Resource Allocation for Minimized Power Consumption in Hardware Accelerated Clouds

Nazım Umut Ekici*†, Klaus Werner Schmidt†, Alper Yazar*†, Ece Güran Schmidt†

† Department of Electrical and Electronics Engineering, METU, Ankara, Turkey
{nazim.ekici, schmidt, alper.yazar, eguran}@metu.edu.tr
* Defence Systems Technologies, ASELSAN, Ankara, Turkey
{uekici, ayazar}@aselsan.com.tr

***Abstract*—In this paper we propose ACCLOUD-MAN, a novel resource manager for heterogeneous cloud data centers. In heterogeneous clouds a user request can be satisfied with more than one physical resource alternative. That is, the resource manager must decide which resource alternative will be chosen, along with the decision of the server the request will be assigned to. ACCLOUD-MAN's resource management objective is to reduce the power consumption of the cloud data center. To this end, the manager is modeled as an integer linear programming problem and is implemented in MATLAB, along with a cloud data center simulation platform. Simulation results show that the proposed ACCLOUD-MAN outperforms existing resource allocation methods such as Openstack.**

***Index Terms*—cloud computing, data center, hardware accelerator, resource management, green computing.**

I. Introduction

Cloud computing systems use virtualization to serve resources on physical machines to users in a dynamic manner. In this respect, resource allocation methods aim to create a configuration that will meet requests of the users according to different performance goals and constraints with the existing physical resources. Heterogeneous cloud architectures having computational resources such as GPU (Graphics Processing Unit), TPU (Tensor Processing Unit) as well as FPGA based accelerators in addition to classical resources like CPU draw attention from both academia and industry.

In heterogeneous clouds, resource allocation methods need to cover new computation resources in addition to standard existing resources such as CPU, memory, disk, bandwidth (BW). Furthermore, a cloud resource manager assigns resources for SaaS (Software as a Service) requests, depending on the type of software the user requests. In heterogeneous clouds a SaaS request can be satisfied by using standard CPUs or alternatively by using other computing resources introduced by the heterogeneous clouds. Each of these alternatives have different performance constraints and resource usage costs. Any resource allocation method should keep up with the demand arrival rate without compromising an efficient solution and adding too much latency.

In this paper, we propose and evaluate ACCLOUD-MAN (ACcelerated CLOUD MANagement) as a novel resource manager including new computing resources introduced by heterogeneous clouds. ACCLOUD-MAN simultaneously considers alternative ways requests can be satisfied and allocates resources optimally, by deciding on which type of computing resource will be used and on which server it will run. In the current development phase, ACCLOUD-MAN incorporates FPGA-based hardware accelerators as a new type of computing resource and performs resource allocation to minimize power consumption within the scope of green computing.

This work was supported by the Scientific and Research Council of Turkey (TUBITAK) [Project Code 117E667-117E668].

II. Cloud Computing Resource Categorization and Previous Works

Cloud resources are served mainly as 3 different types of services: Infrastructure (Infrastructure as a Service, IaaS), Platform (Platform as a Service, PaaS) and Software (Software as a Service, SaaS) [1]. IaaS users get dedicated virtual machines (VMs) with allocated resources like CPU, memory, disk, network and a running OS. For PaaS users, a cloud provider serves a managed software development or deployment environment. Both IaaS and PaaS users explicitly state the amount and types of resources they request. SaaS users are not interested in software development. They just provide their data, which is processed by software developed by the cloud provider. SaaS users do not explicitly state the amount and type of resources they require. The configuration satisfying their request is decided by the cloud provider depending on the type of software and size of submitted data.

Resource management studies in the literature aim to assign a given request to different configurations in the best possible way. [2] tries to minimize either energy consumption or response time. In [3], a resource allocation method for a cloud with different types of servers is based on fairness. Resource management is an NP-hard problem [4], [5]. [6] solves the multi-purpose optimization problem that is formulated to predict and improve energy and resource usage in a meta-heuristic manner using genetic algorithms. [7] proposes a prediction-based and power-aware allocation algorithm that includes energy consumption of network devices in the cloud data centers.

To the best of our knowledge, there is no study covering new resource types in heterogeneous clouds, as well as a resource allocation method for SaaS, which evaluates alternative resource allocation schemes to meet user requests in pure traditional computing resources or in different heterogeneous configurations, including hardware accelerators.

III. Formulation of ACCLOUD-MAN

This section formulates the proposed cloud computing resource management framework ACCLOUD-MAN. Section III-A explains the general setting and the objectives of the ACCLOUD-MAN resource manager. Section III-B formulates an integer linear program (ILP) in order to realize the ACCLOUD-MAN resource manager.

A. Cloud Computing Resource Management

Cloud data centers (CDC) offer certain types of resources to the user. In this work, we consider that 5 different resource types are offered by physical machines (PM) in a CDC: CPU, FPGA, Memory, Disk and network Bandwidth. However, the proposed method is applicable for any type and number of resources. In analogy to our previous work [8], we assume that FPGAs are virtualized by partitioning them into smaller reconfigurable regions. These regions are then served to the user as standalone computing resources or as hardware accelerators with traditional CPUs.

Users can request resources from the CDC in two ways. For IaaS/PaaS (IPaaS) requests in accordance with the service models in Section II, the user explicitly specifies the amount of requested resources for each resource type. For SaaS request, the user only specifies the requested cloud application. The amount of resources required for such SaaS request is then determined by ACCLOUD-MAN, taking into account that there may be more than one resource alternative to meet a SaaS request. For example, one (or more) processor cores or one (or more) FPGA regions can be assigned to run an application to compress a 1 GB file.

