Genetic Algorithms for Energy Efficient Virtualized Data Centers

Helmut Hlavacs, Thomas Treutner
Research Group Entertainment Computing
University of Vienna, Austria
Email: firstname.lastname@univie.ac.at

***Abstract*—Avast majority of servers in classical data centers of all scales are underutilized for a significant amount of time. These servers operate at a very low rate of efficiency and consume huge amounts of energy. In this work, we investigate how dynamic consolidation and workload forecasting can be exploited to increase the energy efficiency of a virtualized, heterogeneous server infrastructure. We base our evaluation on real utilization traces of a production system operated by the University of Vienna's central IT department. The traces contain the CPU utilization of more than 30 VMs over a period of four weeks. These VMs offer all kinds of services to students, staff and other visitors. We use these traces to investigate a business infrastructure scenario, where energy costs are just one of several parts of operational costs. We present a novel cost model using configurable penalties for the most important operational cost categories. We compare the total costs of a bin-packing related heuristic and a new genetic VM mapping algorithm used for dynamic consolidation (GA) and load balancing (LB). We trade-off forecasting against resource reserves in combination with shorter measurement intervals. We demonstrate the flexibility of a genetic algorithm. The GA and LB approaches are directly influenced by penalizing cost parameters. Our cost model allows easy adaption by infrastructure operators to implement custom priorities and optimization goals.**

I. Introduction

A vast majority of servers in classical data centers of all scales are underutilized for a significant amount of time. These servers operate at a very low rate of efficiency [1] and consume huge amounts of energy. It is estimated by the Uptime Institute [4] that beginning with 2009, a typical server causes higher electricity costs over a 3 year lifespan than the purchase cost. To find a remedy, modern data centers frequently use static system virtualization to consolidate a set of idle or badly utilized server to a smaller set of better utilized servers. There are many different approaches to virtualization, but in general, computing resources are virtualized to gain higher efficiency and flexibility. However, business infrastructures often have to bear workload with strong periodic components. Here, static reservation of virtualized resources is a dead end regarding efficiency. Virtualization by itself can not change that provisioning resources for day hours is wasting resources during the night hours. On the contrary, provisioning resources for night hours will typically result in catastrophic quality of service and annoyed users during the day. In both cases, the absence of an automatism causes unnecessary costs. Especially in environments with periodic workload variability (e.g., business infrastructures with 9-5 cycles), automatic resource allocation mechanism are most beneficial [3]. In this work, we investigate the combined potential of workload forecasting and dynamic consolidation based on real utilization traces of a production system operated by the University of Vienna's central IT department. The traces contain the CPU utilization of more than 30 VMs over a period of four weeks, offering all kinds of services to students, staff and other visitors. Due to the lack of variability in the memory utilization, we concentrate on the CPU utilization. However, it is important to have a target utilization in mind that is to be achieved. Werner Vogels [11] , CTO and Vice President of Amazon, has mentioned that achieving 40% utilization is a major success for workloads that are not strictly CPU-bound. To accomplish this challenge, and in consequence, to dynamically scale the power consumption of a virtualized business infrastructure to the workload situation in an automatic way, two major components must be implemented and carried out by a management system: a) VM workload forecasting, and b) Energy efficient mapping of VMs to servers. We will briefly discuss these two topics in the following.

A. VM workload forecasting

The VMs' resource allocations must be automatically adapted to their near future resource requirements to avoid both resource wastage and shortage. In contrast to number-crunching infrastructures as e.g., HPC clusters, business infrastructures offering services to users are likely to show periodic patterns related to the exact nature of the service and its users' habits. With more and more formerly stand-alone applications being transformed into Cloud services, this pattern will even get stronger. Unconsciously, millions of users will not only use their own device for everyday tasks (e.g., reading their mails, fixing appointments in their calendar or browsing their picture gallery) but also a growing amount of infrastructure hosting these Cloud services. The field of time series analysis offers several methods. We evaluate two approaches, the Holt-Winters method (triple exponential smoothing considering seasonal patterns) and auto-regressive moving-average methods (ARMA, ARIMA, SARIMA, etc.). We will further discuss our approaches in Section IV-C.

B. Energy efficient mapping of VMs to servers

The VMs' distribution across the servers must be automatically adapted to the workload forecasts, aiming at consuming the least possible power given the workload intensity and potential service level constraints. VMs can be live migrated [5] to consolidate a set of idle servers to a smaller set of well utilized servers, thus saving energy by switching off servers. Live migration should be carried out preventively to avoid serious impacts on service quality [8] . Hence, workload forecasting is not only important to avoid direct impacts on service quality due to e.g., an overloaded server, but also indirect impacts caused by management actions.

