2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing

Distributed Resource Allocation to Virtual Machines via Artificial Neural Networks

Dorian Minarolli and Bernd Freisleben
Department of Mathematics and Computer Science, University of Marburg
Hans-Meerwein-Str. 3, D-35032 Marburg, Germany
{minarolli, freisleb}@informatik.uni-marburg.de

***Abstract*—The goal of the provider with respect to dynamic resource allocation in cloud computing is to maintain application performance according to service level agreements while reducing electrical power costs. To achieve this goal, we present a resource manager that optimizes a utility function expressing the trade-off between the conflicting objectives of maintaining application performance and reducing power costs. It is based on an artificial neural network (ANN) to find the best resource allocation to virtual machines that optimizes the utility function. To provide support for a potentially large number of virtual machines, we present a distributed version of the resource manager consisting of several ANNs in which each ANN is responsible for modeling application performance and power consumption of a single VM while exchanging information with other ANNs to coordinate resource allocation. Simulated and real experiments show the effectiveness of the distributed ANN resource manager over static allocation, a centralized version and a distributed non-coordinated version.**

***Keywords*-cloud computing; utility function; artificial neural network; resource allocation;**

I. Introduction

Infrastructure-as-a-Service (IaaS) clouds provide resources (e.g., CPU and storage) as services that can be leased and released by users via the Internet in an on-demand fashion [29]. One of the most promising features of IaaS clouds is dynamic allocation of resources to virtual machines executing users' applications according to workload changes. This is achieved by virtualization technologies such as Xen [3] or VMware [25] that provide resource provisioning to virtual machines (VMs) through three main mechanisms: a) horizontal scaling – resource allocation is achieved by adding or removing VMs; b) vertical scaling – resource allocation is achieved by changing resource shares given to VMs; and c) live migration – resource allocation is achieved by moving a VM to another physical machine. The important issue for a cloud provider is how to use the above mechanisms in order to maintain application performance according to service level agreements (SLAs) while reducing power consumption to minimize operating costs.

In this paper, we focus on vertical resource scaling and present an approach to assist the Virtual Machine Monitor (VMM) of an IaaS cloud to optimize resource allocation to VMs running on the physical machines. The proposed approach addresses the mentioned performance-power trade-off by expressing the two conflicting objectives of application performance and power consumption in a utility function and optimizes the utility function at runtime. We present a resource manager for utility function optimization that is based on an artificial neural network (ANN) to model the relationship between the resources allocated to VMs, the performance of applications and the power consumption of physical machine. ANNs are well known for being universal approximators [12] to model any complex non-linear function. Another feature of our approach is that the model is learned and adapted on-line to workload changes, avoiding the need to build the model beforehand.

There are several versions of the proposed resource manager. In our centralized resource manager, we use a single Multiple-Input Multiple-Output (MIMO) ANN based model for capturing the relationship between resource allocation to all VMs running on a physical machine and their corresponding performance metrics. Another similar MIMO model is used to capture the relationship between VM resource allocation and power consumption of the physical machine. To provide support for a potentially large number of VMs, we present a distributed resource manager where the single MIMO model is divided into several models, each of them being responsible for performance and power modeling of a single VM. Each of the distributed resource managers optimizes its own utility function through its ANN model, but exchanges information with other resource managers to coordinate resource allocations. We present an evaluation of the approach via simulation and real experiments. The results show the effectiveness of the decentralized resource manager over static allocation, a centralized version and a distributed non-coordinated version.

The paper is organized as follows. Section II discusses related work. Section III describes the resource manager architecture, the utility function and the power model. Section IV introduces the centralized resource manager approach. Section V presents its distributed version. Section VI discusses implementation issues. Section VII describes experimental results. Section VIII concludes the paper and outlines areas for future work.

II. Related Work

The proposals made in the literature for dynamic resource management in non-virtualized and virtualized environments address different aspects of resource and performance management. We focus on proposals that are directly related to our work and group them into three categories: a) model-based utility optimization, b) feed-back control theory and c) machine learning approaches.

Model-based Utility Optimization. There are several proposals [4], [7], [14], [17] managing resources by optimizing a utility function. They are based on queuing models for predicting resource allocations to achieve the optimal utility value. The problem with these approaches is the difficulty to build accurate analytical models to predict the behavior of applications in virtualized environments. They are also specific to certain multi-tier web applications. In contrast, our approach builds a general model during runtime without any prior knowledge using ANNs that can capture the complex behavior of applications.

Feed-back Control Theory. Other proposals [2], [15], [18], [26] use feed-back control theory for resource and performance management. Their objective is to keep particular performance metrics to a given reference level even in the presence of disturbances. This is achieved by using a control law typically derived from a linear system model identification process. Although they offer stability guarantees, they are based on linear models that cannot capture complex non-linear application behaviors and interferences between applications running on shared virtualized infrastructures. Our approach has the potential to model complex non-linear application behaviors thanks to the universal approximation capability of ANNs.

Machine Learning Approaches. These proposals are based on machine learning techniques to model and manage application performance. For example, Wildstrom *et al.* [27] use M5 trees to learn a model for predicting whether the revenue of a cloud provider will increase by changing the memory allocation. In the approach presented by Bodik *et al.* [6], a machine learning technique is used to learn a performance model. Kundu *et al.* [13] use ANN to model application performance in virtualized infrastructures using multiple resource knobs. Wildstrom *et al.* [28] also use machine learning for modeling performance based on low level system metrics to find the best configuration to increase the throughput. The difference of our approach to these proposals is that they train a single model offline for all workload and resource allocation combinations, while we train the model online. Bitirgen *et al.* [5] use ANNs to predict performance and support online model training, but they consider resource allocation at the multiprocessor chip level, while we manage resources at the CPU and memory level in a virtual environment. Our approach finds a suitable performance-power trade-off through utility function optimization and supports a potentially large number of VMs by a distributed resource manager.