The ACCLOUD-MAN resource manager has two competing objectives. On the one hand, it is desired to minimize power consumption of the data center for the requested resources. On the other hand, is is required to make the resource allocation decision at a speed close to real time so as to avoid interference with the operation of the cloud. Hereby, it has to be considered that the power consumption increases when new requests are assigned to currently running nodes. If running servers do not have sufficient resources for new requests, then offline servers must be powered up. When a server is powered up, it consumes constant power even if it isn't used for computation.

Addressing the stated objectives is not a trivial task. For example, the goal of low power consumption is generally not achieved by selecting the PM with the least need for power. If the PM with the lowest power consumption is off, assigning a more power intensive alternative to a PM that is already open may result in less total power consumption. For this reason, the resource manager should decide not only which PM will be assigned to each incoming request, but also which alternative will be used. Hereby, ACCLOUD-MAN selects an assignment among previously known resource alternatives for SaaS requests. The difference observed by the user among the alternatives is the time of completion of the work. It is assumed that a preliminary study and profiles of different types of inputs are available for SaaS software running in the cloud and the alternatives are kept in a database by the service provider according to these profiles.

B. Resource Allocation Model

This section formulates the stated ACCLOUD-MAN functionality in the form of an ILP. Hereby, a set $\mathbb{PM} = \{PM_1, \ldots, PM_N\}$ of available PMs is assumed, whereby each PM $PM_i \in \mathbb{PM}$ offers CPU, FPGA, Memory, Disk, and network Bandwidth resources defined by C_i, F_i, M_i, D_i, B_i, respectively. CPU and FPGA resources are measured and offered to the user in terms of number of cores and reconfigurable regions (module) respectively. Memory, disk are in terms of GB and bandwidth is in terms of Mbps.

The main task of ACCLOUD-MAN is to assign user requests to the PMs in $\mathbb{PM}$. Since user requests arrive sporadically, ACCLOUD-MAN collects a set of K pending requests $\mathbb{REQ} = \{\mathbb{REQ}_1, \ldots, \mathbb{REQ}_K\}$ within a pre-defined time interval and then determines a suitable assignment of these requests to PMs at the end of this time interval. We denote the time instant when a decision is made as a decision instant of ACCLOUD-MAN. This time interval should be chosen such that it would accumulate enough user requests but should not keep users waiting too much for their request to be evaluated. Hereby, it is assumed that each request $\mathbb{REQ}_j$ can be served as one of n_j alternatives. That is, $\mathbb{REQ}_j = \{\mathrm{req}_{j,1}, \ldots, \mathrm{req}_{j,n_j}\}$ is a set of alternatives $\mathrm{req}_{j,k}$ and each alternative $\mathrm{req}_{j,k}$ requires the physical resources $c_{j,k}$ (CPU), $f_{j,k}$ (FPGA), $m_{j,k}$ (Memory), $d_{j,k}$ (Disk), $b_{j,k}$ (Bandwidth).

In order to formulate the desired minimization of the power consumption, we introduce a model of the power consumption of a CDC. Hereby, we take into account that the CDC is in a well-defined state at each decision instant of ACCLOUD-MAN. That is, a certain number of PMs is on, whereas the remaining PMs are off. We introduce the parameter on_i such that $\mathrm{on}_i = 1$ if PM_i is on and $\mathrm{on}_i = 0$ if PM_i is off right before the decision instant. In addition, we introduce a decision variable q_i for each PM_i such that $q_i = 1$ if PM_i is on and $q_i = 0$ if PM_i is off after the decision instant. We emphasize that $\mathrm{on_i}$ is a given parameter, whereas q_i is a decision variable to be computed by ACCLOUD-MAN.

The power consumed by each PM depends on the state of the PM (on/off) and the used resources of the requests served by the PM. Accordingly, we introduce $P_{\mathrm{on},i}$ as the idle power consumption of PM_i. In addition, $P_{\mathrm{CPU},i}$ and $P_{\mathrm{FPGA},i}$ indicate the power consumption of one CPU core and one FPGA module of PM_i, respectively.

Using the defined parameters and variables, it is now possible to determine the additional power consumption when allocating the pending requests in $\mathbb{REQ}$ to PMs in $\mathbb{PM}$. The additional power consumption consists of two components.

First, there is the power P_{newPM} of PMs that are newly switched on. It evaluates to

$$P_{\text{newPM}} = \sum_{i=1}^{N} P_{\text{on},i} \cdot (1 - \text{on}_i) \cdot q_i. \tag{1}$$

In (1), $(1 - \text{on}_i) \cdot q_i$ evaluates to 1 if PM_i was off before the decision instant $(1 - \text{on}_i = 1)$ and is on after the decision instant $(q_i = 1)$. Second, there is the power P_{comp}, which is the power used by the newly assigned CPU cores and FPGA modules. In order to compute this power, we introduce the decision variable $s_{i,j,k}$, which is 1 if alternative $\text{req}_{j,k}$ of request $\mathbb{REQ}_j$ is allocated to PM_i and 0 otherwise. Then, we obtain

$$P_{\text{comp}} = \sum_{ijn} (P_{\text{CPU},i} c_{j,k} + P_{\text{FPGA},i} f_{j,k}) \cdot s_{i,j,k} \tag{2}$$

