# Dynamic SLA Management with Forecasting using Multi-Objective Optimization

Alexandru-Florian Antonescu*†, Philip Robinson*, Torsten Braun†
*SAP Research
SAP (Switzerland) Inc., Althardstrasse 80, 8105 Regensdorf, Switzerland
SAP (UK) Limited, The Concourse, Queen's Road, Queen's Island, Titanic Quarter, Belfast BT3 9DT, UK
alexandru-florian.antonescu@sap.com, philip.robinson@sap.com
†Communication and Distributed Systems (CDS) Research Group
University of Bern, Neubrueckstrasse 10, 3012 Bern, Switzerland
braun@iam.unibe.ch

***Abstract*—Cost-efficient operation while satisfying performance and availability guarantees in Service Level Agreements (SLAs) is a challenge for Cloud Computing, as these are potentially conflicting objectives. We present a framework for SLA management based on multi-objective optimization. The framework features a forecasting model for determining the best virtual machine-to-host allocation given the need to minimize SLA violations, energy consumption and resource wasting. A comprehensive SLA management solution is proposed that uses event processing for monitoring and enables dynamic provisioning of virtual machines onto the physical infrastructure. We validated our implementation against serveral standard heuristics and were able to show that our approach is significantly better.**

## I. Introduction

The efficient management of Service Level Agreements (SLA) is of particular importance for Cloud Computing, where Virtual Machines (VMs) are allocated to hosts in a shared physical infrastructure [1], [2]. However, a multi-objective optimization problem for resource allocation arises for physical infrastructure providers, where the ability to deliver advertised levels of performance and capacity availability need to be maximized, while minimizing energy consumption and resource wastage.

Leading cloud service providers [3], [4] (of infrastructure or application services) use SLA management for specifying and maintaining the quality of service (QoS) and availability levels to their customers. An important phase of this process is allocation of resources including initial and runtime placement optimization.

Dealing with exclusively-owned virtual machine (VM) instances deployed on a shared physical infrastructure presents a greater challenge, given the multi-objective optimization problem introduced earlier, as well as the differentiation in demands from different classes of VMs and VM users. Furthermore, the violation of SLAs results in cash penalties for the provider, adding a direct economic dimension to the problem.

The main contributions of this paper are: (i) a VM to host allocation algorithm that considers the effect of the existing SLAs and monitoring data, (ii) usage of historical monitoring data to forecast the incoming load on both the physical and virtual infrastructure, in order to select the allocation that produces the highest profit contribution (maximize SLA satisfaction, minimize penalties, minimize energy consumption), (iii) extensive evaluation of the architecture in simulated cloud environments.

The remainder of the paper is structured as follows: Section II presents the related work; in Section III we introduce the system model, including the dependencies between the optimization criteria used for evaluating the cloud resource allocations, the cost model used for modeling the allocation, and assumptions about the incoming requests. In Section IV we describe the genetic group allocation algorithm, the multi-criteria evaluator and forecaster. Section V summarizes the technical implementation of the resource allocator and evaluator together with our design decisions. Section VI describes the experimental evaluation of the system, while Section VII contains our conclusions and future work.

## II. Related Work

We classify the related work into three areas: (i) multi-objective virtual machine to server allocation algorithms [5], [6] (ii) forecasting algorithms in resource allocation [7], [8], and (iii) SLA violation prevention [9], [10].

Mazzucco and Dyachuk [5] propose an approach for allocating VMs to servers by considering energy efficiency aspects by controlling the number of running servers in the datacenter. However, they do not consider the case when a server could host multiple VMs. They also use a forecasting approach for estimating the arrival rate, similar to the one described in this paper, but only use the number of running servers as the means of saving energy without considering consolidating VMs into fewer servers.

Xu and Fortes [6] describe a multi-objective resource allocation algorithm using a group oriented genetic algorithm with a fuzzy averaged fitness function while we propose a cost based multi-objective evaluation function using forecasted utilization levels based on historical monitoring data. The authors consider the initial allocation of virtual machines but not the costs associated with the SLA violations, nor the possibility of oversubscribing the resources based on forecasted utilization data.

978-3-901882-50-0 ©2013 IFIP

Zhang et al. [7] describe a runtime balancing system which uses statistical forecasting to determine if a VM will experience high CPU or network utilization during either day or night period and use this information to place the VM on corresponding hosts, while our approach uses a triple exponential estimation for forecasting of resource utilization, considering also the data seasonal trends.