Other approaches [19], [22], [23] use reinforcement learning for resource allocation. In this technique, the resource manager learns the best resource allocation configuration by trying different actions in the environment and taking the ones that increase long term utility. The problem with this approach is the long learning time for finding the optimal policy, making it impractical in production environments.

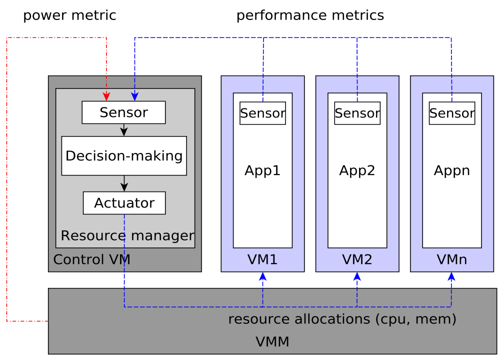


Figure 1: Resource manager architecture

III. Resource Manager Architecture

The proposed resource manager architecture is shown in Fig. 1. It consists of three components: sensor, actuator and decision-making module. It works in discrete time intervals where at the end of each time interval it gathers application performance and power metrics through the sensor module, performs resource allocation to all VMs for the next interval through the decision-making module and allocates the resources through the actuator.

The problem addressed is how to allocate resources to VMs running on each physical machine of the cloud provider in order to maintain application performance according to SLA levels while reducing power consumption. To solve this performance-power trade-off, we express both objectives in a utility function and based on ANN modeling find the resource allocation that optimizes this utility function. The utility function used is shown in Equation (1):

$$U = \delta * (\alpha \frac{\sum_{i=1}^{n} V_i(perf_i)}{n} - \beta U_p(power)) \qquad (1)$$

U is the average global utility over all VMs representing the average profit the cloud provider gets from one physical machine during a time interval, n is number of VMs, V_i is the performance utility that represents the profit the provider gets from the consumer for guaranteeing a certain performance level to VM_i during a time interval, and U_p is a power utility function that represents the power consumption costs of the physical machine in one time interval. The coefficients α and β are used to control the priority given to both objectives. δ is a constant to make the utility value larger than 1 for display convenience. The performance utility function V_i used in our study is given as a function of normalized application performance $perf_i$, as shown in Equation (2). Normalized performance is used to make the resource manager independent of specific application performance metrics. Since we use the throughput as our performance metric (the number of operations served per second), the normalized performance is the ratio of the actual throughput to the desired throughput stated in an SLA.

$$V_i(perf_i) = \begin{cases} perf_i & \text{if } perf_i < 1, \\ 1 & \text{if } perf_i >= 1. \end{cases} \tag{2}$$

The power utility function $U_p(power)$ is a function of the $power$ consumed by a physical machine in a time interval. Since power consumption of a physical machine generally depends on resource utilization, we adopt a linear model for CPU and disk I/O utilization [9] to approximate power consumption as shown below:

$$power = P_{idle} + P_{cpu}\frac{U_{cpu}}{C_{cpu}} + P_{disk}\frac{U_{disk}}{C_{disk}} \tag{3}$$

P_{cpu} is the maximum dynamic power consumption of the CPU at full utilization, U_{cpu} is the CPU utilization as a percentage of the total CPU capacity, C_{cpu} is the total CPU capacity, P_{disk} is the maximum dynamic power consumption of the disk at full bandwidth utilization, U_{disk} is the disk utilization as a percentage of the total disk bandwidth capacity, and C_{disk} is the total disk bandwidth capacity. P_{idle} is the power consumed by a physical machine when it is idle. These metrics are average values measured in a single time interval. We use a power utility function that is given as the ratio of actual dynamic power consumption to the maximal dynamic power at full utilization:

$$U_p(power) = \frac{power - P_{idle}}{P_{cpu} + P_{disk}} \tag{4}$$

IV. Centralized Resource Manager

The architecture of our centralized resource manager approach is shown in Fig. 2. It consists of three components: 1) a $performance\ model$ representing the relationship between VM resource allocation and application performance metrics, 2) a $power\ model$ representing the relationship between VM resource allocation and physical machine power consumption 3) a $utility\ optimizer$ that uses the above models to search for resource allocations that reach maximum utility.

1) Performance Model: In the centralized resource manager, a MIMO model of the relationship between VM resource allocation and application performance is used. For two VMs, its structure is shown in Fig. 2. It has four inputs representing resource allocations, two inputs (CPU and memory) for each VM and two outputs representing performance metrics, one output for each VM. The model predicts the performance of each VM if certain resource allocations are given to all VMs. To build the model we used an ANN to approximate the possibly non-linear function of the mentioned multi-input multi-output parameters.