Together, we want to minimize the power consumption, which amounts to

$$\min \quad F = P_{\text{newPM}} + P_{\text{comp}}. \tag{3}$$

While minimizing F in (3), various constraints on the resources have to be respected. First, it must hold that each PM has enough resources for the requests allocated to it. (4) to (8) represent these constraints for the different resource types: CPU, FPGA, Memory, Disk and Bandwidth.

$$\sum_{j=1}^{K} \sum_{k=1}^{n_j} c_{j,k} s_{i,j,k} \le C_i \quad \forall i \in PM \tag{4}$$

$$\sum_{j=1}^{K} \sum_{k=1}^{n_j} f_{j,k} s_{i,j,k} \le F_i \quad \forall i \in PM \tag{5}$$

$$\sum_{j=1}^{K} \sum_{k=1}^{n_j} m_{j,k} s_{i,j,k} \le M_i \quad \forall i \in PM \tag{6}$$

$$\sum_{j=1}^{K} \sum_{k=1}^{n_j} d_{j,k} s_{i,j,k} \le D_i \quad \forall i \in PM \tag{7}$$

$$\sum_{j=1}^{K} \sum_{k=1}^{n_j} b_{j,k} s_{i,j,k} \le B_i \quad \forall i \in PM \tag{8}$$

In addition, it must hold that each alternative $\text{req}_{j,k}$ of a given request $\mathbb{REQ}_j$ is assigned to a unique PM. That is,

$$\sum_{i=1}^{N} \sum_{k=1}^{n_j} s_{i,j,k} = 1 \quad \forall \mathbb{REQ}_j \in \mathbb{REQ}. \tag{9}$$

Finally, it must be the case that any PM that serves at least one request is on. This is represented by

$$\sum_{j=1}^{K} \sum_{k=1}^{n_j} s_{i,j,k} \le q_i \cdot M_1 \quad \forall PM_i \in \mathbb{PM}. \tag{10}$$

Here, M_1 is a large integer value that is larger than the maximum number of requests that can be handled by a single PM.

Altogether, the resource allocation problem of ACCLOUD-MAN is given by minimizing F in (3) subject to the constraints in (4) to (10). It is readily observed that this optimization problem is an ILP. According to the described operation of ACCLOUD-MAN, this optimization problem has to be solved at each decision instant. The result is the information about all PMs that need to be on $(q_i = 1)$, the selection of resource alternatives $\text{req}_{j,k}$ and their assignment to a PM PM_i if $s_{i,j,k} = 1$.

We denote the PM and alternative selected for a given request $\mathbb{REQ}_j$ as pm_j and alt_j, respectively. In order to determine pm_j and alt_j, the following equations can be used.

$$\text{pm}_j = \sum_{i=1}^{N} \sum_{k=1}^{n_j} (s_{i,j,n} \cdot i) \quad \forall \mathbb{REQ}_j \in \mathbb{REQ}, \tag{11}$$

$$\text{alt}_j = \sum_{i}^{N} \sum_{k=1}^{n_j} (s_{i,j,n} \cdot k) \quad \forall \mathbb{REQ}_j \in \mathbb{REQ} \tag{12}$$

Remark 1: It has to be noted that the defined optimization problem is suitable for all different types of services. In the case of IPaaS requests, the physical resource requirement is given by a single alternative. For SaaS requests, alternatives are determined according to the above-mentioned alternative database.

Remark 2: It also has to be emphasized that the formulated optimization problem is general in the sense that new/different resources can be added in a straightforward way. If a new resource contributes to the power consumption, then an additional term needs to be included in (2). Moreover, an additional resource constraint similar to (4) to (8) is needed.

C. Complexity and Simplification

In the above formulation, the cardinalities of the sets $\mathbb{PM}$ and $\mathbb{REQ}$ is N and K, respectively. If there is only one alternative per request, then the complexity of the resource allocation problem is $\mathcal{O}(N^K)$. Otherwise, the complexity is $\mathcal{O}((NL)^K)$ if the average number of alternatives per request is L.

One way of reducing this computational complexity is to divide the set of pending requests into smaller groups and processing these groups sequentially. Assuming that requests are divided into groups of M requests, there are $K' = \lceil K/M \rceil$ groups to be processed. We obtain $(NL)^M$ combinations for each group and $K' \cdot (NL)^M$ combinations in total. In this way, the new computational complexity will be $\mathcal{O}(K \cdot (NL)^M)$. Because of the smaller exponent, better scalability is expected. However, it is also expected that the decisions taken with smaller groups of request will deviate from the optimal solution.

Another way of reducing the computational complexity is to reduce the number of PMs that are included in the optimization problem. For example, assume that a number of K request has to be allocated and a number of N_{on} PMs is currently on. Then, it is possible to include a reduced number of $N_{\text{on}} + \gamma \cdot K$ PMs (instead of N PMs) in the optimization problem, leading to a reduced computational complexity of $\mathcal{O}(((N_{\text{on}}+\gamma\cdot K)\cdot L)^M)$. Hereby, γ is a coefficient that ensures solvability of the ILP. Again, it is expected that the decisions taken with a smaller number of PMs will slightly deviate from the optimal solution. This idea is further explored in the next section.

IV. ACCLOUD-MAN Evaluation

This section presents simulation results of a CDC that implements the proposed ACCLOUD-MAN and a comparison to the existing Openstack resource allocation.