Finding an optimal mapping of VMs to servers based on a current mapping is a combinatorial optimization problem. For V VMs and R RMs (physical servers), there are R^V theoretically possible mappings, resulting in a huge search space even for low V and R. Mappings with at least a single overloaded RM are invalid in the context of this work. Thus, a vast amount of theoretically possible solutions found are practically irrelevant and the computation involved was in vain. Additionally, practically possible solutions must be evaluated for their cost. Important cost categories are energy consumption, number of live migrations, server boots/shutdowns, etc. An exhaustive search through V^R possible mappings is most likely intractable because of the computational demand involved, given practically interesting values of V and R. We examine two algorithms to map VMs to servers: A heuristic approach derived from bin-packing, denoted by Balanced-First-Fit (BFF), and a genetic algorithm (GA) flexibly adaptable to various scenarios. The adaption is possible by using a customizable cost model feeding into the GA's target function. We will further address them in Section IV-D.

II. Related Work

Bobroff et al. [3] have studied the potential of dynamically consolidating periodic workload regarding required online capacity and possible SLA violations. They analyse several different basic workload profiles and show that periodic workload is most suitable for dynamic consolidation, implemented by a Measure-Forecast-Remap algorithm. The mapping step is done by a first-fit bin packing heuristic, minimizing the number of online servers. This target function is only applicable to completely homogeneous server pools, and even under such restricted circumstances, bin packing heuristics often deliver results far from optimal. Some examples are mentioned below. Although energy costs are mentioned in [3] , they are not part of the evaluation.

Entropy [7] is an open-source implementation of dynamic consolidation of virtual machines. It is focusing on HPC clusters, where broader assumptions can be made and higher packing rates of VMs are possible due to the typical on/off characteristic of number-crunching tasks. pMapper [10] is another example for such a system focusing on HPC.

Beloglazov et al. [2] have presented an architecture for energy efficient Cloud computing, trying to minimize energy costs under QoS constraints. For mapping VMs to servers, they apply an adapted version of the Best Fit Decreasing heuristic. These heuristics were originally developed for tasks related to bin packing. They can happen to create solutions far from optimal, especially when a server infrastructure is assumed to be heterogeneous. Then, a solution using the minimum number of bins (servers) is not necessarily the solution consuming the least energy. A sorting criteria is also required for the servers to decide which of them are to be filled first. In [2], a VM is mapped to the server that shows the least increase in energy consumption. We interprete that this criteria is also used for the first VM, so that the increase in energy consumption is then dominated by the chosen server's idle power. However, such measures can not avoid that the final mapping is obviously not consuming the least energy. E.g., it could have been better to choose a different server, although it is showing a bigger power consumption increase in the first step, but offering a smaller slope in the following due to a different CPU model, finally leading to a lower energy consumption. Bin packing related heuristics have also been studied by Wood et al. [12] .

In general, at least CPU and memory utilization should be taken into account when mapping VMs, so balancing a server's resource components is an important artefact of every heuristic. Furthermore, reconfiguration costs, here most importantly live migration and server state change overheads, must be considered. Therefore, we define a cost model in Section IV-A, where several cost categories contribute with configurable weights to a sum of weighted costs, which we try to minimize. These are just briefly mentioned reasons why we compare a heuristic VM placement algorithm with a genetic algorithm, which is far more flexible within still reasonable computation time. Of course, exact solutions are practically impossible due to the huge number of possible solutions [9].

Finally, a proper base of comparison is required to evaluate algorithms implementing dynamic consolidation. We regard it reasonable to collate dynamic consolidation against load balancing, as they represent opposed paradigms. The latter always uses all available resources in way that ignores energy costs in favour of very low SLA violation probability. The former always uses the minimum required amount of resources that keeps QoS on an acceptable level, risking failure to do so. Hyser et al. [9] have implemented a load balancing policy based on simulated annealing. We compare two implementations of dynamic consolidation with a load balancing policy implemented by a genetic algorithm, where the whole server pool is used at all times.

III. Utilization Trace Data

The utilization traces thankfully provided by the University of Vienna's central IT department consist of CPU utilization traces of 33 VMs running in a production system, thus bearing real workload. The CPU utilization is measured in MHz, giving the amount of CPU cycles spent by the VMware hypervisor for a specific VM. The traces represent a period of four weeks, with intervals of two hours, where a utilization sample gives the average utilization within an interval. The infrastructure is driven by VMware, offering only little flexibility regarding utilization trace export. Trace length and resolution are not arbitrarily configurable. Traces representing a period of time longer than four weeks are not possible with a resolution of two hours. Already, an interval length of two hours is rather long, thus we will evaluate shorter interval times using interpolation.

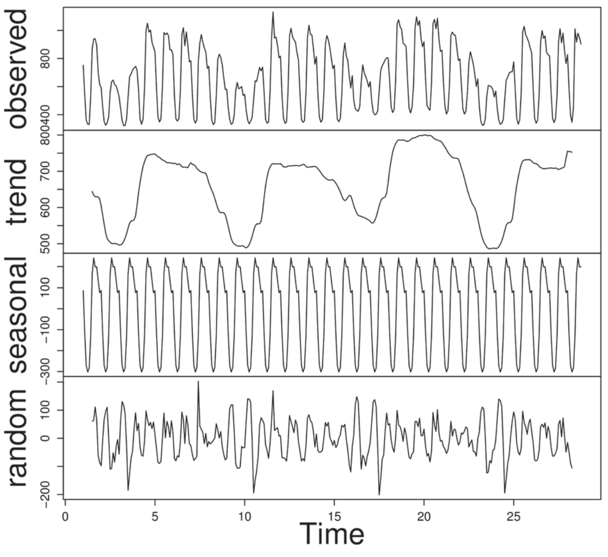


Fig. 1. Decomposition of a typical CPU utilization trace into a trend, seasonal and random component.