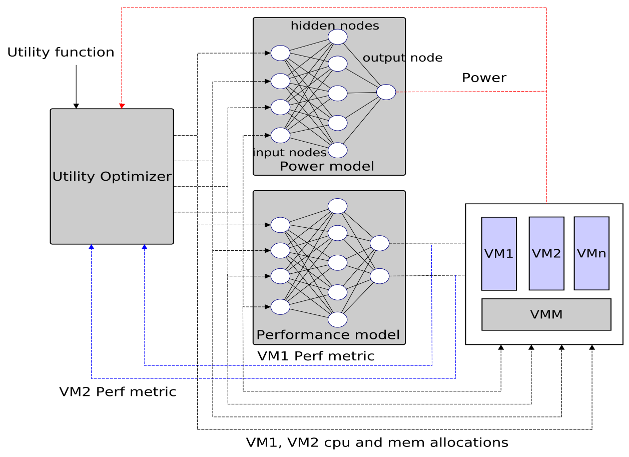
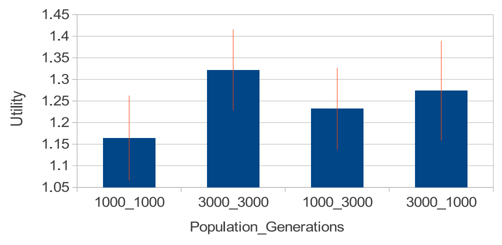


Figure 2: Centralized ANN resource manager

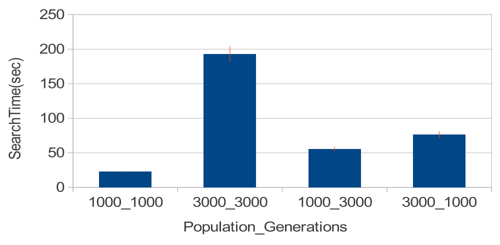
An ANN is a structure of multiple layers of nodes connected through links, as shown in Fig. 2. It has three types of nodes: input, hidden and output nodes. Each link passes the output value of a node to the input of another node. The output of a node is calculated as the weighted sum of all inputs to the node which is then fed into a (non-linear) activation function. The ANN learns a model in a training phase where it is exposed to a number of (input, output) samples from a training set. Since it is difficult to decide which training algorithm performs best for which task, we use one of the well known training algorithms for ANNs, namely the resilient backpropagation (RPROP) algorithm [20]. To build the ANN, we also use the cascade correlation algorithm [8] that starts with a minimal network, then automatically adds new hidden nodes one by one during training, creating a structure with the number of hidden nodes necessary for the current training set size.

2) Power Model: We have built a MIMO model of the relationship between VM resource allocation and physical machine power consumption. For two VMs, its structure is shown in Fig. 2. It has four inputs representing resource allocations, two inputs (CPU and memory) for each VM and one output representing the power. The model predicts the power consumption of the physical machine if certain resource allocations are given to all VMs. To model power consumption, we also used an ANN.

3) Utility Optimizer: The goal of the utility optimizer is to find the resource allocation configuration that maximizes the utility function given in Equation (1). It does this by trying all resource configurations, predicting the performance using the performance model and power consumption using the power model, calculating the resulting utility and selecting the configuration that gives the maximum utility value. The total number of configuration combinations to try depends on the granularity of the resource allocations (minimum allocation change from one configuration to another)



(a) Utility for combination of population_generations



(b) Search time for combination of population_generations

Figure 3: Utility and search time for different combinations of population size and number of generations

and the number of VMs. For a larger number of VMs, the total number of combinations to try can require considerable time, making it impractical for an exhaustive search, and in this case a stochastic search technique, namely a genetic algorithm [11] is used to optimize the utility function.

A genetic algorithm is based on natural evolution for exploring a search space to find the solution of a problem. It starts with a random initial population of individuals where each individual represents a feasible solution to the problem. There is a function that evaluates the fitness of an individual; the utility function is used as our fitness function. Then, from an initial population based on selection, crossover and mutation, a new population is generated that is supposed to be better than the old one in terms of the fitness function. This process is repeated for several generations until a maximum generation number is reached and individual with the best fitness found so far is the final solution of the problem. Each individual encodes information that represents a feasible solution. In our case, this is a resource allocation configuration in the form of a string of real numbers that represents resource shares given to VMs. The selection process selects from the population two parent individuals based on their fitness values according to a roulette wheel scheme. Then, with some probability, a crossover operator is applied where different parts of parents are combined to create two children such that each child takes a CPU allocation configuration from one parent and a memory allocation configuration from the other parent. At the end, a mutation operation is applied where with a small probability each gene of the child is altered to a random value. This process of selection-crossover-mutation is repeated to create the number of children needed for the new population to replace the old one for the next generation.

Since population size and number of generations are two important parameters that can influence the genetic algorithm performance, we have performed experiments to determine their influence on our utility optimization problem. In these experiments, we have run 16 VMs, managed by a centralized ANN manager, for 100 control intervals, and the average utility value over all intervals has been measured. We give each parameter a low and high value of 1000 and 3000, respectively, creating four combinations. We did not test values lower than 1000, since they already have shown acceptable search times. In Fig. 3, the average utility and search time in seconds of five runs are shown for the four combinations X_Y where X represents population size and Y the number of generations. Increasing any of the parameters to higher than 1000 does not have any effect on the average utility value, since their differences are statistically insignificant, as shown from ANOVA statistical test. Since increasing each of the above parameters affects search time, we use the combination 1000_1000, which has the lowest search time for our utility optimization problem.