A. Simulation Environment

ACCLOUD-MAN is implemented in MATLAB, together with a simulation environment as shown in Figure 1. The "Simulation Controller" simulates a CDC with physical and software resource information from the "Cloud Assets" database. It sends request events from a "Request Traces" file to ACCLOUD-MAN via the "Controller Interface" and generates resource assignments according to incoming decisions. The "Simulation Settings" block adjusts the "Simulation Controller" on how to create groups of requests or smaller sets of PMs. The "Log" block collects the data of all simulations and compares and reports the effectiveness of resource allocation strategies. The "Control Interface" communicates with the "Simulation Controller" via a TCP Socket and provides an interface to the "Allocation Strategist". The "Allocation Strategist" block groups incoming requests and sends them to the "ILP Solver" block. The "ILP Solver" block solves the ILP defined in Section III-B and transmits the resulting resource allocation decisions for each request to the "Allocation Strategist". The "Allocation Strategist" collects the decisions given for all request groups and sends them to the "Controller Interface" to be sent to the outside world. The current version of the simulation environment is designed to also be able to communicate with the OpenStack [9] software that is used in real-world applications for cloud resource allocation.

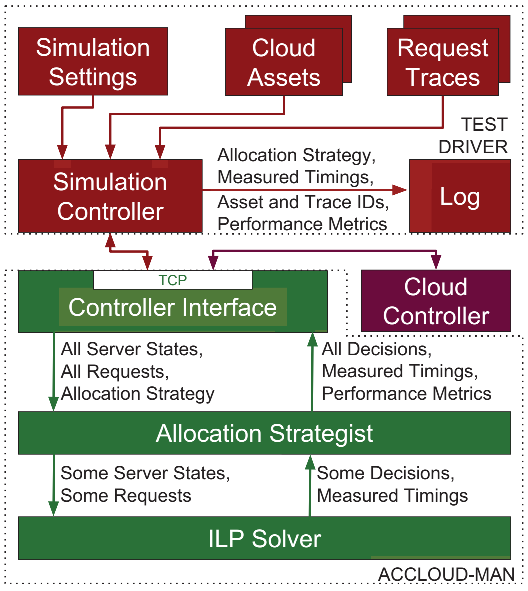
B. Simulation Setup

In order to evaluate the performance of ACCLOUD-MAN, we developed a simulation setup within the described environment. The task of the simulation setup is to

1) generate a CDC with a certain number of PMs with different resources and power consumptions,
2) dynamically generate user requests for the CDC with different resource types,
3) determine a resource allocation for pending requests,
4) update the state (resource utilization and power consumption) of the CDC.

We next describe the realization of the stated task in detail.

Regarding 1), we define intervals for the possible values of each resource type similar to the dataset in [10]. For each PM, the values for Memory (M_i), Disk (D_i) and Bandwidth (B_i) are randomly selected from an normalized interval $[80, 100]$, whereby 100 represents the maximum possible resource. Likewise, the number of CPU cores (C_i) and the number of FPGAs (F_i) is randomly chosen from the intervals $[16, 32]$ and $[4, 8]$,



Simulation Settings
Cloud Assets
Request Traces
TEST DRIVER
Simulation Controller
Allocation Strategy, Measured Timings, Asset and Trace IDs, Performance Metrics
Log
TCP
Controller Interface
Cloud Controller
All Server States, All Requests, Allocation Strategy
All Decisions, Measured Timings, Performance Metrics
Allocation Strategist
Some Server States, Some Requests
Some Decisions, Measured Timings
ILP Solver
ACCLOUD-MAN

Fig. 1. Software architecture of the resource manager.

respectively. Regarding the power consumption, a potentially high power in the interval $[900, 1100]$ is needed to turn on a PM ($P_{\text{on},i}$), whereas the power consumption for each CPU ($P_{\text{CPU},i}$) and FPGA ($P_{\text{FPGA},i}$) are in the intervals $[15, 25]$ and $[10, 20]$, respectively. In a CDC with N PMs, each PM is randomly generated with values from the stated intervals.

Regarding 2), alternatives are generated for a number of K requests in a given time interval T. The number of alternatives for a request is chosen randomly from the interval [1,3]. In agreement with Section III-B, the resource requirement (CPU, FPGA, Memory, Disk, Bandwidth) of each request alternative $\text{req}_{j,k}$ has to be specified. To this end, we proceed in a similar way as in item 1). The CPU requirement ($c_{j,k}$) and the FPGA requirement ($f_{j,k}$) are selected randomly from the intervals $[0, 16]$ and $[0, 4]$. The remaining resources (Memory – $m_{j,k}$, Disk – $d_{j,k}$, Bandwidth – $b_{j,k}$) are selected from the normalized interval $[0, 20]$. Hereby, it is respected that request alternatives might not need a certain type of resource.

Regarding 3), we include different options in our simulation environment for comparison. The first option corresponds to a method that can be implemented with Openstack's filter and weight based resource management framework, which takes individual requests and assigns them to the most power-efficient available PM. Since Openstack does not consider the case of different alternatives for the same request, the alternative with the lowest power consumption is used. The second option is a modification of the Openstack resource management that was developed in the scope of our work. In order to handle different request alternatives of a request $\mathbb{REQ}_j$, we perform the Openstack resource allocation for each alternative $\text{req}_{j,k}$ and select the most power-efficient assignment. However, requests are still processed individually. As the third option, we implemented the proposed ILP formulation in Section III-B for a certain number K of requests and a number $\gamma \cdot K$ of additional PMs that can be used for the resource allocation. For the solution of the ILP, the CPLEX solver is integrated in the simulation environment in Matlab using Tomlab [11].