Caron et al. [8] propose using a string matching algorithm for forecasting resource utilization demand in cloud environments by identifying the VMs with similar characteristics. Our approach differs by the used forecast algorithm, which considers data seasonal trends, and by using predefined values for resource utilization of unknown VMs.

Emeakaroha et al. [9] propose using a reactive method based on case based reasoning for determining actions in case of SLA violations together with using low level monitoring metrics for determining when SLA will be breached. In contrast, we support dynamic allocation for VMs, together with migration as a means of preventing SLA violations by using forecasting of resource utilization based on historical monitoring data.

Gambi et al. [10] propose a model driven framework for SLA violation prevention using detailed predefined models of the interactions between the physical and virtual resources including the services running on the virtual machines. They also consider seasonal and utilization trends, but do not take into consideration the cost aspect of the infrastructure, nor the impact on the energy consumption.

## III. System Model

A typical cloud environment consists of $h \in H$ servers each with a given amount of CPU cores $c_h$, main memory $m_h$ and available network bandwidth $b_h$. These servers will be used for hosting one or more virtual machines, which will use a predefined amount of server resources, as determined by the service level defined (e.g. standard, gold, silver) in the SLA agreement.

The purpose of SLAs [11], [12] is to define the guaranteed configuration [3] of the VMs in terms of CPU, memory and network bandwidth and to also specify their hourly utilization tariff. They can also be used for defining the penalties in case of SLA violation [3], [4]. As such, a dependency can be defined between the duration of the SLA violation with regards to a full month of utilization and a percent of the monthly bill which will be returned to the customer as a compensation for the suffered losses. This can be depicted using formula 1.

$$Penalty_{SLA} = \frac{a_i}{100}B,\ up \in (U_i^1, U_i^2],\ a_i \in [0, 100] \quad (1)$$

| Penalty [%] | Lower availability limit [%] | Higher availability limit [%] |
|---|---|---|
| 10 | 99 | 99.95 |
| 25 | 95 | 99 |
| 50 | 80 | 95 |
| 100 | 0 | 80 |

TABLE I
SLA penalties model

where $i$ is the penalty level, as shown in Table I, $Penalty_{SLA}$ is the penalty cost calculated as a percent $a_i$ of the monthly bills value $B$, when the uptime (in percent) $up$ is between the thresholds $U_i^1$ and $U_i^2$. An example of such penalty calculation can be that 10% of the monthly bill (B) will be returned to the customer if the uptime is between 99% and 99.95%. The estimated costs of violating the CPU or network SLAs is given by the time interval while the sum of estimated CPU/network utilization of each VM exceeds the available resources of the hosts. Formula 2 determines the uptime value used in formula 1 for calculating the penalty value

$$\begin{aligned} up &= \frac{1}{T}\sum_{i=1}^{T/t_m} c_i^{cpu}\, c_i^{net}\, t_m \quad (2) \\ c_i^{cpu} &= \begin{cases} 1 & if\ u_i^{cpu} < U_i^{cpu} \\ 0 & otherwise \end{cases} \\ c_i^{net} &= \begin{cases} 1 & if\ u_i^{net} < U_i^{net} \\ 0 & otherwise \end{cases} \end{aligned}$$

where $t_m$ is the monitoring interval, $T$ is the billing period (e.g. one month), $u_i^{cpu}$ and $u_i^{net}$ are the VM's CPU and network utilization level at the $i^{th}$ monitoring time slot, $U_i^{cpu}$ are the VM's host CPU and network utilization level (combined for all VMs running on the host) at the $i^{th}$ monitoring time slot, $c_i^{cpu}$ and $c_i^{net}$ are the SLA compliance indicators at the $i^{th}$ monitoring time slot.

For a given server, it is possible to model its energy consumption as a linear dependency of the CPU utilization [5]. In case of multi-core/CPU servers the average CPU utilization of all cores can be used. As a consequence of the fact that the idle power consumption is almost 60% of the one at full load [5], keeping servers in idle state or at low utilization would produce low revenues due to the fact that the server will consume almost as much energy as running with high load, but will generate only low revenues, if at all. Formula 3 describes the energy costs calculation, as described in [5].

$$\begin{aligned} P_i &= P_{idle} + (P_{max} - P_{idle})\,\overline{U_i^{cpu}} \\ C_{energy} &= C_{KWh} \cdot T \sum_{i=1}^{T/t_m} P_i \quad (3) \end{aligned}$$

where $P_i$ is the power consumption during $i^{th}$ monitoring time slot, $P_{idle}$ and $P_{max}$ are the host power consumption at idle and full load, $C_{KWh}$ is the energy cost per KWh and $C_{energy}$ is the total energy cost during $T$ time interval.