Algorithm 1: Resource manager algorithm

```
1  apply few predefined samples to build an initial model
2  while true do
3      based on models find the optimal allocation
4      for $j \leftarrow 1$ to $N$ do
5          broadcast(resource allocation of the VM)
6          receive(resource allocation of other VMs)
7          based on models find the optimal allocation
8      end
9      apply optimal allocation and sleep(S seconds)
10     get performance and power metrics
11     broadcast(resource allocation of VM)
12     receive(resource allocation of other VMs)
13     calculate_max_resource_capacity_every_M_intervals
14     add the new sample to the training set
15     train perf and power model with new training set
16 end
```

The steps followed by the centralized ANN resource manager are given in Algorithm 1 without lines 4 to 8 and 11 to 13 which are used only in the distributed version. It starts by applying a fixed number of samples (configurations) to build initial models of performance and power. The initial training set is constructed by generating random samples to

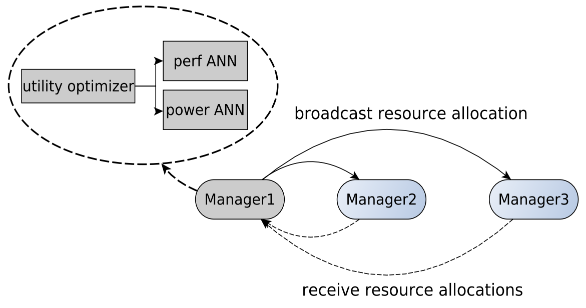


Figure 4: Distributed resource manager

cover the whole range of resource allocations that can be given to VMs. Then, based on the built models, it finds an optimal configuration that is applied to the system. After a time interval of S seconds (S=30 in our case) set by sleep, it gets performance and power metrics that constitute a new sample that is added to the training set. Then, the models are trained with the new training set, and this process is repeated. This repeated training makes it possible to adapt the models to workload changes. Since the training set/time grows by adding new samples, we fixed the training set to 50 samples, as a trade-off between prediction accuracy and training time, and adding a new sample means removing the oldest one.

V. Distributed Resource Manager

In the distributed version, the resource manager is divided into several resource managers, each responsible for resource allocation of a single VM. Its architecture for three VMs is shown in Fig. 4. Each manager builds an ANN based performance model of its own VM that has only two inputs (CPU and memory allocation of one VM). It also builds an ANN based power model for predicting the power consumption of the physical machine. Since the power consumption of the physical machine depends on resource allocation to all VMs, the number of inputs to the power model is two times the number of VMs. In the distributed resource manager, the global utility given in Equation (1) is divided into local utility functions, given in Equation (5), one for each resource manager. This function depends on the performance of the corresponding VM and power consumption of the physical machine. Based on the performance and power models, the utility optimizer finds resource allocations given to the VM to optimize the local utility function.

$$U_i = \alpha V_i(perf_i) - \beta U_p(power) \tag{5}$$

Each resource manager controls the resource allocation of only one VM, therefore the number of resource allocation combinations to try for local utility optimization can be performed in negligible time. On the other hand, since the power consumption depends on the resource allocation to all VMs, each resource manager exchanges resource allocation decisions with other resource managers through a coordination procedure organized in a number of iterations where in each iteration a search for different resource allocation combinations to find the optimal utility is made. For each resource allocation combination, the resource manager uses ANN models to predict the performance of the VM and the power consumption of the physical machine. To predict power consumption, the resource manager fixes the resource allocations to other VMs to the values decided by other managers in the previous iteration (in the beginning, it sets them to some random values), while changing the resource allocations of its VM. After this search, it selects the resource combination that has the maximum local utility value. At the end of the iteration, the resource manager exchanges this allocation decision with other managers to be used for the next iteration. This process of search and exchange is repeated for a number of iterations N as shown in lines 4 to 8 of Algorithm 1. The intuition behind this iterative process is that after predicting the power consumption for the current iteration, the resource allocation decision of other VMs may have changed, and in the next iteration after getting the new resource allocations, an improved prediction can be made. At the end of N iterations, the last predicted resource allocation that has the maximal utility is applied to the system.

The steps followed by each resource manager are similar to the centralized resource manager shown in Algorithm 1, but they are used for allocating resources to only one VM and there is the addition of lines 4 to 8 and 11 to 13. In order for the resource manager to train the power model, it gets the final resource allocation decisions of other resource managers corresponding to the measured power metric by exchanging allocation information with other managers through lines 11 and 12 in Algorithm 1.

Since resource managers act independently, they can allocate resources to VMs that overpass the maximum physical machine resource capacity. To overcome this problem, a maximum resource capacity is assigned to each resource manager. The method, which is shown in line 13 of Algorithm 1, to calculate the maximum resource capacity is executed by each resource manager every M time intervals ($M = 5$ in our case). Through the broadcast and receive operations in lines 11 and 12 of Algorithm 1, apart from resource allocations in the current time interval, resource managers exchange resource consumption information. By resource consumption we mean an exponentially weighted moving average of resource allocations of several time intervals in the past. After having obtained resource consumptions of all VMs, the method proceeds as follows. In the beginning, each resource manager has the same maximum resource capacity. Then, at the time of calculation, the free resource amount (i.e., the difference between the maximum capacity and the resource consumption) is computed for each VM and also the total free resource amount. This total free

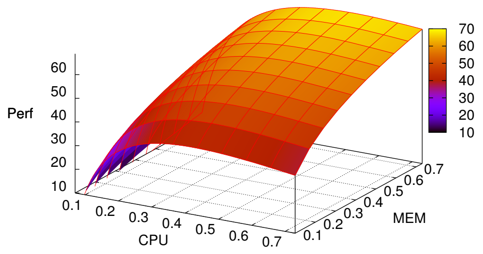


Figure 5: Web performance vs. resource allocation

resource amount is divided by the number of VMs to obtain the new free resource amount for each VM. By adding this new free resource amount to the VM resource consumption, the new maximum resource capacity is obtained for each VM. This method is executed by each resource manager to compute its own maximum resource capacity and also the capacity of others needed for the next calculation.