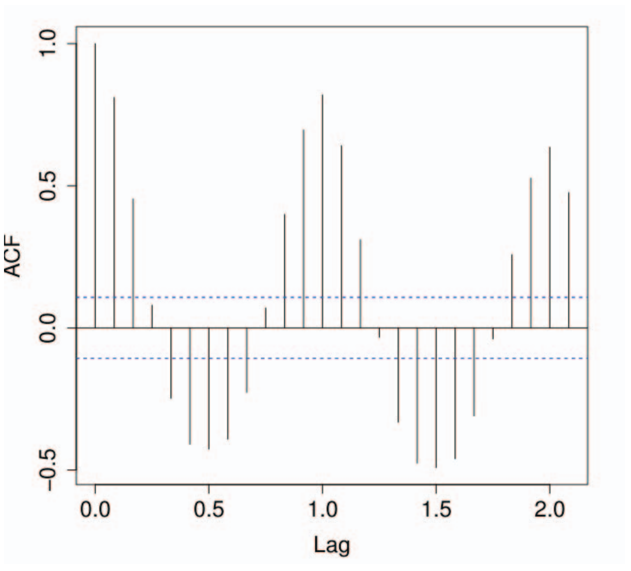


Fig. 2. Autocorrelation plot of the same trace as in Figure 1.

Interestingly, the infrastructure's memory utilization showed little to no variability at all. We explain this by the memory management behaviour of UNIX-like operating systems, which use otherwise unrequired memory space for buffering and caching and quickly reach a steady state. On the other hand, Windows operating systems have the peculiarity to zero-fill the whole memory during startup, thus effectively prohibiting a hypervisor from assigning memory pages to a guest on demand. Hence, we concentrate exclusively on CPU utilization and assume a VM's memory allocation as heterogeneous, but fixed over time.

Figure 1 shows a typical CPU utilization trace and its decomposition into a trend, seasonal and random component. We can clearly see the differences between day times and night times, as well as between weekdays and weekends. During weekdays, there is almost a factor of three between minima and maxima. The timeseries starts at 9pm, and we can clearly see a pattern that during morning hours, utilization rises quickly to stabilize during day hours, slowly decreases at afternoon until it quickly declines at evening.On weekends, the pattern is qualitatively similar, but quantitatively weaker.

In Figure 2, an autocorrelation plot of the same trace is depicted. A lag of 1.0 represents 1 period of the timeseries, which is here 24 hours. As expectable, there is a strong negative autocorrelation at lags around 0.5 (12 hours) and even stronger, but positive autocorrelation at lags around 1.0. The blue dashed lines in Figure 2 mark the range of statistically significant autocorrelations.

Of course, the trace briefly presented above is just one of over 30 traces, nevertheless it is a qualitatively typical sample, representing a common pattern within the traces. However, not all of the traces show such a strong negative/positive autocorrelation at 12/24 hour lags. Figures 3 and 4 show the autocorrelations of all VMs at a lag of 12 hours and 24 hours, respectively. Interestingly, although most VMs show strong positive autocorrelation at a lag of 24 hours, not all of them show a strong negative autocorrelation at a lag of 12 hours. Some VMs even experience positive autocorrelation at a lag of 12 hours. This gives evidence of heterogeneity regarding the VMs' usage patterns, which will be considered in Section IV-C.

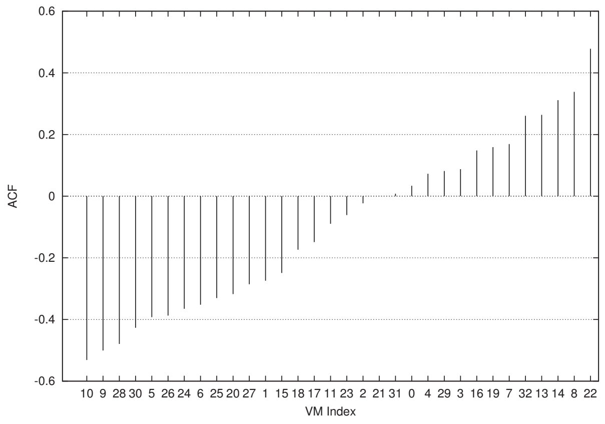


Fig. 3. Autocorrelations at lag 0.5, meaning 12 hours.