In a cloud environment, usually the VMs experience low CPU utilization, with 30% average [5] and usually having daily, weekly and monthly seasonality [13]. This helps to predict the resource utilization and do a better allocation by taking into account how much resources a VM will use, in fact enabling using the virtual machine live migration as a load balancing method [14].

The actual costs of VM migration can be expressed by formula 4 which is the cost caused by blocking resources on the destination host during the live migration, due to the reservation of memory before the migration. We ignored the VM downtime which is usually in terms of seconds.

$$C_{migration}^{VM} = C_{uh}^{VM} \cdot t_{migration} \tag{4}$$

where $C_{uh}^{VM}$ is the cost per hour of utilization of a VM and $t_{migration}$ is the estimation of time in hours needed for performing the migration. Migration time is estimated using a linear dependency [14], [15], [16] between the amount of VMs reserved memory and CPU utilization, as expressed by formula 5. The formula could be extended with other factors, such as the average network bandwidth utilization, as the calculation method would remain the same: applying the supperposition principle [17].

$$\begin{aligned} t_{migration} &= T_{idle}^{Mem} + \left(T_{f.load}^{Mem} - T_{idle}^{Mem}\right) \cdot \overline{u^{cpu}} \\ T_{idle}^{Mem} &= T_{idle}^{M_{min}} + \left(T_{idle}^{M_{max}} - T_{idle}^{M_{min}}\right) \cdot \alpha_{Mem} \\ T_{f.load}^{Mem} &= T_{f.load}^{M_{min}} + \left(T_{f.load}^{M_{max}} - T_{f.load}^{M_{min}}\right) \cdot \alpha_{Mem} \\ \alpha_{Mem} &= \frac{Mem}{M_{max} - M_{min}} \end{aligned} \tag{5}$$

where $Mem$ is the amount of main memory reserved for the VM, $\overline{u^{cpu}}$ is the average CPU utilization of the VM at the migration time, $T_{idle}^{Mem}$ and $T_{f.load}^{Mem}$ are the durations of the VM migration while it uses $Mem$ amount of memory and its CPU is idling, respective, at full load. The $T_{idle}^{Mem}$ and $T_{f.load}^{Mem}$ values can be either calculated by linear interpolation, by considering the time required for migrating a VM configured with the minimum, respective maximum amount of memory, at constant CPU utilization, either by directly measuring the live-migration time of a VM with the specified characteristics in terms of memory and average CPU utilization. The $\alpha_{Mem}$ is the percentage that $Mem$ represents of the considered memory range [$M_{max}$ , $M_{min}$].

Although the technical implementation of monitoring physical and virtual resources is not in the focus of this paper, they play an important role [18], especially the monitoring interval, $t_m$. The selected value of this parameter will be given in the implementation section.

We consider four objectives in our approach at allocating the virtual resources: maximizing the total revenues, minimizing the energy costs, minimizing the migration costs and also minimizing the SLA penalty costs. These different objectives are linked by a aggregate objective function which will evaluate the resource allocations using the function as described in equation 6.

$$\begin{aligned} F_{avg}(obj) &= w_r \sum_{i=1}^{M} \left(C_{uh}^{i} \cdot T^{i}\right) - w_e \sum_{i=1}^{H} C_{energy}^{i} - \\ &\quad -w_m \sum_{i=1}^{M^{'}} C_{migration}^{i} - w_p \sum_{i=1}^{M^{''}} Penalty_{SLA}^{i} \\ obj &= [w_r, w_e, w_m, w_p] \end{aligned} \tag{6}$$

where $obj$ is the evaluation objective composed of four weights: $w_r$ for revenues, $w_e$ for energy costs, $w_m$ for migration costs and $w_p$ for SLA penalty costs, $M$ is the total number of VMs, $C_{uh}^{i}$ is the per-hour utilization tariff of VM $i$, $T^i$ is the utilization period of VM i during the given billing period $T$, and $M^{'}$ is the number of migrated VMs, $M^{''}$ is the number of VMs with SLA violations, and $Penalty_{SLA}^{i}$ is the SLA penalty cost of VM $i$.

An example of a final objective could be maximization of the total profit, in which case each objective would have an equal importance, represented by giving each weight the value one. The variation in the weights can be also used for offering different price plans to the client, depending on the desired type of optimization. It might be the case that a provider might want to offer a 'green' service, with a strong emphasis on energy efficiency, in which case he will increase the corresponding weight of the energy costs. This, however, will affect the other objectives, for example, the costs with the SLA penalties, as the allocator might select fewer physical servers for running the VMs in order to decrease the energy costs. Another possibility would be to offer a performance-oriented service, in which case, the weights corresponding to the migration costs and SLA penalties would be increased, which would affect the produced allocations by using more servers and would raise the energy costs.