VI. Implementation Issues

We have implemented a simulated environment in the C++ programming language and simulated two type of applications. One is a High Performance Computing (HPC) application running CPU intensive calculations. Its performance, measured as the number of calculations per second, depends linearly on the CPU share allocated to the virtual machine. The other is a web application that has file I/O intensive workload serving web data from the disk. Its performance metric, the throughput, depends on both CPU and memory allocation to the VM. The relationship between web performance and resource allocation of CPU and memory used in our simulations is shown in Fig. 5. By increasing the resource allocation, the performance is increased until some point is reached where it is saturated, showing bimodal behavior that is typical for this kind of application [26]. For the web application, we simulated two types of workloads called workload1 and workload2, that have the general form of Fig. 5, but differ with respect to the slope of the performance surface and the load intensity. For both applications, we also simulated CPU and disk I/O utilizations our power model depends on. CPU utilization of the HCP application is linearly depending on the CPU allocation, while for the web application it shows the same bimodal behavior as in the case of performance. Disk I/O utilization is high with low memory allocation and is low with high memory allocation. To the performance metric and the resource utilization generated from the models we added some small random number generated from a normal distribution with zero mean to simulate measurement noise and uncertainty present in real environments.

We also performed experiments in a realistic IaaS cloud environment based on the Xen virtualization technology. To measure application performance, we used the application heartbeat framework [10] that allows application code to emit heartbeats. The number of heartbeats per second represents application performance. The application performance sensors shown in Fig. 1 run as daemons inside each VM and are attached to an application through a shared memory mechanisms to gather performance metrics using the heartbeat framework. The main sensor that is part of the resource manager requests performance metrics through the IP socket interface from all application sensors in each control interval. CPU and disk utilizations are gathered using xentop and iostat commands in Xen environment. The actuator sets the CPU and memory allocations using the xm sched-credit and xm mem-set commands of the Xen hypervisor.

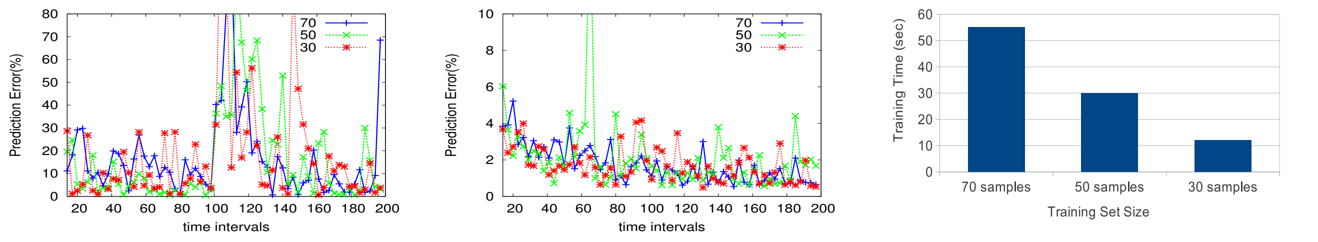
We have implemented our approach using the Fast Artificial Neural Network library (FANN) [21] that offers multilayer artificial neural networks in the C programming language. The genetic algorithm has been implemented using the GAlib [16] C++ library. We have set both coefficients α and β in Equation (1) to value 1. We have set the following genetic algorithm parameters: crossover probability to 0.85 and mutation probability to 0.01. We have implemented the distributed resource manager as a multi-threaded application where each manager is run by one thread in parallel with other managers. After testing several values, the parameter N of Algorithm 1 has been set to the value of 10, since increasing it further did not lead to further improvements.

VII. Experimental Results

A. Experimental Setup

Our simulation experiments are based on a simulated two quad-core CPU machine with a total of 800% CPU capacity and 16 GB of memory. The maximum power of the physical machine at full utilization has been set to 288 Watt, the power at idle state to 156 Watt and the maximum dynamic power at full utilization to 132 Watt. Since the CPU is the biggest power consumer, we have set its dynamic power at full utilization to 106 Watt and the dynamic power of the disk at full utilization to 26 Watt. These are typical values found in real servers [24].

For our experiments in a realistic IaaS environment, we have set up a physical machine that has two dual-core AMD Opteron 2.4 GHz processors and 8 GB of RAM. It runs Xen 4.1 and the Ubuntu 12.04 operating system in the Dom0 and DomU VMs. We have used the following benchmarks: a) cpu_bench is a CPU intensive benchmark created by ourselves that performs mathematical calculations such as computing the factorial of numbers, b) filebench [1] is a file system benchmark that emulates a number of applications such as web and file servers. We have set maximum power of the physical machine at full utilization to 212 Watt, the power at idle state to 132 Watt, the maximal dynamic power



(a) Prediction error of performance model (b) Prediction error of power model (c) Training time of performance and power models

Figure 6: Prediction error and training time for different training set sizes

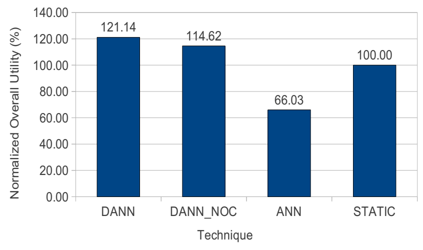
at full utilization to 80 Watt, the CPU dynamic power at full utilization to 56 Watt and the dynamic power of the disk at full utilization to 24 Watt. These values have been derived from the power values of a simulated machine, taking into account the fact that the physical machine has half of the CPU cores of the simulated machine.