Regarding 4), we perform a simulation of the resource utilization of a CDC in two main stages. The first stage simulates a cold start and the second one simulates a steady state of the CDC. In the first stage, we initialize the CDC such that all PMs are turned off. Then, we periodically generate a number of K requests and assign them to PMs according to one of the methods in item 3). The first stage continues until a certain state of the CDC is reached where most of the open PMs are almost fully utilized for at least one resource. The CDC state that is reached after the first stage can be considered as the state of an operating CDC with a high utilization of PMs. In the second stage, new requests are generated and currently served (active) requests are terminated in the CDC. K new requests are generated in a time interval T in the same way as in the first stage. Active requests for termination are randomly selected such that an average number of K requests is removed from the CDC within the time interval T. The second stage is run for a pre-defined amount of time in each experiment. In all our experiments, we assume that the rate of incoming requests is in the order of 1 request per 2.5 seconds similar to [10].

In the sequel, we report on the results of various simulation experiments that illustrate the benefits of the proposed ACCLOUD-MAN.

C. *Exp. 1: Comparison of Resource Allocation Methods*

In this section, we use the simulation setup described in Section IV-B to validate the functionality of ACCLOUD-MAN (based on the ILP solution) in comparison to the resource allocation of Openstack and the modified Openstack. To this end, we run experiments with $N = 400$ PMs, a number of $K = 10$ request that are served together and $\gamma = 2$. That is, the ILP is formulated under the assumption that $\gamma \cdot K = 20$ candidate PMs can be added to the currently on PMs. In our experiments, the first stage of the simulation is completed after about 45 min and the second stage covers about 3 hours.

Exemplary results are shown in Fig. 2 to 4. It is readily observed from Fig. 2 and 3 that ACCLOUD-MAN achieves the smallest number of used PMs and the least amount of consumed power. The improvement compared to the Openstack and modified Openstack (with alternatives) resource allocation is in the order of 10% and 5%, respectively. It can further be seen from Fig. 4 that the resource allocation computations can be performed in a small amount of time even when solving the ILP for ACCLOUD-MAN.

In addition, Fig. 5 shows the utilization of the CDC for the different resources and resource allocation methods. The fact that ACCLOUD-MAN uses the smallest number of PMs, is consistent with the observation that the resource utilization of ACCLOUD-MAN is the highest among the different methods. It is further interesting to note that all the methods achieve a high utilization of one dominating resource. In this experiment,

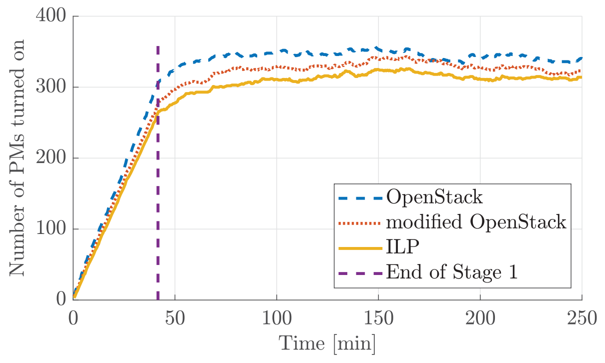


Fig. 2. Experiment 1: Number of PMs that are turned on over time.

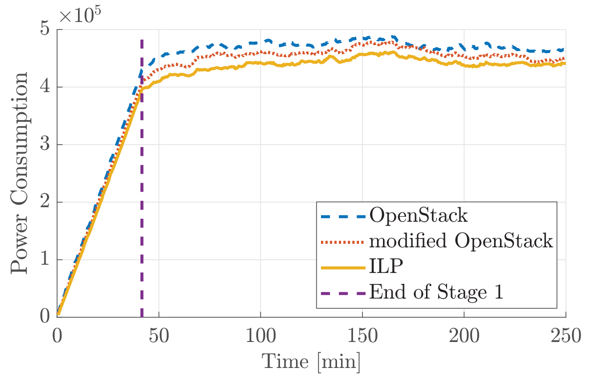


Fig. 3. Experiment 1: Power consumption over time.

this observation is meaningful since, on average, an equal number of requests with a similar distribution of resources enter and leave the CDC.

D. *Exp. 2: Dependency on the Number of Requests*

One parameter that affects the computation time and performance of ACCLOUD-MAN is the number of requests K in one group. On the one hand, a larger value of K increases the number of decision variables of the ILP in Section III-B and hence leads to an increased computation time. On the other hand, a smaller value of K is expected to have a negative effect

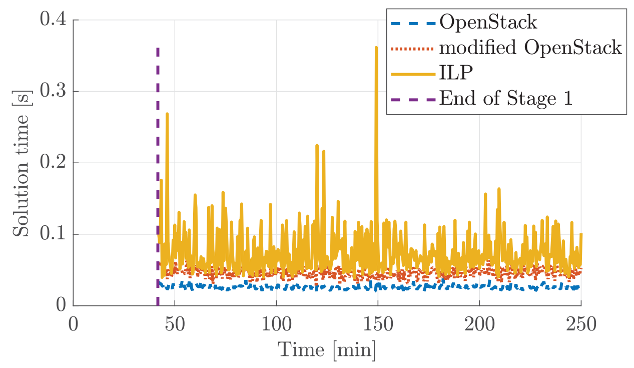


Fig. 4. Experiment 1: Run-time of the resource allocation computation.