An efficient planning algorithm will try to find a global optimum for the allocation of VMs to hosts for a given billing period by maximizing the total revenues. Our proposed algorithm attempts to achieve this by performing a search of a local optimum with a time horizon of one billing period (e.g. one month).

## IV. Planning and Forecasting

The problem of allocating VMs to physical hosts can be seen as a bin-packing problem, which is known to be NP-hard, suggesting the need for a heuristic solution. Genetic algorithms are a class of heuristic solutions that can make use of multi-objective evaluation functions for searching solutions in multi-dimensional spaces.

Given the nature of the VM allocation problem, we selected a specialized version of genetic algorithms oriented at groups [19]. The group oriented genetic algorithm [20] operates on groups of objects, leveraging the fact that the VMs are naturally grouped by the servers on which they are deployed and thus maintains the previously determined good properties of the groups.

The solution space of our allocation problem is multi-dimensional due to the multiple objectives used for evaluating them, such as the costs of SLA violation, energy consumption, VM migration and the total revenues. As the problem of allocating VMs to physical hosts is an NP-hard combinational optimization problem, it is not feasible to demonstrate the optimality of a solution. The produced solutions are Pareto-optimal [21] regarding the applied criteria. In our case, the criterion used for selecting a solution is the value of the predicted total profits of the allocation, considering the costs of the SLA violations, energy consumption and VM migrations.

Given that the average CPU utilization of a VM hosted in a data center is usually around 30% [5] it is common to overcommit the CPUs of the physical hosts with factors between 1.5 and 3 [5], [22]. For example, OpenNebula [23] allows allocating virtual CPUs to a fraction of the available physical CPUs, using support of Xen and KVM [24] hypervisors. This, however, could lead to violating the SLAs in case that the collocated VMs simultaneously experience high CPU utilization [22]. Given the fact that a server uses almost 60% [5] of the total power consumption when running idle, distributing VMs across a large number of servers results in poor server utilization levels and would hence also diminish revenues due to the high amount of energy used per VM. Our genetic algorithm allocator with load forecasting mitigates these problems for VMs with an existing monitoring history, by choosing the best current allocation that minimizes the costs of the allocation at the next time step, given that domain-specific constraints from the application are not breached.

Although it is possible to overcommit both CPU ([22], [25]) and memory ([25], [26]) in modern hypervisors, we are considering only oversubscription of the CPU, as the oversubscription of memory is usually associated with system instability [25].

Below, the basic structure of the genetic algorithm used is given.

*Step 1:* Generate the initial population

1) randomly allocate genes (VMs) to groups (servers) using the first-fit heuristic
2) ensure the chromosome is valid with regards to VM's allocated memory by reinserting the excluded genes in the groups using the first-fit heuristic

*Step 2:* rank and sort the population using the fitness function defined in equation 6

*Step 3:* copy E ”elite” chromosomes (with highest fitness value) into the new population

*Step 4a:* generate an offspring using crossover with a given probability

1) select two chromosomes from the population using fitness proportionate selection
2) generate offspring using the group-crossover operator
3) ensure the chromosome is valid with regards to VM's allocated memory by reinserting the excluded genes in the groups using the first-fit heuristic

*Step 4b:* or select the fittest of the parents

*Step 5:* mutate the offspring with a given probability

*Step 6:* rank and sort the population using the fitness function defined in equation 6

*Step 7:* If the stopping criterion is reached, terminate the search and return to the current population, else, go to *Step 3*.

The group oriented genetic algorithm [20] searches for solutions by generating populations of chromosomes composed of genes which belong to groups. In our implementation, a chromosome encodes an allocation by representing the groups as hosts and the genes as VMs packed onto a given group. The initial population is created by applying a heuristic algorithm, such as first-fit. Next, the population is sorted according to the multi-objective fitness function. At each step the algorithm performs two group oriented operations on the current population: crossover and mutation (described below). Top 10% chromosomes from the current population are passed into the next population as elitism seems to improve solution convergence [21]. We use two criteria for ending solution searching. First is determining when there are no more improvements, or they are below a certain threshold, in the overall fitness value of a population. Second involves finding when a solution is not possible (e.g. the profits generated by the new allocation are lower than the initial allocation due to the costs of SLA violations, VM migrations and energy consumption).