IV. Evaluation

In order to evaluate the combined potential of both workload forecasting and dynamic consolidation regarding energy efficiency, we have developed a custom simulation. It allows us to use utilization traces from several data sources, here plain text

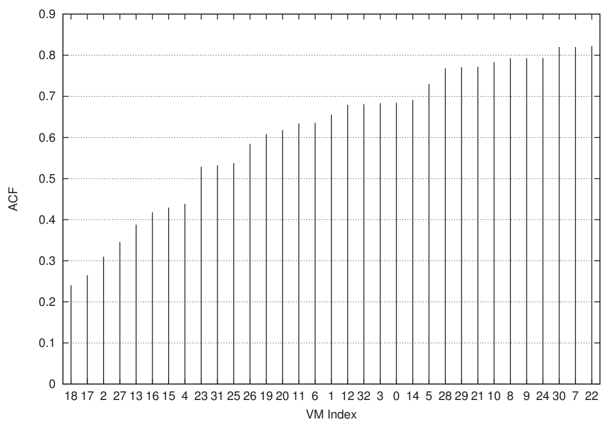


Fig. 4. Autocorrelations at lag 1, meaning 24 hours.

files. Optionally, values are linearly interpolated to implement a preconfigured resolution. For future work, we plan to use splines instead of linear interpolation. The simulation's core is a control loop, where we denote the (simulated) time between control loop iterations as *interval length* and each iteration as *interval*. To create a simulation run, four components are necessary:

1) The simulation input data, comprised of number a of VMs and their resource demands, and a list of servers and their characteristics, e.g., resource capacities, power consumption etc.
2) A forecasting implementation, cf. Section IV-C .
3) A VM mapping implementation, denoted as *decision module*, cf.Section IV-D .
4) Several, partly optional parameters influencing the simulation itself or its evaluation, cf. Section IV-B .

In the first interval an initial mapping is created, using the chosen decision module. The initial mapping is not evaluated, to give each decision module the chance to warm-up by creating a reasonable initial state according to its algorithm. Subsequently and in each iteration, the following steps are carried out:

1) VM utilizations are updated, as they will have changed since the last observation. Based on the mapping from the last interval which was not recalculated by a decision module yet, server utilizations are updated accordingly. E.g., this can lead to the detection of server overload if the mapping chosen in the last interval was too optimistic or aggressive, or forecasting went wrong.
2) Before recalculating the mapping, it is evaluated regarding servers having reached a warning or alarm level of utilization. The exact thresholds are externally configurable, cf. Section IV-B .
3) If forecasting is enabled, then for each VM, a utilization forecast is calculated given a chosen implementation and all historic utilizitation observations, chronologically relative to the current interval. Thus, we are effectively creating a new model in every interval for every VM.
4) A new mapping is calculated by a chosen decision module. Several operations are possible here, e.g., VMs are migrated to another server, servers are powered down or woken up, and related information like energy consumption is updated accordingly.
5) Finally, the new mapping is evaluated considering several aspects, feeding into a cost function we elaborate on in Section IV-A .

A. Mapping Costs

We consider several aspects of a mapping to define its total cost. In the following, we briefly describe them, and in Section IV-B, we will mention weighting parameters for each cost category used to calculate total weighted costs. E.g., the energy consumption can be weighted differently depending on saving goals, energy prices, etc.

1) Energy consumption: In order to calculate the energy consumption (kWh) of the total infrastructure, we first estimate the power consumption for each server, on average. We use a simple linear approximation studied by Fan et al. [6] .

$$P = P(U_{CPU}) = P_{idle} + (P_{peak} - P_{idle}) \cdot U_{CPU} \qquad (1)$$

Based on Equation 1, we use each server's average CPU utilization in an interval and the interval length to estimate the total infrastructure energy consumption in kWh. We have measured P_{idle} and P_{peak} for each server we are using in our simulation employing a high precision wattmeter (Hioki 3334) with a sampling rate of over 70 kHz. Additionally, we have measured $P_{standby}$, the standby power consumption of a currently switched off server.

2) Live Migration: The number of VM live migrations is contributing to the total costs. Live migration is a resource intensive task, so the energy consumption of the servers involved will be temporarily higher. Additionally, the service running in the VM may exhibit, e.g., worse response times during a live migration [8] .

3) Server Boots and Shutdowns: The number of server boots and shutdowns are also part of the total costs. Booting or shutting down a server takes time and consumes additional energy. Further, it is a phase of heavy strain on mechanical and electronical parts.

4) Utilization Thresholds: In constrast to number-crunching infrastructures like HPC clusters, it is not possible to fully utilize servers with VMs. If they are sheltering web, database, mail etc. services, massive queueing issues will arise, leading to catastrophic quality of service. Additionally, servers need to have resource reserves for short time peaks where live migration is not an option. Further, resource reserves are required for the tedious process of live migration. Hence, we use configurable utilization thresholds for both CPU and memory resource of each server: A desired level of utilization, a warning level, and an alarm level. If at least one resource component has reached the alarm level, the server is regarded as overloaded, if a warning level was reached, it is regarded as highly loaded.

5) Utilization Variance: The variance of server utilization is considered to penalize an imbalance of utilization across servers. This cost category is especially important for the GA presented and evaluated in this work. We use a fitness function, reciprocal to the cost function, to evaluate population individuals representing possible solutions. Then, penalizing utilization imbalance with a heavy weight leads to a preference of solutions that balance the workload as much as possible across all available servers. Hence, the GA will effectively do load balancing using the whole existing infrastructure. Load balancing is perhaps the most interesting scenario for comparison with the approach of this work, as it means to always use all existing resources instead to always use only the estimated required resources.