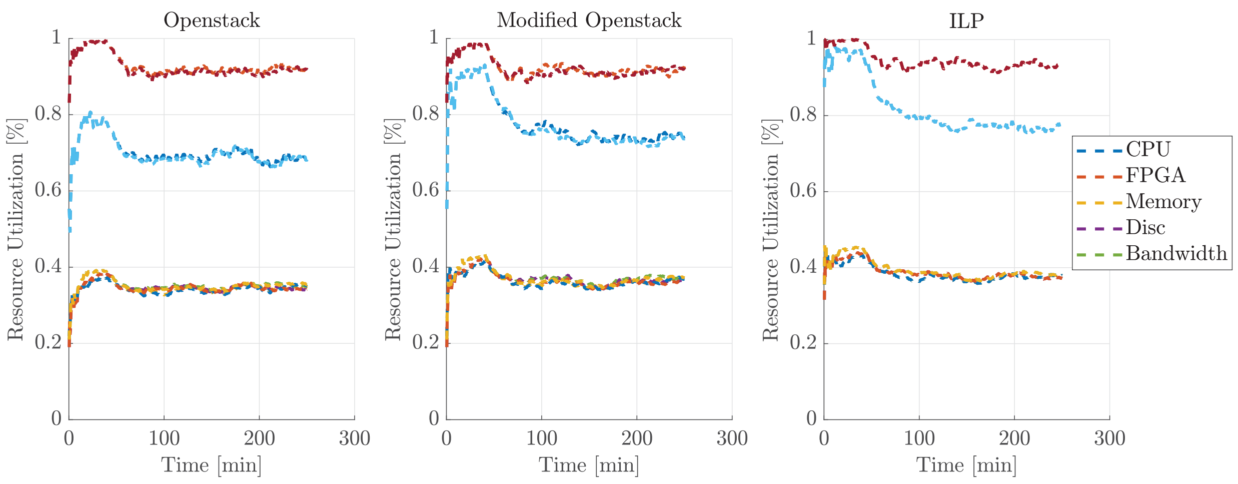


Fig. 5. Experiment 1: Resource utilization of the different methods.

on the performance (number of PMs, power consumption, resource utilization) of ACCLOUD-MAN. In order investigate this effect, we perform the same simulation experiments as in Section IV-C with different values of $K = 5$, $K = 7$ and $K = 10$. Exemplary results are shown in Fig. 6 to 8.

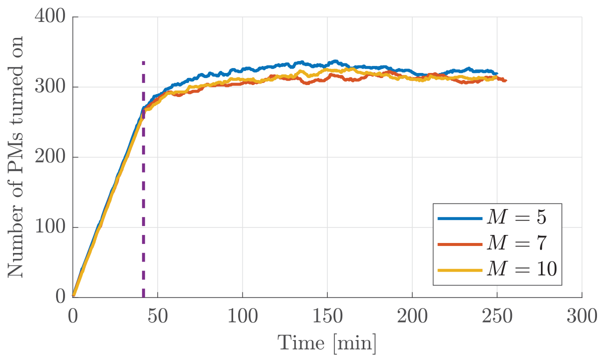


Fig. 6. Experiment 2: Number of PMs that are turned on over time.

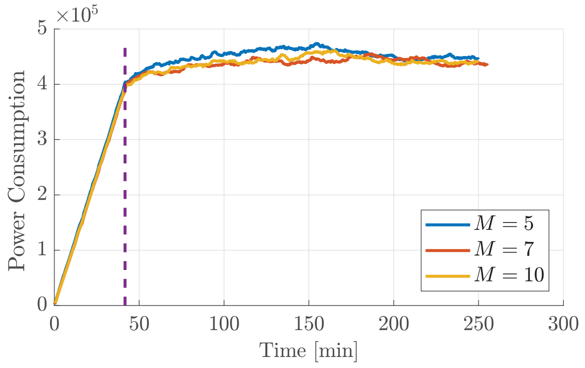


Fig. 7. Experiment 2: Power consumption over time.

The interesting observation from Fig. 6 and 7 is that the performance for $K = 7$ is slightly better than that for $K = 10$.

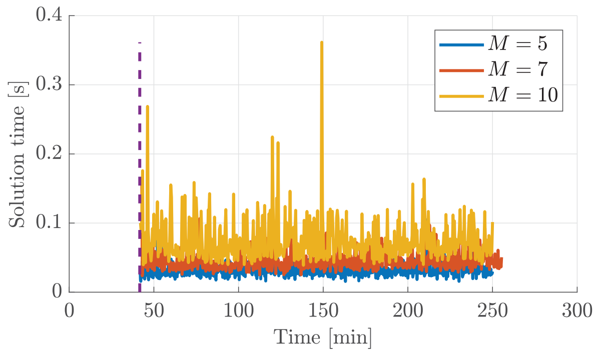


Fig. 8. Experiment 2: Run-time of the resource allocation computation.

This indicates that it is actually not necessary to include a large number of requests in one group in order to achieve a good performance of ACCLOUD-MAN.

The solver's computation time for different grouping sizes is shown in Fig. 8. The computation time strongly depends on the grouping size. For sizes of 5 and 7, it takes less than 0.1 s to compute an allocation. For a grouping size of 10, the computation takes less than 0.2s for all but 5 cases. None of these 5 exceptional cases exceed 0.4s. Comparing to the time it takes to boot up a typical VM, the worst case of 0.4s is small enough not to hinder the cloud's operation.