Each allocated group inside each chromosome needs to have another local allocation applied for determining the allocation of physical CPU cores to the VM cores. This is achieved by running a similar genetic algorithm as the one used for allocating VMs to hosts. The allocation is performed every time a group is changed.

The evaluation of each chromosome is performed by calculating the predicted energy costs for the allocation (using a linear power model [5] combined with forecasted CPU utilization data), revenues generated by the allocation assuming one period of utilization, predicted costs caused by CPU/network SLA violations (determined using forecasted utilization data) and costs associated with VM migrations - which are the values of the objective functions. These values will then be combined by the aggregate objective function, as described in equation 6.

The next step consists of applying the roulette selection [27] for identifying two possible candidates for producing the new chromosome. With a given probability either the group-crossover operator is applied for producing a new offspring, or the fittest chromosome if selected. After this, with a given probability, the mutation operator is applied to the offspring, before adding it to the new population.

After the population has been created, it will be re-evaluated and the process is repeated until the stop condition is encountered.

The group-oriented crossover genetic operator functions by retaining the qualities of the groups and selecting from each chromosome the corresponding group with the highest fitness value, thereby preserving the good allocations. After all groups have been processed it is possible that there are unallocated VMs. For these a 'healing' process is applied by redistributing them according to first-fit-descending heuristic.

The mutation operator is applied by randomly removing a VM from a CPU or network oversubscribed host and re-allocating it according to the first-fit heuristic. A key component of the planning system is the forecasting module used by the allocator algorithm in evaluating the fitness of various VM to host distributions based on the forecasted VM request rate, CPU core and network utilization. Given the fact that both the VM request and resource utilization distributions experience

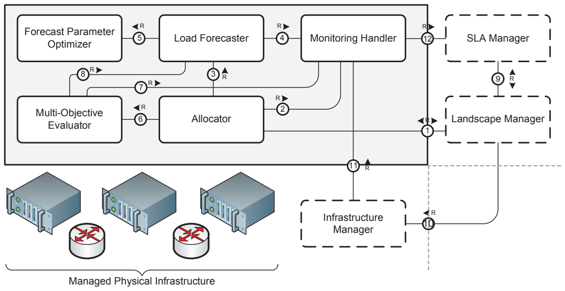

Fig. 1. System Control Loop

daily, weekly or monthly patterns [13] we have selected the Holt-Winters algorithm [28] for performing triple exponential smoothing of the utilization data.

The Holt-Winters algorithm performs an exponential smoothing of the data by assigning exponentially decreasing weights to the past data comprised of a period, considering also the data trend and seasonality. The seasonality refers to repeating of a data pattern after a given number of periods, called season. The trend refers to the tendency of data to either increase or decrease in the long term. In our case, the monitoring data obtained from the VM's CPU and network utilization, as well as the number or VM requests, experience seasonality with daily and monthly patterns [29].

## V. Implementation

The system functions in a control loop, as shown in Fig. 1. First, the requests are prepared by the Landscape Manager and then are passed to the Allocator module (1) for determining how the virtual machines should be allocated on the physical infrastructure. The allocator uses the historical data from the Monitoring module (2) and the system load forecast (3) for producing an allocation of the VMs to hosts. The actual load forecast is determined using historical monitoring data (4), using of specifically fitted forecast parameters calculated (5) by the Forecast Parameters Optimizer module. Next, the allocation is passed (6) to the Multi-Objective Evaluator, which then uses both existing monitoring data (7) and the forecasted resource utilization data (8) in order to calculate the costs involved in the allocation. After selecting the allocation with the minimal cost, the Allocator returns it to the Landscape Manager which will then register the new virtual resources with the SLA Manager (9) and will instruct the Infrastructure Manager (10) to actually provision the resources on the physical resources. The control loop is closed by the returning of the monitoring data to the Monitoring Handler (11), followed by informing the SLA Manager about the resources' state (12). Three implementations of allocator algorithms were created in Java for the planner system: First Fit [30] Descending, Best Fit Descending and the Genetic Group-oriented with Forecasting. The actual Holt-Winters forecasting is delegated to an external implementation of R Statistical Computing [31]. The forecasted series are kept in a memory cache, as the prediction values are reused multiple times by the genetic allocator. The forecast is run in a progressive manner, first after on hour from the VM instantiation, then after 6, 12, 24, 48, 72, 168, 168, 216, 720 hours, after which the forecast is recalculated every month.