B. Parameters

In Section V, we will present results for simulation runs with different parameter values. We briefly explain these parameters here.

As mentioned in Section IV-A, we employ a cost function with configurable weights for each category of cost. We denote these weights as *penalties*. The heuristic BFF mapping is evaluated using these penalties, the genetic mapping algorithm additionally tries to find solutions by minimizing the total weighted cost C_{total}, which is simply calculated by

$$C_{total} = \sum_{i=0}^{n-1} c_i \cdot p_i, \tag{2}$$

where c_i is the raw cost value in category i, e.g., the amount of kWh consumed, and p_i the penalty for category i. Although the penalties are basically dimensionless, for most categories it is reasonable to think of them as amount of money, e.g., Euro cents, representing the cost of an action or state. E.g., for each kWh consumed by the server infrastructure, an energy penalty of 60 could represent a price of 30 cents for a kWh, and a power usage effectiveness (PUE) factor of two.

Important penalties opposing these energy costs are related to the utilization thresholds mentioned above. For each VM hosted on a server exceeding a utilization threshold, either the warning level or the alarm level, the respective penalty applies. The scenario is that the infrastructure is used to earn money, either directly by an online shop, or indirectly by ads. If the attempt to save energy was too aggressive or a forecast went wrong, visitors may abstain from placing an order as they experience a bad QoS. Additionally, we consider the amount of time, here the interval length, a VM was hosted on a server exceeding a utilization threshold. We do not know the exact amount of time a VM was hosted on an overloaded server, as the interval length determines the granularity of information. We simply treat the respective server as having exceeded thresholds for the whole interval length, and for each VM hosted on it, the penalty applies for each minute of the interval. The idea is to penalize exceeded thresholds in configurations with higher interval length stronger. Reasons for this are: a) it takes longer then to detect them and b) it is more probable to be unable to even detect them, when it is averaged out over a long period of time.

For the GA, penalizing utilization variance is especially interesting. We calculate it by adding the CPU utilization variance and the memory utilization variance of all existing servers in the pool. Multiplying the resulting value with configurable penalty and adding it to the total costs allows us to evaluate different scenarios. In the extreme case, with a high variance penalty, the GA effectively tries to do load balancing among all existing servers. Depending on the live migration penalty, in this case adversary to the variance penalty, the load balancing will be carried out more or less aggressively. For lower variance penalties, the consolidation of servers will be done less aggressively, as the variance penalty is then adversary to the energy penalty. Furthermore, each live migration, server boot and shutdown action is contributing by configurable penalties to the total costs. These state changes represent the necessary overhead to establish a target state.

C. Forecasting

We have evaluated two forecasting methods, the seasonal Holt-Winters method (triple exponential smoothing considering seasonal patterns) and auto-regressive moving-average methods. For both, we use the implementations of `GNU R`. We generate a new model for each VM in each interval, to consider the obvious heterogeneity in the traces. For Holt-Winters, we additionally use the model selection abilities of `GNU R` to estimate the best values for the parameters α, β and γ parameters and the type of seasonality, which can be additive or multiplicative. For the class of auto-regressive moving-average methods, we also apply model selection, but here the number of auto-regressive, differencing and moving-average terms is selected, as well as (non-)stationary modelling. We initially planned to also use seasonal ARIMA, but since it dramatically increased the run time of our simulation, independently from model selection and especially for short interval lengths when using a reasonable amount of seasonal auto-regressive terms, we had to postpone it until future work.

D. VM Mapping

1) Balanced First Fit Heuristic: The first mapping algorithm we compare with load balancing is a greedy heuristic originally developed for bin-packing, knapsack problems and related tasks. There are several changes necessary to use such heuristics for mapping VMs to heterogeneous server, considering at least two resource components, CPU and memory. Probably most important, instead of the items (the VMs), we have to sort the bins (the servers) by some criteria to decide which of them are to be filled first. Further, we always want to map all VMs, so sorting VMs is only reasonable for e.g., balancing the resource utilization of the servers. Additionally, there is a crucial difference regarding multiple dimensions of items and bins here. In contrast to bin packing or multidimensional, multiple knapsack problems, items are not alignable edge by edge, but must be aligned corner by corner, best denoted by *vector packing*. As our goal is to save energy, we define an energy efficiency sorting critera by

$$e = \frac{cpuCapacity + memCapacity}{powerPeak}, \quad (3)$$

where higher values of e represent a higher efficiency. The necessity for such a sorting criteria is a specific problem of heuristics here. For each defintion of efficiency, we can easily find situations where another definition would be better. E.g., for phases of low workload, a defintion using the idle power instead the peak power would be better. The general problem is that the efficiency of a server is heavily dependent on its utilization, but its utilization must here be determined by having an efficiency metric. Nevertheless, we plan to experiment with the metric proposed by Beloglazov et al. [2] for future work, although as mentioned above, we think it may still create solutions far from optimal in several situations.

