Energy Efficiency in Cloud Computing

Christoph Aschberger, Franziska Halbrainer

July 18, 2013

1 Introduction

In the last years cloud computing has become more and more popular. This increase in popularity of cloud services results in higher resource demands on the providers end. More resources means more energy consumption and thus higher electricity bills. To get an idea how much energy is consumed by a cloud service provider we found that Google published their energy consumption in 2011 to have been 2,675,898 MWh [4]. To better grasp this number they compared it to energy usage of an American city:

> We found that we use roughly as much electricity globally as 220,000 people, based on electricity use per capita in the US. If we compared our electricity use to, say, an American city, we'd most closely compare to the metropolitan area of Sioux Falls, South Dakota. [5]

There is a need to make a cloud service more profitable by reducing energy usage while at the same time keeping the service level for the customer. In this paper we want to discuss several ways found in scientific literature to achieve this goal.

The easiest and most obvious way to save energy is to run fewer machines. This however comes with a trade off. Customers expect the cloud to handle sudden increases in demand, as such it is not as simple as turning unused machines off. As a result algorithms are needed to decide in a smart way which machines can be turned off and if new machines should be powered up. Moreover to get empty machines to turn off, scheduling needs energy awareness.

Since most modern machines have different power states, it is also an option to power on/off not directly but by passing through these different states. By running machines in low power state they need less energy but do not provide their full capacity. But the change from one power state to its next is much faster than from full capacity to turned off, so sudden increases can easier be served.

The remainder of this paper is organized as follows. Section 2 gives a general idea about energy consumption and workload placement in the cloud. Section 3 presents several methods for workload placement focusing on energy saving and for every method its preconditions, the basic idea and the evaluation. In section 4 we draw a conclusion on this paper.

2 General

2.1 Energy Model

When talking about energy efficiency in clouds one has to identify the components that consume energy and needs to analyze how much energy these components consume on different workloads. In [9] the energy consumption of a cloud over a time span is calculated as shown in the following equations:

$$E_{Cloud} = \int_{t1}^{t2} E_{Nodes} + E_{Switches} + E_{Storages} + E_{Others} dt \tag{1}$$

$$E_{Node} = E_{CPU} + E_{Memory} + E_{Disk} + E_{Mainboard} + E_{NIC} \tag{2}$$

$$E_{Switch} = E_{Chassis} + E_{Linecards} + E_{Ports} \tag{3}$$

$$E_{Storage} = E_{NASServer} + E_{StorageController} + E_{DiskArray} \tag{4}$$

Although these factors all influence the total used energy most of the methods we present mainly focus on reducing the number of nodes. Energy usage of switches is only indirectly taken into account by trying to move workloads as few times as possible. One method focuses solely on reducing CPU usage (3.1.1).

2.2 Cloud Model

In the following methods there is one fundamental difference: the cloud model. Either tasks or virtual machines are executed in the cloud. This affects the possible approaches to energy saving.

Tasks have a nice advantage for energy aware scheduling, since many requirements are known in advance. For example the exact needed CPU usage must be known and more importantly it is required to know the execution time. The given values are not considered to be an upper bound but what a task really needs for the whole execution time. Since the execution times are known it is also clear when a resource is free again.

Virtual machines (VM) on the other hand are allocated to physical machines (PM or host) based on an upper bound that is given by the customer and the VM might use less than agreed upon. Lifetime of a virtual machine is unknown because customers reserve instances and pay for them as long as they are running. As a result machines might stop suddenly or run indefinitely. An energy aware scheduling algorithm for VMs must not only handle sudden increases but should also be capable of sudden decreases.

2.3 VM migration [2, 1]

As VMs in a cloud computing environment may have different requirements over time one has to consider adapting to this changing behavior. There is a possibility that a VM does not need the agreed performance and this opens a chance to allocate more VMs on a PM than would fit when only considering

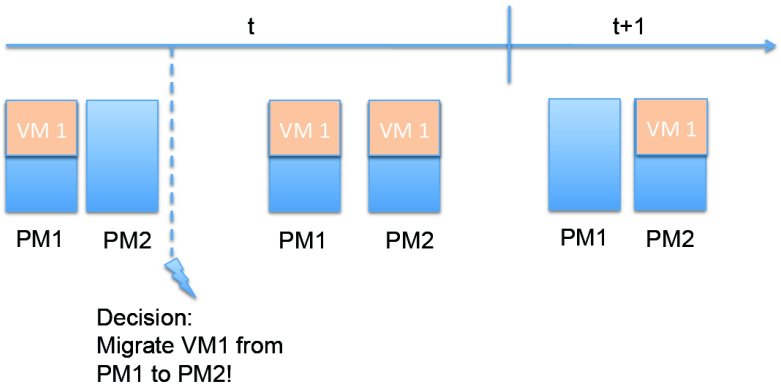

Figure 1: migrating VM1 from PM1 to PM2 [2]