Our system makes some assumptions about characteristics of the incoming load, such as the distribution of load according to hourly and daily patterns, having one service instance per VM, and having a predictable trace of CPU and network utilization. We assume that the VM network utilization data refers only to inter-hosts traffic but not to the intra-host traffic of the collocated VMs; and that the VM memory is always reserved all at once, while the VM's CPU cores are allocated to the physical CPU cores using the affinity mechanism [32]. We assume that the monitoring samples are collected every 5 minutes, in order to keep the generated data volume to a manageable value.

While the assumption of having just one type of service per VM seems restricting, this might be needed in an environment with automatic scaling in order to enable taking the decision on when the service should be scaled, based on previously agreed SLAs. This, however, does not prevent the existence of composite applications containing multiple services. An example of such application, with which we experimented, is an Enterprise Information System (EIS) composed of a load-balancer, a session handler, multiple workers and a storage service. We will describe in a future paper how the SLA-based System Landscape Orchestrator works to perform the dynamic instantiation, configuration and scaling of the services. Even if there are multiple service instances per VM, this does not change the nature of the VM-to-Host allocation problem, as the resource metrics would remain the same.

The penalty model used for calculating the costs of violating the CPU or network SLAs is described in Table I. The penalty in percent refers to the amount of the current bill that will be paid to the customer in the next moth if the SLA availability is between the lower and the upper bounds. This implies that the target SLA availability is 99.95

For estimating the VM migration duration (Fig. 2) we used a linear model, validated against experimental data [15], [16], dependent on the amount of reserved memory and on the average CPU load of the VM. The figure represents the linear dependency between the VM migration time and the amount of reserved memory for when the CPU utilization is near 0% (idle) and almost at 100% (full load). The actual

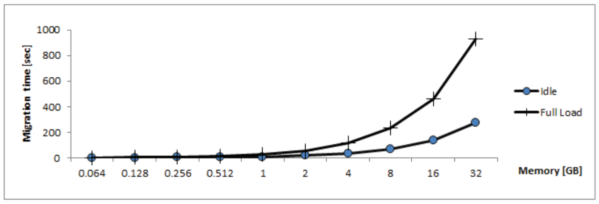

Fig. 2. Migration time vs. memory and CPU load

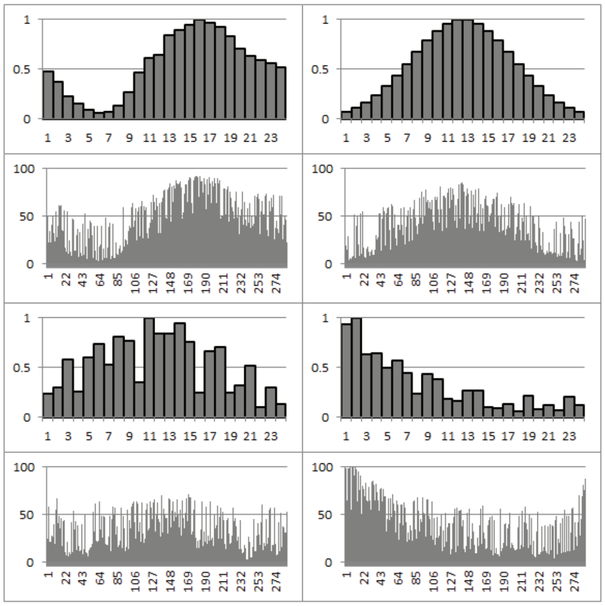

Fig. 3. Distribution of resource utilization: above, trend per hour, below, trace per 5 minutes time slot

migration time is determined by interpolating the time for the average CPU load using the values for the migration at idle and full CPU utilization. Also, we model the live-migration impact of the VM by increasing the load of the VM's CPU with 10% over a monitoring period, and also increasing the network bandwidth utilization with the value amount required for transferring the VM's reserved memory.

## VI. Evaluation

We conducted a number of simulations for validating the system's characteristics with regards to the multi-criteria optimization of VM allocation. For this purpose we used synthetic generated VM request traces with seasonal distributions (matching Fig. 3) in order to load the system. For each VM we generated CPU and network traces with which we fed the monitoring and forecasting modules. We also varied the amount of noise added to the traces between 30% and 90%, in order to test the system stability.

In order to test the multi-objective evaluator, we selected the scenario in which the provider wishes to maximize his profits and so we assigned the value of 1 to all four weights described in section III.

We simulated a month of VM requests, including adding new VMs and removing existing ones, and we compared the SLA violations and energy efficiency of the allocations produced by the first-fit and energy aware algorithms with the group-oriented genetic algorithm with forecasting. The actual calculations for determining the costs of SLA violations and energy consumption were performed using the generated monitoring data and not the forecasted data.