E. Exp. 3: Dependency on the Size of the CDC

Another interesting parameter is the number of available PMs in a CDC compared to the number of currently used PMs. We next perform an experiment where about 300 PMs are in use, whereas the number of PMs in the CDC is $N = 400$, $N = 1000$ and $N = 4000$. Regarding the other parameters, $K = 10$ and $\gamma = 2$ is used in this experiment. The simulation results are shown in Fig. 9 to 11.

It can be seen from Fig. 9 that N has a small effect on the number of PMs turned on. However, a considerable effect on the power consumption is observed from Fig. 10. This result

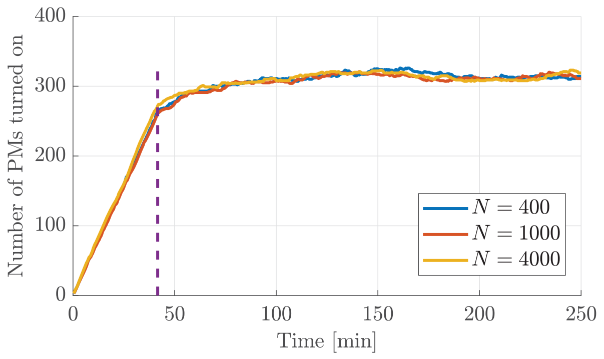


Fig. 9. Experiment 3: Number of PMs that are turned on over time.

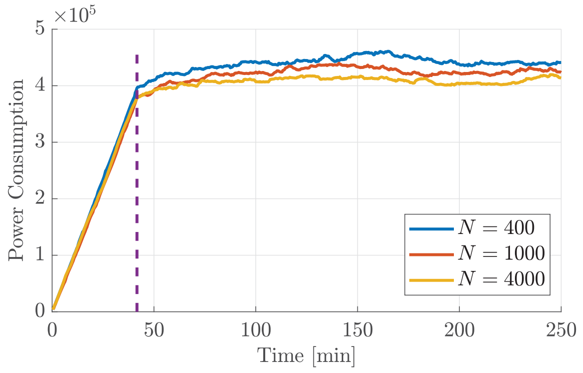


Fig. 10. Experiment 3: Power consumption over time.

can be explained as follows. In case of a small number of available PMs ($N = 400$), the PMs to be chosen for resource allocation might not fit perfectly for the incoming requests. In case of a large number of available PMs ($N = 1000$ and $N = 4000$), it is more likely to find PMs that fit the incoming requests. That is, although a similar number of PMs is needed, the PMs are more suitable for the incoming request if N is larger, leading to a lower power consumption. There is no notable difference in the computation times depending on N as can be seen from Fig. 11.

Table I shows the average performance parameters for ACCLOUD-MAN and the modified Openstack resource allocation. All the values are normalized with respect to the value of the ILP solution for $N = 4000$. The values in the table confirm that a larger number of PMs offers more options for the resource allocation and hence leads to a better performance for both the modified Openstack and ACCLOUD-MAN. Hereby, it is interesting to note that the run-time of the modified Openstack method increases considerably with the larger number of PMs.

TABLE I
COMPARISON FOR DIFFERENT NUMBERS OF PMS.

	on PM		**Power**		**Time**	
	mod. OS	ILP	mod. OS	ILP	mod. OS	ILP
$N = 400$	1.05	1.0	1.12	1.08	0.65	1.06
$N = 1000$	1.04	0.99	1.06	1.03	3.14	1.03
$N = 4000$	1.05	1.0	1.03	1.0	15.84	1.0

F. Experiment 4: Dependency on the Number of Candidate PMs

We finally evaluate the dependency of ACCLOUD-MAN performance on the parameter γ that determines the number of additional PMs included in the ILP. We perform simulations for $N = 400$, $K = 10$ and $\gamma = 1, 1.5, 2, 2, 2.5$. The simulation results are shown in Fig. 12 to 14.

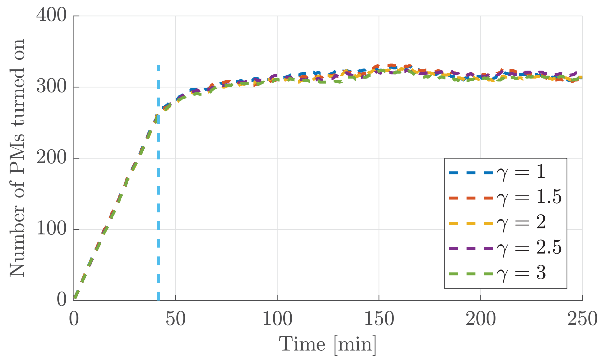


Fig. 12. Experiment 4: Number of PMs that are turned on over time.

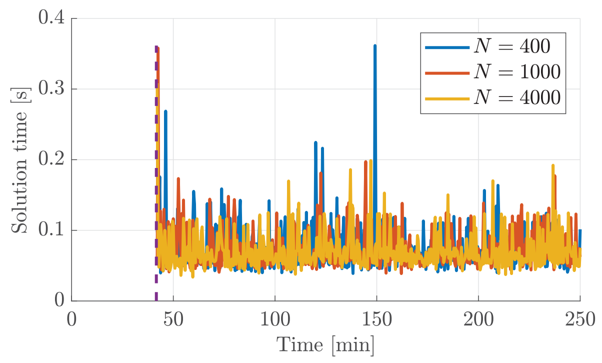


Fig. 11. Experiment 3: Run-time of the resource allocation computation.