We give a brief picture of its main concepts in the following. We start with the current mapping to minimize reconfiguration costs. Then, we split the problem into three sub-parts. First, all online servers are examined for utilization levels that are exceeding the warning threshold. If so, we remove VMs in a resource balanced way until the server is not exceeding any thresholds, and denote all removed VMs as *homeless*. Second, we try to map these VMs to other servers, beginning with the most energy efficient. Again, we try to add VMs in a way that balances CPU and memory utilization of a server. During phases of high workload, it is possible that not all VMs have found a new place, as all servers might have reached their desired level of utilization. Then, we force a mapping by ignoring the desired level, and just consider the warning level. Third, we try to consolidate the infrastructure. In a dry-run, we first check if *all* VMs of a host could be migrated to other, more energy-efficient hosts. Reasons for a negative results are e.g., that the more efficient servers do not have enough resources left. Additionally, we use a hysteresis parameter denoted by `vmConsolidationInertia`, which gives the amount of time a VM should stay on a server, unless it is not exceeding the warning thresholds. By adjusting the value, we can consolidate more or less aggressively. If all VMs could be placed on more efficient server, we implement the results from the dry-run.

2) Genetic Algorithm: The genetic algorithm is fundamentally different from the heuristic. Here, we can directly use the cost function defined in Section IV-A and configurable by the parameters described in Section IV-B to evaluate individual solutions of a population. To be specific, the reciprocal of the costs is used as *fitness* value. At the end of each generation, individuals are then selected to be part of the next generation depending on their fitness. We primarily use the *elitism selection* mode here, where the best n individuals are selected. Another option is the so-called *roulette wheel selection*, where individuals with higher fitness are selected with a probability proportional to their relative fitness. We have implemented a *single point crossover* operation. Here, two randomly selected parent individuals create two offsprings. One offspring gets all “genes“ left of the cutting point from one parent, and right from the cutting point from the other parent. For the other offspring, it is vice versa. Additionally, we use several *mutation* operations. The most important is perhaps the `migrateVm` mutation, where a VM is set to run on another server. Further, there is a `swapRm` mutation, where all VMs of two servers are exchanged, and a `swapVm` mutation, where two servers each migrate a VM between them. The process which servers and/or VMs to take as operands is random for all operations, which are not carried out if servers would be overloaded. With the parameters listed in Table I, we can influence the amount of these operations within each generation, the number of generations, and the population size.

V. Results

As already mentioned, the original trace data has an interval length of two hours, which is rather long. In a first evaluation, we have interpolated the trace data into resolutions of 3600 s, 1800 s, 900 s and 300 s, comparing them with the original resolution of 7200 s. The resulting total weighted costs for each interpolation, mapping decision module and optional forecasting method are shown in Figure 5, the parameters used to yield these results are listed in Table I. Here, we demonstrate the feasibility and flexibility of our cost model and the described genetic algorithm. It is easily adaptable to carry out load balancing by adjusting a single parameter. Regarding the chosen parameter values, we are examining a case study in the scope of this work, and primarily focus on investigating the influence of the interval length.

Relevance	Parameter	Value
All	cpuUtilizationDesiredLevel	0.6
	cpuUtilizationWarningLevel	0.7
	cpuUtilizationAlarmLevel	0.85
	memoryUtilizationDesiredLevel	0.7
	memoryUtilizationWarningLevel	0.8
	memoryUtilizationAlarmLevel	0.9
	energyPenalty	60
	migrationPenalty	1
	vmOnHighloadedRmPenalty	1
	vmOnOverloadedRmPenalty	10
	bootPenalty	1
	shutdownPenalty	1
Load Balancing	variancePenalty	100
BFF	rmIdleTimeoutSeconds	1
	vmConsolidationInertiaSeconds	500
GA and LB	numberOfGenerations	20
	sizeOfPopulation	1000
	crossoverRate	0.25
	migrateVmRate	0.2
	swapRmRate	0.03
	swapVmRate	0.03
	elitism	true
Optional Forecasting	requiredPeriodsForForecasting	3

TABLE I
PARAMETERS USED FOR YIELDING RESULTS SHOWN.

In Figure 5, the total weighted costs for each combination of interpolation, decision module and optional forecasting is shown. It is clearly visible that load balancing (LB) achieves best for all cases except when interpolating to an interval

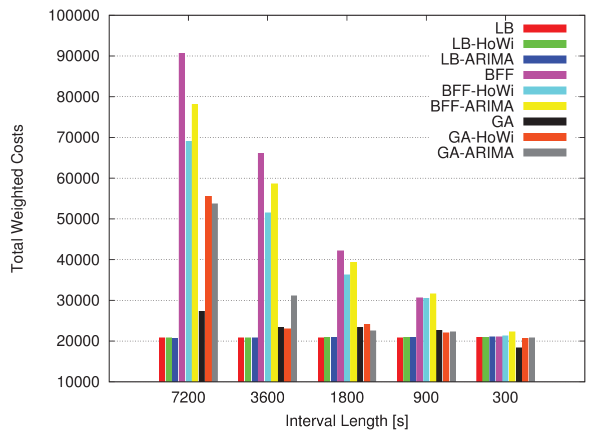


Fig. 5. Total weighted costs for different interpolations of the trace data, comparing load balancing (LB) to a heuristic bin packing (BFF) and genetic algorithm (GA) with optional Holt-Winter (HoWi) forecasting.