For further testing of the algorithm's stability, we varied the genetic algorithm's parameters considering three different population sizes corresponding to one, two and four utilization weeks, four values for crossover probability (0.3, 0.5, 0.8 and 1) and the same four values for mutation probability. The results were consistent with the ones described below.

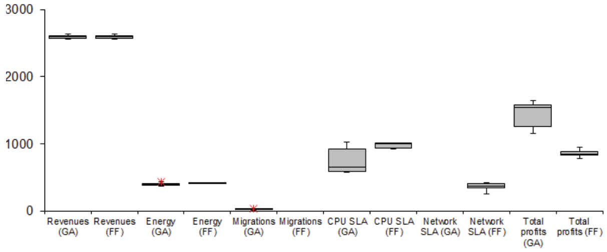

Fig. 4. Cost and profit distribution for GA and FF allocators

Our simulated infrastructure was composed of 10 hosts with quad-core CPUs, 16GB RAM and Gigabit networking. We varied the number of VMs between 10 and 50. The algorithm performed in average 1600 live migrations per simulation month (equivalent with two live migrations per hour) and at most one VM was selected for migration at a time.

Our results (Fig. 4 - due to space constraints, we include only one comparison between allocators) show a consistent 100% reduction of network-SLA penalties together with a 30% reduction in CPU-SLA penalties, at the expense of below 1% of the revenues used for live migration. The total profit generated by using the GA allocator were in average 50% higher than the ones generated by the FF and energy-aware allocators. Also, the GA allocator distributes better the load across the infrastructure, leading to a more uniform host utilization, lowering of the total energy consumption and reducing host wear.

## VII. Conclusions

We described a system for supporting the planning and load distribution disciplines of SLA management while taking into consideration multiple objective optimizations and the impact of SLAs into resource provisioning and into dynamic scaling of the virtual infrastructures. We proposed a way of combining resource utilization estimation, with cost prediction and impact of the infrastructure operations for implementing the complete set of disciplines used in SLA management. We validated our model using simulation data and we were able to show that our proposed resource allocation approach significantly outperforms several standard heuristics. We plan to extend our work to more complex scenarions as the ones found in enterprise information systems and to compare the results against more sophisticated algorithms.

### *Acknowledgments*

The work in this paper has been (partially) funded by the European Union through project GEYSERS (contract no. FP7-ICT-248657). We also would like to thank Matthias Thoma and the anonymous reviewers for their valuable comments and suggestions for improving the quality of this paper.

## REFERENCES

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, “Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility,” *Future Generation computer systems*, vol. 25, no. 6, p. 599616, 2009.

[2] L. Schubert and J. Keith, “Advances in clouds,” European Union, Tech. Rep., 2012.

[3] Amazon Web Services, “Amazon EC2 service level agreement,” http://aws.amazon.com/ec2-sla/.

[4] Google App Engine, “Google app engine service level agreement,” https://developers.google.com/appengine/sla.

[5] M. Mazzucco and D. Dyachuk, “Optimizing cloud providers revenues via energy efficient server allocation,” *Sustainable Computing: Informatics and Systems*, vol. 2, no. 1, pp. 1 – 12, 2012.

[6] J. Xu and J. Fortes, “Multi-objective virtual machine placement in virtualized data center environments,” in *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int’l Conference on Int’l Conference on Cyber, Physical and Social Computing (CPSCom)*, Dec. 2010, pp. 179 –188.

[7] Z. Zhang, H. Wang, L. Xiao, and L. Ruan, “A statistical based resource allocation scheme in cloud,” in *Cloud and Service Computing (CSC), 2011 International Conference on*, Dec. 2011, pp. 266 –273.

[8] E. Caron, F. Desprez, and A. Muresan, “Forecasting for grid and cloud computing on-demand resources based on pattern matching,” in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, Dec. 2010, pp. 456 –463.

[9] V. C. Emeakaroha, R. N. Calheiros, M. A. Netto, I. Brandic, and C. A. De Rose, “DeSVi: an architecture for detecting SLA violations in cloud computing infrastructures,” in *Proceedings of the 2nd International ICST Conference on Cloud Computing (CloudComp10)*, 2010.

[10] A. Gambi, M. Pezze, and M. Young, “SLA protection models for virtualized data centers,” in *Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS ’09. ICSE Workshop on*, May 2009, pp. 10 –19.

[11] M. Kajko-Mattsson and C. Makridis, “Outline of an SLA management model,” in *Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on*, Apr. 2008, pp. 308 –310.

