization of resources, virtual network topologies established between VMs and thermal state of computing nodes.

Generally, the Decentralized algorithm schedules slightly worse in terms of makespan than the Centralized algorithm.

Figure 1: Centralized (Cent) vs Decentralized (Dec) algorithms: Energy.

Typical results of experiments are presented in Figures 1, 2, 3 and 4. In Figure 1 it is easy to notice that energy consumed by the distributed algorithm is comparable to centralized strategy for low number of jobs. These poor results are caused by low number of migrations since majority of jobs can be executed without exceeding their due dates. This situation changes for higher loads when number of migrations is increased and the distributed algorithm outperforms the centralized in some case. We achieved similar results for centralized algorithm which use migration and anti load-balancing techniques. When a cluster load is below the under-loaded threshold, centralized and Decentralized algorithm are able to migrate jobs to more-loaded clusters and switch off under-loaded cluster. In this case, performance measures and energy depend strongly on the collaboration of less-loaded clusters. When their cooperation is too low the system as a whole starts to be inefficient, although the performance of the less-loaded clusters is not affected. Consequently, we consider that there must be some minimal cooperation that results from a cloud agreement. As in real systems the job stream changes, this minimal cooperation can be also inter-

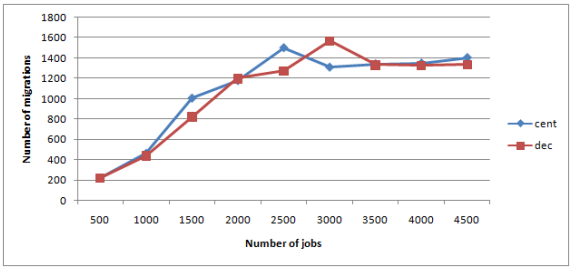

Figure 2: Centralized (Cent) vs Decentralized (Dec) algorithms: Migration.

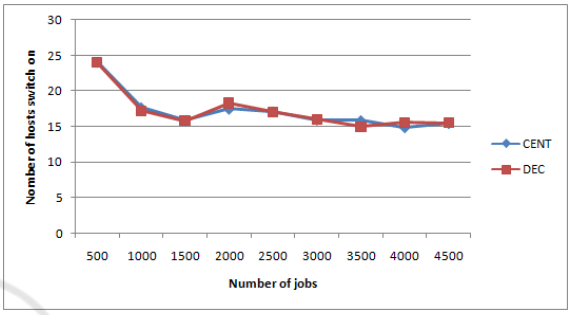

Figure 3: Centralized (Cent) vs Decentralized (Dec) algorithms: Maximum nodes switched on.

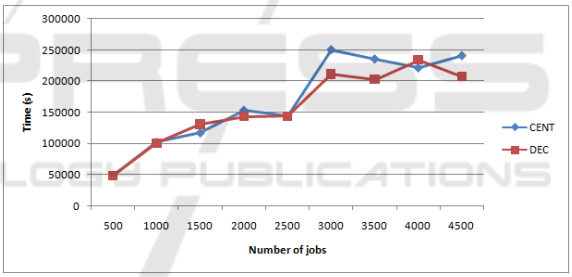

Figure 4: Centralized (Cent) vs Decentralized (Dec) algorithms: Makespan.

preted as an ”insurance” to imbalance the load. From the experiments above, we can get the obvious conclusion that both the Centralized algorithm and Decentralized algorithm can reduce energy consumed of data centers. Figure 4 above shows the execution time for all tasks and both schedulers. We can see that the two algorithms have the same behavior.

5 CONCLUSIONS

We have compared the Decentralized algorithm to Centralized algorithm EACAB over a range of realistic problem instances using simulation. The proposed algorithm can compute allocations effectively with an important energy gain. The simulation results showed that our algorithm is capable of obtaining energy-efficient schedules using less optimization time. Our experimental results have shown that: (1) the Decentralized algorithm is capable of obtaining energy-efficient schedules using less optimization time; (2) Application performance is not impacted by performing migration using under load and over load thresholds; (3) The system scales well with increasing number of resources thus making it suitable for managing large-scale virtualized data centers; (4) The anti load-balancing technique used by the two approaches achieve substantial energy savings.
Finally we can get the obvious conclusion that both Centralized and Decentralized algorithm algorithm can reduce energy consumed of data centers. Compared to centralized algorithms, decentralized algorithms have a simplicity that makes them promising in practice though for a verification more experiments are required.

REFERENCES

Brant, C., Choudhary, S., Garrison, J., and McKay, M. (2017). Distributed virtual machine image management for cloud computing. US Patent 9,747,124.

Feller, E., Rilling, L., Morin, C., Lottiaux, R., and Leprince, D. (2010). Snooze: a scalable, fault-tolerant and distributed consolidation manager for large-scale clusters. In *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, pages 125–132. IEEE Computer Society.

Hermenier, F., Lèbre, A., and Menaud, J.-M. (2010). Cluster-wide context switch of virtualized jobs. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 658–666. ACM.

Kang, B. and Choo, H. (2016). A cluster-based decentralized job dispatching for the large-scale cloud. *EURASIP Journal on Wireless Communications and Networking*, 2016(1):25.

Khosravi, A., Andrew, L. L., and Buyya, R. (2017). Dynamic vm placement method for minimizing energy and carbon cost in geographically distributed cloud data centers. *IEEE Transactions on Sustainable Computing*, 2(2):183–196.

Mehrsai, A., Figueira, G., Santos, N., Amorim, P., and Almada-Lobo, B. (2017). Decentralized vs. centralized sequencing in a complex job-shop scheduling. In *IFIP International Conference on Advances in Production Management Systems*, pages 467–474. Springer.

Pattanaik, S. (2016). *Efficient Energy Management in Cloud Data center using VM Consolidation*. PhD thesis.

Quesnel, F. and Lèbre, A. (2011). Cooperative dynamic scheduling of virtual machines in distributed systems. In *European Conference on Parallel Processing*, pages 457–466. Springer.

Rouzaud-Cornabas, J. (2010). A distributed and collaborative dynamic load balancer for virtual machine. In *Euro-Par Workshops*, pages 641–648. Springer.

Thiam, C., Da Costa, G., and Pierson, J.-M. (2014). Energy aware clouds scheduling using anti-load balancing algorithm: Eacab. In *3rd International Conference on Smart Grids and Green IT Systems (SMARTGREENS 2014)*, pages pp–82.

Tseng, H.-W., Yang, T.-T., Yang, K.-C., and Chen, P.-S. (2017). An energy efficient vm management scheme with power-law characteristic in video streaming data centers. *IEEE Transactions on Parallel and Distributed Systems*.

Yang, D., Deng, C., and Zhao, Z. (2016). Dynamic scheduling method of virtual resources based on the prediction model. In *International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 384–396. Springer.

Yazir, Y. O., Matthews, C., Farahbod, R., Neville, S., Guitouni, A., Ganti, S., and Coady, Y. (2010). Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 91–98. Ieee.

Yin, L., He, W., and Luo, J. (2016). Distributed virtual machine placement based on dependability in data centers. In *Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), 2016 12th International Conference on*, pages 2152–2158. IEEE.