We performed simulations with 4, 8 and 16 VMs where in each case half of them run the HPC application and the other half runs the web application. We restricted the simulations to 16 VMs, since we focus on a single physical machine, and running 16 VMs is reasonable and sufficient to study the behavior of the different resource allocation approaches when the number of VMs is increased. For 4 VMs, we assigned a maximum of 2 CPU cores to all VMs, for 8 VMs we assigned 4 CPU cores and for 16 VMs we assigned 8 CPU cores. For simplicity, we have not simulated the control VM and its power consumption, but this has been considered in the experiments in a realistic IaaS environment. For each of the cases above, we have considered two workload mixes: a) a steady mix and b) a periodic mix. In the steady workload mix, the experiment is run for 200 time intervals where from interval 0 to 100 all web applications run workload1 and from 100 to 200 they run workload2. In the periodic workload mix, the web application workload changes between workload1 and workload2 every 24 intervals. We experimented with 4 resource allocation techniques abbreviated as follows: ANN is the centralized ANN resource manager, DANN is the distributed ANN resource manager, STATIC is an allocation where the total resource capacity is distributed equally to all VMs, and DANNNOC is a distributed ANN resource allocation technique without coordination between the ANNs where the power model is based only on resource allocation to the corresponding VM. For each combination of the allocation techniques, the number of VMs and the workload mixes, we repeated the experiment five times and the collected results were exposed to ANOVA and TukeyHSD statistical tests.

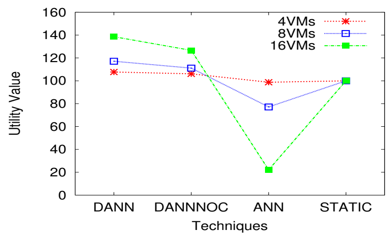
B. Simulation Experiment: Results

1) ANN Model Prediction Accuracy and Training Time: First, we performed simulation experiments to determine the size of the training and the prediction accuracy of the model. In this experiment, 16 VMs are executed with a steady workload mix in a single simulated machine. In Fig. 6(a), the average performance prediction error over all VMs performance models for DANN is shown. We tested three different training set sizes of 70, 50 and 30 samples. For all training set sizes, the average prediction error, excluding the time of adaptation to new workload from 100 to 130, is less then 12%. On the average, a slight decrease of the prediction error is evident when the training set size is increased from 30 to 50 and to 70. Since the training time increases when the training set size is increased, as shown in Fig. 6(c), a training set size of 50 samples has been chosen as a trade-off for the rest of the experiments. In Fig. 6(a), we can observe the time needed to achieve an acceptable prediction error of the performance model after a workload change in time interval 100: it is around 30 time intervals. The power prediction error is not affected by the training set size, and all sizes achieve an error of less than 3%. This is because the dynamic power change is small compared to the actual power, making it possible to build an accurate ANN power model.

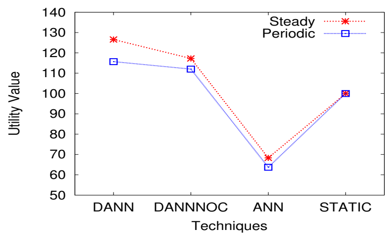
2) Utility, Performance and Power Results: In Fig. 7(a), the cumulative utility is shown, which is the sum of the global utility values for all time intervals, for each allocation technique averaged over all numbers of VMs and workload mixes. The cumulative utility is shown as a percentage of the STATIC allocation utility. DANN achieves a higher utility than all other approaches. Although DANNNOC has a better utility than the STATIC and ANN techniques, it is worse than DANN with statistical significance, indicating that coordination between ANNs is important to perform better. In Fig. 7(b), for each technique it is shown how the cumulative utility changes with the number of VMs. With an increased number of VMs, DANN becomes more effective than the other techniques. This is because by increasing the number of VMs, the uncertainty about the power modeling without considering all VMs increases, making the coordination between the ANNs a necessity for more accurate power modeling. In Fig. 7(c), it is shown



(a) Overall utility

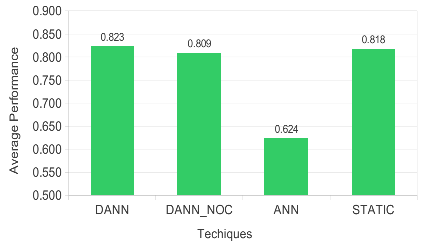


(b) Utility for different number of VMs

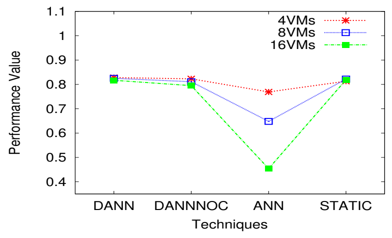


(c) Utility for different workload mixes

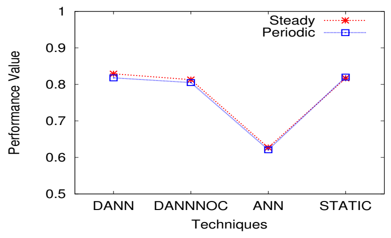
Figure 7: Overall utility and utility for different number of VMs and workload mixes.