[12] A.-F. Antonescu, P. Robinson, L. M. Contreras-Murillo, J. Aznar, S. Soudan, F. Anhalt, and J. A. Garcia-Espin, “Towards cross stratum SLA management with the GEYSERS architecture,” in *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, Jul. 2012, pp. 527 –533.

[13] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, “Youtube traffic characterization: a view from the edge,” in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, ser. IMC ’07. New York, NY, USA: ACM, 2007, pp. 15–28. [Online]. Available: http://doi.acm.org/10.1145/1298306.1298310

[14] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, “Cost of virtual machine live migration in clouds: A performance evaluation,” in *Cloud Computing*, ser. Lecture Notes in Computer Science, M. Jaatun, G. Zhao, and C. Rong, Eds. Springer Berlin / Heidelberg, 2009, vol. 5931, pp. 254–265. [Online]. Available: http://www.springerlink.com/content/7052149402656584/abstract/

[15] “IBM techdocs white paper: Evaluating microsoft hyper-v live migration performance using IBM system x3650 m3 and IBM system storage DS3400,” http://129.33.205.81/jct03001c/support/techdocs/atsmastr.nsf/WebIndex/WP101828, Dec. 2010.

[16] M. Nelson, B. H. Lim, G. Hutchins *et al.*, “Fast transparent migration for virtual machines,” in *Proceedings of the annual conference on USENIX Annual Technical Conference*, 2005, p. 2525.

[17] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab, *Signals and systems*. Prentice Hall, 1997.

[18] C. Molina-Jimenez, S. Shrivastava, J. Crowcroft, and P. Gevros, “On the monitoring of contractual service level agreements,” in *Electronic Contracting, 2004. Proceedings. First IEEE International Workshop on*, 2004, p. 18.

[19] H. Iima and T. Yakawa, “A new design of genetic algorithm for bin packing,” in *Evolutionary Computation, 2003. CEC ’03. The 2003 Congress on*, vol. 2, Dec. 2003, pp. 1044 – 1049 Vol.2.

[20] E. Falkenauer, “A hybrid grouping genetic algorithm for bin packing,” *Journal of heuristics*, vol. 2, no. 1, p. 530, 1996.

[21] A. Konak, D. W. Coit, and A. E. Smith, “Multi-objective optimization using genetic algorithms: A tutorial,” *Reliability Engineering & System Safety*, vol. 91, no. 9, pp. 992–1007, Sep. 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0951832005002012

[22] “High performance SQL server workloads on hyper-v,” May 2010. [Online]. Available: http://download.microsoft.com/download/D/F/8/DF89D22D-39C8-4728-A990-3BD4467891B7/HighPerformanceSQLServerWorkloadsOnHyper-V_Final.docx

[23] D. Milojic andic and, I. M. Llorente, and R. S. Montero, “Opennebula: A cloud management tool,” *Internet Computing, IEEE*, vol. 15, no. 2, pp. 11 –14, march-april 2011.

[24] F. Camargos, G. Girard, and B. Ligneris, “Virtualization of linux servers: a comparative study,” in *Proceedings of the Linux Symposium*, vol. 47, no. 06, 2008, pp. 63–76.

[25] D. Williams, H. Jamjoom, Y. H. Liu, and H. Weatherspoon, “Overdriver: Handling memory overload in an oversubscribed cloud,” in *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, 2011, p. 205216. [Online]. Available: http://dl.acm.org/citation.cfm?id=1952709

[26] C. A. Waldspurger, “Memory resource management in VMware ESX server,” *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, p. 181194, Dec. 2002. [Online]. Available: http://doi.acm.org/10.1145/844128.844146

[27] T. Bäck, *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, USA, 1996.

[28] P. S. Kalekar, “Time series forecasting using holt-winters exponential smoothing,” *Kanwal Rekhi School of Information Technology*, 2004.

[29] “Host server cpu utilization in amazon ec2cloud,” http://huanliu.wordpress.com/2012/02/17/host-server-cpu-utilization-in-amazon-ec2-cloud/.

[30] A. C. Yao, “New algorithms for bin packing,” *Journal of the ACM*, vol. 27, no. 2, p. 207227, 1980.

[31] R Development Core Team, *R: A Language and Environment for Statistical Computing*, ISBN 3-900051-07-0, R Foundation for Statistical Computing, 2008. [Online]. Available: http://www.R-project.org

[32] Z. Li, Y. Bai, H. Zhang, and Y. Ma, “Affinity-aware dynamic pinning scheduling for virtual machines,” in *2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, Dec. 2010, pp. 242 –249.