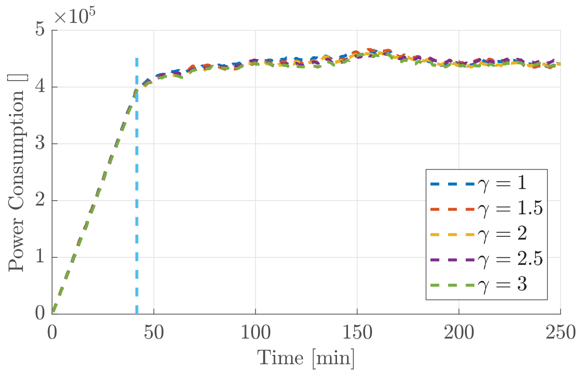


Fig. 13. Experiment 4: Power consumption over time.

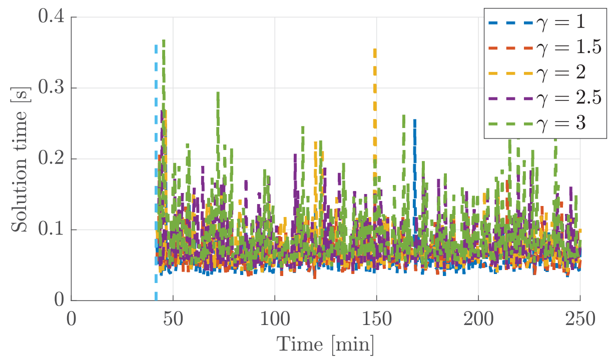


Fig. 14. Experiment 4: Run-time of the resource allocation computation.

TABLE II
COMPARISON FOR DIFFERENT VALUES OF γ.

	$\gamma = 1$	$\gamma = 1.5$	$\gamma = 2$	$\gamma = 2.5$	$\gamma = 3$
on PM	1.02	1.02	1.01	1.01	1.0
Power	1.01	1.01	1.0	1.01	1.0
Time	0.64	0.73	0.81	0.90	1.0

The interesting observation in this case is that the difference between the choices of γ is small. That is, offering a small number of additional power efficient PMs for solving the formulated ILP is sufficient to achieve a good performance of ACCLOUD-MAN. This result is confirmed by the values in Table II. Note that the values are normalized with respect to the last column.

V. CONCLUSIONS

Cloud data centers use the concept of virtualization in order to provide resources on physical machines to users. In order to meet diverse user requests, available resources need to be allocated efficiently. This paper proposes a new cloud computing resource allocation model for heterogeneous cloud architectures with different computational resources and a corresponding resource manager ACCLOUD-MAN (ACcelerated CLOUD MANagement). The resource allocation model includes IaaS and PaaS request as well as SaaS business requests that can be met with multiple physical resource alternatives. The objective of the ACCLOUD-MAN resource manager is to assign the IaaS/PaaS/SaaS request to physical machines (servers) with a minimum power consumption. The resource allocation problem is formulated as an integer linear programming (ILP) problem and solved using the CPLEX solver.

In order to evaluate the performance of ACCLOUD-MAN, the paper develops a simulation environment and performs several simulation experiments. As the main result, it is observed that ACCLOUD-MAN outperforms existing resource allocation methods such as Openstack. The experiments further show that a good performance of ACCLOUD-MAN can be achieved for reduced versions of the formulated ILP that can be solved in less than 1 second. In future work, ACCLOUD-MAN will be implemented and tested in a laboratory scale cloud data center.

ACKNOWLEDGMENTS

The authors would like to thank TUBITAK and ASELSAN for their support.

REFERENCES

[1] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in *2009 Fifth International Joint Conference on INC, IMS and IDC*, Aug 2009, pp. 44–51.

[2] J. Cao, K. Li, and I. Stojmenovic, "Optimal power allocation and load distribution for multiple heterogeneous multicore server processors across clouds and data centers," *IEEE Transactions on Computers*, vol. 63, no. 1, pp. 45–58, Jan 2014.

[3] W. Wang, B. Liang, and B. Li, "Multi-resource fair allocation in heterogeneous cloud computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 10, pp. 2822–2835, Oct 2015.

[4] A. Yousafzai, A. Gani, R. M. Noor, M. Sookhak, H. Talebian, M. Shiraz, and M. K. Khan, "Cloud resource allocation schemes: review, taxonomy, and opportunities," *Knowledge and Information Systems*, vol. 50, no. 2, pp. 347–381, Feb 2017.

[5] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J. Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A cloud-scale acceleration architecture," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016.

[6] F. Tseng, X. Wang, L. Chou, H. Chao, and V. C. M. Leung, "Dynamic resource prediction and allocation for cloud data center using the multiobjective genetic algorithm," *IEEE Systems Journal*, vol. 12, no. 2, pp. 1688–1699, June 2018.

[7] M. Tarahomi and M. Izadi, "A prediction-based and power-aware virtual machine allocation algorithm in three-tier cloud data centers," *International Journal of Communication Systems*, vol. 32, no. 3, p. e3870, 2019, e3870 IJCS-18-0389.R1. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.3870

[8] A. Yazar, A. Erol, and E. G. Schmidt, "Accloud (accelerated cloud): A novel fpga-accelerated cloud archictecture," in *2018 26th Signal Processing and Communications Applications Conference (SIU)*. IEEE, 2018, pp. 1–4.

[9] "Open Source Software For Creating Private and Public Clouds," https://www.openstack.org/.

[10] "Microsoft Azure What is Azure," https://azure.microsoft.com/en-us/overview/.

[11] "TOMLAB Optimization," https://tomopt.com/tomlab/.