(a) Overall average performance

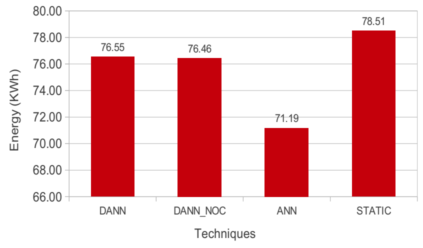


(b) Performance for different number of VMs

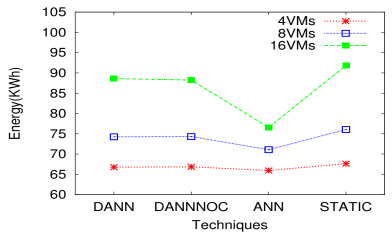


(c) Performance for different workload mixes

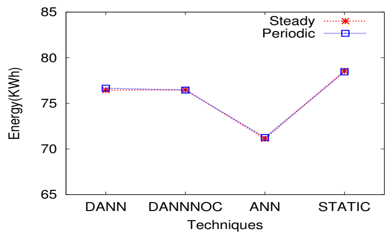
Figure 8: Overall average performance and average performance for different number of VMs and workload mixes.



(a) Overall Energy



(b) Energy for different number of VMs



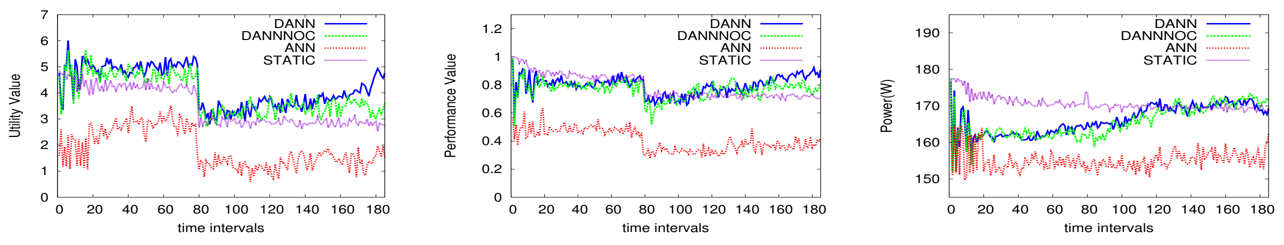
(c) Energy for different workload mixes

Figure 9: Overall energy and energy for different number of VMs and workload mixes.

how the cumulative utility changes with the workload mix. The utility achieved for each technique with a periodic workload mix is lower than with a steady workload mix. This is because with a periodic workload, the accuracy of the performance and power models decreases, since they need to adapt frequently to workload changes.

In Fig. 8(a), the overall performance for each technique averaged over all numbers of VMs and workload mixes is shown. DANN has a better performance than the other techniques (with statistical significance). Although DANN has a slightly better performance than STATIC, it consumes less power, resulting in a higher utility. In Fig. 8(b), it is shown how the performance changes with the number of VMs. The performance of DANN is not influenced by increasing the number of VMs, which is not the case for ANN. In Fig. 8(c), it is shown how the performance changes with the workload mix. With statistical significance, the performance achieved by DANN is only slightly reduced by changing the workload type.

In Fig. 9(a), for each technique the energy consumption during the experiment time for 200 machines averaged over all numbers of VMs and workload mixes is shown. DANN achieves energy savings compared to STATIC with statistical significance. Although DANN consumes the same energy as DANNNOC, it achieves a higher utility due to the higher performance than DANNNOC. The least energy consumption is achieved by ANN, but with the price of a lower performance. In Fig. 9(b), it is shown for each technique how the energy changes with the number of VMs. The energy savings of DANN compared to STATIC become larger with an increased number of VMs. This is because the range of dynamic power consumption becomes larger



(a) Utility (b) Average performance (c) Power consumption

Figure 10: Utility, average performance and power consumption for 8 VMs

with an increased number of VMs, increasing the potential of the technique to influence the energy consumption. In Fig. 9(c), it is shown how the energy changes with the workload mix. The energy consumption of each technique is not influenced by the workload mix, indicating the robustness of the techniques to keep the same level of energy savings.

C. Realistic IaaS Experiment: Results

We have performed experiments in a realistic Xen environment with 8 VMs, half of them running cpu_bench and the other half filebench. We have pinned 8 VMs to 3 CPU cores and have given them 3200 MB of memory, while we have pinned Dom0 to the remaining CPU core. We have assigned one VCPU to each of them. The experiment has been run for 185 time intervals, and in interval 80 we have triggered a workload change emulated by increasing the SLA performance level of 4 VMs. The experiment has been run 5 times and the average results are shown. Fig. 10(a), 10(b) and 10(c) show utility, average performance and power consumption over time for each technique. DANN has a better utility value than the other approaches. Although the difference with DANNNOC is not statistically significant for most of the experimental time, there are intervals where it has a better utility, such as in the intervals (60:80) and (170:185). Although most of the time DANN achieves the same levels of average performance as DANNNOC and STATIC, in the interval (150:185) DANN achieves better values. This is because in the second half of the experiment, the performance SLA levels are increased, making it more difficult for DANNNOC and STATIC to satisfy them. With respect to power, although DANN achieves the same power consumption as DANNNOC, it achieves much lower power levels than STATIC in the first half of the experimental time and increases it during the second half. During this interval (0:80), the SLA performance levels can be achieved with much less power consumption than using STATIC, while in the second half the power consumption should be increased to meet the new SLA performance level.

D. Overhead Results

The overhead of DANN comes from two factors: a) the coordination time in lines 4-8 of Algorithm 1 and b) the models' training times in line 15 of Algorithm 1. Running the resource manager on the physical machine with 4 CPU cores and managing 16 VMs, we get, on the average, 2.5 sec for the first factor and 30 sec for the second factor, showing the feasibility of the resource manager in practice.