the maximum needed characteristics. If one of the VMs would need more of a resource than is available on the current PM it has to be moved to a different one. By distributing all VMs of a PM to other under utilized hosts the then unused PM can be switched off. The process of moving a VM to another physical location is called (live)migration. Migration however does not come without costs. The performance of a VM during migration may be degraded, additional energy is needed while the VM is allocated on two hosts as illustrated in figure 1 and the network load is increased for the migration time. There is a trade off between using as few PMs as possible and risking migrating often or minimizing the number of migrations at the cost of powering more PMs. One also has to consider that switching hosts off and on takes time, therefore a machine just turned off is not immediately available for migration. To compensate that either some additional unused machines need to be kept powered on or can only put into sleep or hibernate state to quickly react to sudden load peaks.

3 Approaches

3.1 Task

3.1.1 Dynamic Voltage Frequency Scaling [6]

Preconditions A heterogeneous hardware basis is assumed. Hosts need to support DVFS. The tasks may vary in their needs.

Method The energy usage E of a task running with a certain frequency f can be expressed with the following equation

$$E = k \cdot v^2 \cdot f \cdot \Delta t \tag{5}$$

where k is a device dependent constant, v is the voltage and Δt is the execution time.

Since in many cases the important thing about a task execution is when it has to be finished, it might not be necessary to execute every task with the maximum possible frequency.

By lowering the frequency of a task a lower voltage level can be set for the CPU. Lower frequency leads to a longer execution time but due to the quadratic occurrence of the voltage in equation 5 lowering voltage leads to a decrease in energy usage.

The authors assume an interdependent set of tasks which can be executed in parallel. They formulate the problem as a Directed Acyclic Graph (DAG) with the tasks as nodes and the edge weights as communication time. If two tasks with dependencies are executed on different processors the communication time has to be considered, if located on the same processor this time is reduced to zero. To map the tasks onto processors they use a a near-optimal list-scheduling heuristic called Heterogeneous Earliest Finish Time (HEFT).

After the mapping phase idle and processing times of each processor can be determined. To optimize energy usage each execution time of a task is expanded to the time span reaching from the earliest possible start time to the latest possible finish time. Earliest possible start time is the time when all the tasks the current task depends on are finished. Latest possible finish time is the minimum of the latest start time of all tasks that depend on the current task. Slack time is the difference between this time span and the execution time at highest frequency. This time can be used to slow the computation down and save energy. In theory, if no overlapping occurs, the slack time can be added to the execution time and the frequency can be set to a value which uses this whole time for execution. In practice this might lead to overlapping execution times, therefore this frequency needs to be adjusted.

For finding the right frequency one has to start with the task n_i with the global latest finish time and build a set $S = \{n_i\}$. All tasks which have overlapping execution times with at least one task in S and are mapped to the same processor must be added to S. Then $T_{exec}(S)$ can be defined as the sum of all execution times of tasks in set S and $T_{total}(S)$ as earliest earliest start time - latest latest finish time. The frequency of task n_i can then be set as seen in equation 6:

$$f_{global}(n_i) = f_{max} \max\left(\frac{T_{exec(n_i)}}{T_{exec(n_i)} + T_{slack(n_i)}}, \frac{T_{exec}(S)}{T_{total}(S)}\right) \tag{6}$$

This ensures that the frequency is not too slow to exceed its own execution time nor too slow to force the other tasks of the set to run with more than 100% frequency.

After the frequency is determined the execution time of the task needs to be updated. This might also influence execution times of preceding tasks, which need updates for their execution times and deadlines as well. Then a new n_i with a new set S needs to be found, excluding all already handled tasks. These steps are repeated until the frequency for each task is set.

With this algorithm a global optimum can be found.

Table 1 shows some tasks with their execution time used for the following example. Table 2 shows available frequency levels and their associated voltage levels. Figure 2 shows the DAG with nodes being the tasks and edges being dependencies. The weight of an edge is the communication time if tasks are scheduled to different processors. It also shows the initial scheduling after applying HEFT. As the scheduling in this example results in a fully utilized processor P1 only P2 needs optimization. To calculate the energy consumption

task	1	2	3	4	5	6
t_{exec}	1	1	2	1	4	1

Table 1: sample tasks with execution times

frequency	100%	80%	60%	50%	40%	33.3%
voltage	1.2v	1.1v	1.0v	0.9v	0.8v	0.7v

Table 2: frequency and voltage levels

of P2 the following equation can be used.

$$E_{total} = E_{tasks} + E_{idle} \tag{7}$$

The initial scheduling would result in the energy consumption of $E_{total}(initial) = k \cdot \left(1.2^2 \cdot 1 \cdot 2 + 0.7^2 \cdot \frac{1}{3} \cdot 6\right) = 3.86 \cdot k$

Figure 3 shows the optimized schedule of P2. Tasks 2 and 4 run for twice the time with halved frequency which eliminates some idle time.