length of 300 s. The greedy heuristic adapted from bin packing (BFF, Balanced First Fit) benefits most from shorter intervals, achieving comparably to load balancing at 300 s. The genetic algorithm (GA) is in fact able to yield lower costs than load balancing at 300 s, and is always significantly better than BFF. Interestingly, forecasting generally lowers the cost of using BFF, but has no significant positive effect when using GA. The 7200 s interval is a special case, as GA is achieving significantly better without forecasting, although better than BFF with forecasting.

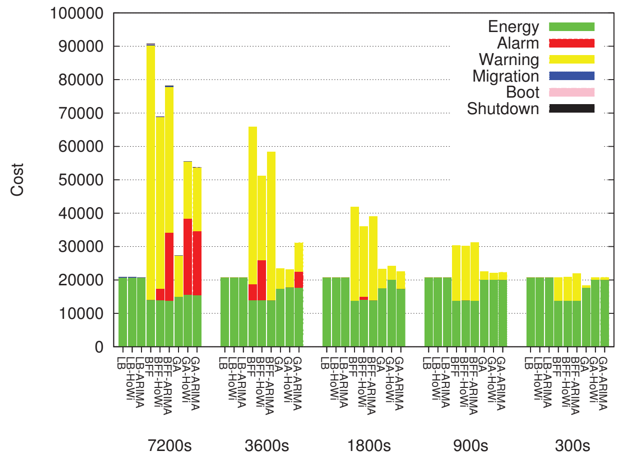


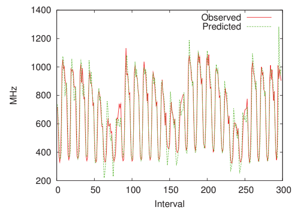
Fig. 6. The inner cost structure of the total costs shown in Figure 5. Depending on the scenario, the yellow parts representing utilization warnings can be included or excluded, leading to different conclusions.

In Figure 6, the costs of each category are shown and give a more detailled insight into the results. We can see with the parameters used here, the most important categories are energy, utilization warnings and in rare cases, utilization alarms. It is also clearly visible that energy costs are rather independent from the interval length, but warnings and alarms are indeed highly dependent. The role of the warning costs is ambiguous. If warning thresholds are deliberately set to values where QoS is still acceptable, but on the edge of becoming unacceptable, then exceeding the threshold is merely triggering preventive action (here, live migration) to stabilize QoS. Hence, the warning costs are simply information how often such actions had to be taken. Then, we can ignore the yellow part of each bar in Figure 6, leading to different conclusions. Given the specified parameters, BFF would achieve well in saving energy while avoiding unacceptable QoS, especially for shorter interval length. For 300 s and compared with load balancing (LB), a saving of approximately a third of total weighted costs is possible when using BFF.

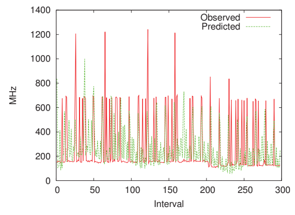
With warnings considered, reducing total weighted costs is only possible with 300 s interval and the genetic algorithm, achieving a reduction of approximately 10%. The ambiguity of the warning thresholds is another example for BFF's inflexibility. A genetic algorithm does not require discrete thresholds, but can use a continuous function defining the penalty of an arbitrary level of utilization. This function could have a slope beginning at a utilization of 50%. Tha GA would compare the cost resulting from having a utilization of over 50% with the costs resulting from taking action against it. The higher the utilization, the higher the probability of an outgoing live migration to remedy the situation, and the lower the probability the such a state is created by an incoming migration.

Furthermore, it is interesting how forecasting actually increases costs in some cases. One of this cases, the genetic algorithm at interval length 7200 s, is shown in Figure 7 using a set of representative examples. In Figure 7(a), we can see how Holt-Winters generally produces good forecasts when the utilization trace is highly periodic. At weekends, it is underestimating the occuring workload, but otherwise, it performs well. However, this is not true for all traces. Figure 7(b) shows how Holt-Winters is clearly unable to predict peaky workloads. There are even more subtle cases, as shown in Figure 7(c), where the trace is changing from a periodic pattern to what seems to be idleness. E.g., the respective VM's service may have been migrated to another VM due to a switch in the used software stack. We can clearly see how Holt-Winters is oscillating around the true, observed values for weeks, underestimating the workload intensity for long periods of time. On the other hand, ARIMA, as shown in Figure 7(d), does not oscillate. Nevertheless, it approximates very slowly to the true value, overestimating the workload during this process. Figure 7(a) and Figure 7(b) are also valid to give a qualitative impression of ARIMA for these traces, as it is performing similarly here.