VIII. Conclusions

In this paper, a resource management approach for VM resource allocation in IaaS clouds has been presented. It finds an adequate performance-power trade-off by expressing the two conflicting objectives of performance and power in a utility function and optimizing it using ANN based performance and power models. To cope with a potentially increased number of VMs, a distributed resource management approach has been developed where each ANN is responsible for modeling performance and power of a single VM, while exchanging information with other ANNs to coordinate resource allocation. Simulated and real experiments show that the distributed ANN resource manager achieves better utility values and performance-power trade-offs than a centralized ANN approach, a distributed non-coordinated version and a static allocation approach.

In the future, we plan to include the disk and network as resources to be allocated. We also plan to apply the approach in more complex environments with multi-tier applications and multiple physical machines where the coordination procedure for power modeling could be applied for coordinating multi-tier VMs running on different physical machines.

IX. Acknowledgment

This work is supported by the German Ministry of Education and Research (BMBF) and the Albanian Government.

References

[1] Filebench: file system and storage benchmark. http://sourceforge.net/apps/mediawiki/filebench/index.php.

[2] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, 13:80–96, 2002.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proc. 19th ACM Symposium on Operating Systems Principles*, pages 164–177. ACM Press, 2003.

[4] M. N. Bennani and D. A. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *Proc. 2nd International Conference on Automatic Computing*, pages 229–240. IEEE Press, 2005.

[5] R. Bitirgen, E. Ipek, and J. F. Martinez. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *Proc. 41st Annual IEEE/ACM International Symposium on Microarchitecture*, pages 318–329. IEEE Computer Society, 2008.

[6] P. Bodík, R. Griffith, C. Sutton, A. Fox, M. Jordan, and D. Patterson. Statistical machine learning makes automatic control practical for internet datacenters. In *Proc. Conference on Hot topics in Cloud Computing*. USENIX Association, 2009.

[7] R. P. Doyle. Model-based resource provisioning in a web service utility. In *Proc. 4th USENIX Symposium on Internet Technologies and Systems*, pages 5–5. USENIX Association, 2003.

[8] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In *Proc. Advances in Neural Information Processing Systems 2*, pages 524–532. Morgan Kaufmann, 1990.

[9] T. Heath, B. Diniz, E. V. Carrera, W. Meira, Jr., and R. Bianchini. Energy conservation in heterogeneous server clusters. In *Proc. 10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 186–195. ACM Press, 2005.

[10] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal. Application heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments. In *Proc. International Conference on Autonomic Computing*, pages 79–88. ACM Press, 2010.

[11] J. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[12] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[13] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao, and K. Dutta. Modeling virtualized applications using machine learning techniques. In *Proc. 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments*, pages 3–14. ACM Press, 2012.

[14] H. Li, G. Casale, and T. Ellahi. Sla-driven planning and optimization of enterprise applications. In *Proc. WOSP/SIPEW International Conference on Performance Engineering*, pages 117–128. ACM Press, 2010.

[15] K. Magnus, Z. Xiaoyun, and K. Christos. An adaptive optimal controller for non-intrusive performance differentiation in computing services. In *Proc. IEEE Conference on Control and Automation*, pages 709–714. IEEE press, 2005.

[16] Matthew Wall. A C++ Library of Genetic Algorithm Components. http://lancet.mit.edu/ga/.

[17] D. A. Menasce and M. N. Bennani. Autonomic virtualized environments. In *Proc. International Conference on Autonomic and Autonomous Systems*, pages 28–38. IEEE Press, 2006.

[18] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *Proc. 4th ACM European Conference on Computer Systems*, pages 13–26. ACM Press, 2009.

[19] J. Rao, X. Bu, C.-Z. Xu, L. Wang, and G. Yin. Vconf: A reinforcement learning approach to virtual machines auto-configuration. In *Proc. 6th International Conference on Autonomic Computing*, pages 137–146. ACM Press, 2009.

[20] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Proc. IEEE International Conference on Neural Networks*, pages 586–591. IEEE Press, 1993.

[21] Steffen Nissen. Fast Artificial Neural Network Library. http://leenissen.dk/fann/wp/.

[22] G. Tesauro, R. Das, H. Chan, J. O. Kephart, C. Lefurgy, D. W. Levine, and F. Rawson. Managing power consumption and performance of computing systems using reinforcement learning. In *Proc. Advances in Neural Information Processing Systems 20*, pages 1–1. Curran Associates, Inc., 2008.

[23] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani. On the use of hybrid reinforcement learning for autonomic resource allocation. *Cluster Computing*, 10(3):287–299, 2007.

[24] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah. Analyzing the energy efficiency of a database server. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 231–242. ACM Press, 2010.

[25] VMWare Inc. VMWare Homepage. http://www.vmware.com/, 2011.

[26] Z. Wang, X. Zhu, S. Singhal, and H. Packard. Utilization and slo-based control for dynamic sizing of resource partitions. In *Proc. 16th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, pages 24–26. Springer-Verlag, 2005.

[27] J. Wildstrom, P. Stone, and E. Witchel. Carve: A cognitive agent for resource value estimation. In *Proc. 5th IEEE International Conference on Autonomic Computing*, pages 182–191. IEEE Press, 2008.

[28] J. Wildstrom, P. Stone, E. Witchel, and M. Dahlin. Machine learning for on-line hardware reconfiguration. In *Proc. 20th International Joint Conference on Artifical Intelligence*, pages 1113–1118. Morgan Kaufmann Publishers Inc., 2007.

[29] Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing: state-of-the-art and research challenges. *J. Internet Services and Applications*, 1(1):7–18, 2010.