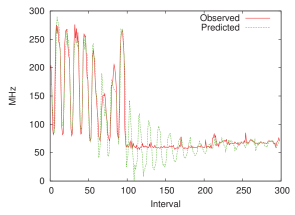
There are various possible consequences of these forecasting complications. We could limit the window of observations used to create forecasting models to n samples. However, choosing a good value of n is difficult. It is important to quickly react to changing patterns while still having enough samples to calculate periodic behaviour. More importantly, it would not be a remedy to the case shown in Figure 7(b). Of course, completely random peaks are impossible to pre-



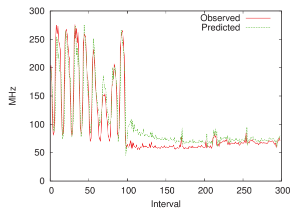
(a) HoltWinters, trace #30



(b) HoltWinters, trace #2



(c) HoltWinters, trace #8



(d) ARIMA, trace #8

Fig. 7. (a) Holt-Winters working well with a periodic trace, (b) Holt-Winters unable to predict peaks, (c) Holt-Winters oscillating after change in trace pattern, (d) ARIMA fading out after change in trace pattern.

dict. They have to be absorbed by server resource reserves. Nevertheless, there are cases of multiple samples of high utilization in straight succession. They are unperceived by forecasting, which is still giving far too low predictions. Here, single exponential smoothing would most likely give better prediction. The start of a burst is hard to predict anyway, but the rest of a burst may be predicted acceptably well, and when the burst ends, forecasting would overestimate just once, and especially not underestimate. As a consequence for future work, we regard it as more appropriate to choose forecasting methods for each VM specifically, and to reexamine the choice after each interval. If the residual between forecast and observation was too big, a safe fallback, e.g., single exponential smoothing should be used until a periodic pattern is observable again.

VI. Conclusions and Future Work

In this work, we have presented results how dynamic consolidation and workload forecasting can be combined to reduce energy consumption of virtualized data centers, based on real utilization traces. We have investigated a business infrastructure scenario, where energy costs are just one of several parts of operational costs. We have presented a cost model feeding into the GA's fitness function. It allows easy adaption by infrastructure operators to specific optimization demands. We have used a case study set of parameter values to compare the total weighted costs of load balancing with a greedy heuristic and a genetic algorithm, combined with Holt-Winters and ARIMA forecasting. As a result, the total operational costs of an infrastructure can be reduced by 10% to 30%, depending on the configuration. For long interval lengths, forecasting can help decreasing the costs of using mapping heuristics. We have shown the flexibility of the genetic algorithm, which can be turned from dynamic consolidation to load balancing by simply penalizing utilization variance. Further, it is easily adaptable to a energy prices, live migration overheads etc.

For future work, we plan several extensions to the status quo. We want to analyse more and longer traces, as we expect smoother state changes for bigger infrastructures. Forecasting will be a major part of future work, as there is much room for improvement. Further, the ambiguity of the utilization thresholds will be addressed in future work for the genetic algorithm, where a continuous function defining the penalty of an arbitrary level of utilization is easily usable instead. A similar approach could be used for live migration, as we have shown in [8] that the consequences of live migration regarding QoS are heavily depending on the utilization of the VM while it is being migrated. Finally, an important topic for future work will be to evaluate different cost scenarios. Possible extensions to the cost function are heterogeneous revenue losses for VMs and thus implementing priorities, or heavily penalizing the co-existance of two specific VMs on a server, thus implementing that some VMs should never run on the same host due to performance reasons, customer isolation, etc.

VII. Acknowledgements

We thank Ralph Staudigl and Ulf Busch of University of Vienna's Central IT department for their efforts to make this research possible by allowing us to examine real VM utilization traces.

References

[1] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *Computer*, 40:33–37, December 2007.

[2] A. Beloglazov and R. Buyya. Energy efficient allocation of virtual machines in cloud data centers. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 577–578. Ieee, 2010.

[3] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. In *Integrated Network Management, 2007*.

[4] K. Brill. The invisible crisis in the data center: The economic meltdown of moores law. *white paper, Uptime Institute*, 2007.

[5] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[6] X. Fan, W. Weber, and L. Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th annual international symposium on Computer architecture*, pages 13–23. ACM, 2007.

[7] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall. Entropy: a consolidation manager for clusters. In *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 41–50, New York, NY, USA, 2009. ACM.

[8] H. Hlavacs and T. Treutner. Predicting web service levels during vm live migrations. In *Systems and Virtualization Management (SVM), 2011 5th International DMTF Academic Alliance Workshop on*, pages 1–10. IEEE, 2011.

[9] C. Hyser, B. McKee, R. Gardner, and B. Watson. Autonomic virtual machine placement in the data center.

[10] A. Verma, P. Ahuja, and A. Neogi. pmapper: power and migration cost aware application placement in virtualized systems. In *Middleware '08: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pages 243–264, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

[11] W. Vogels. Beyond server consolidation. *Queue*, 6(1):20–26, Jan. 2008.

[12] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Black-box and gray-box strategies for virtual machine migration. In *Proc. NSDI*, volume 7, pages 11–13, 2007.