The optimized schedule results in an energy consumption of $E_{total}(slacked) = k \cdot \left(0.9^2 \cdot \frac{1}{2} \cdot 4 + 0.7^2 \cdot \frac{1}{3} \cdot 4\right) = 2.273 \cdot k$

Evaluation The proposed algorithm is compared to a greedy based approach, a path based approach and an energy conscious scheduling algorithm. The simulated hardware is based on three real heterogeneous processors. For test sets randomly constructed DAGs as well as DAGs from a real world application of a Gaussian elimination algorithm were used. Measured were the total energy consumption and compared to the original HEFT schedule. The proposed algorithm is for most of the test cases the most energy efficient.

3.1.2 Task consolidation [8]

Preconditions A homogeneous hardware basis is assumed. The tasks may vary in their needs. CPU requirements and time constraints of a task are known or can be estimated beforehand.

Method Consolidation can be used when operating on the level of tasks. Instead of running each task on its own machine, tasks which require only a fraction of the total resources may be placed on the same machine. This results in energy savings because the energy usage of running two tasks on the same

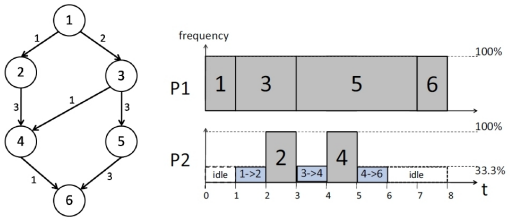

Figure 2: DAG and initial mapping [6]

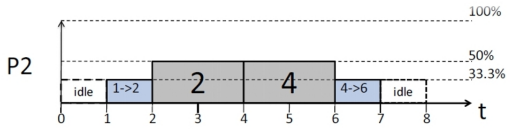

Figure 3: scheduling after slacking [6]

Task	Arrival time	Processing time	Utilization
0	0	20	40%
1	3	8	50%
2	7	23	20%
3	14	10	40%
4	20	15	70%

Table 3: sample tasks

machine is smaller than running each task on its own machine. The scope of the paper is focused on short tasks and decisions for a short term. To decide on which machine a task will be executed a cost function is calculated for every host and the task is assigned to the machine with the highest value. There are two different cost functions: MaxUtil and ECTC.

To show the different effects of the cost functions the sample tasks as shown in table 3 are used in the example figures.

MaxUtil aims to utilize available resources as much as possible, by choosing the solution where the average utilization during the task execution is the highest.

$$f_{i,j} = \frac{\sum_{\tau=1}^{\tau_0} U_i}{\tau_0} \tag{8}$$

Equation 8 shows the cost function of MaxUtil where U_i is the utilization of a resource r_i at a given time. From the equation one can determine the worst and best case. Best case is achieved when the utilization during the period is maximized. The worst case is when the task is the only one running on the machine.

Figure 4 shows how MaxUtil would allocate task t_3 when tasks t_0, t_1 and t_2 are already running. Task t_3 could either be allocated to r_0 or r_1 with the values of cost functions of $f_{0,3} = \frac{6 \cdot 40\%}{10} = 0.24$ and $f_{1,3} = \frac{10 \cdot 20\%}{10} = 0.2$.

As r_0 yields a higher value t_3 is scheduled to r_0.

ECTC calculates the energy consumption of a task and subtracts the idle consumption if other tasks are allocated on this PM at the same time. This is necessary to make solutions where only one task is allocated on PMs less appealing.

$$f_{i,j} = ((p_\Delta \cdot u_j + p_{min}) \cdot \tau_0) - ((p_\Delta \cdot u_j + p_{min}) \cdot \tau_1 + p_\Delta \cdot u_j \cdot \tau_2) \tag{9}$$

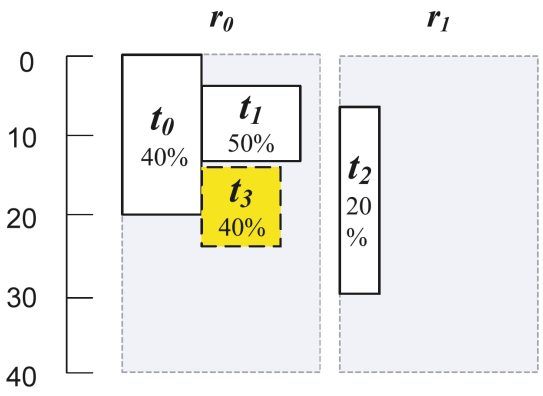

Figure 4: scheduling task t_3 on r_0 using MaxUtil [8]

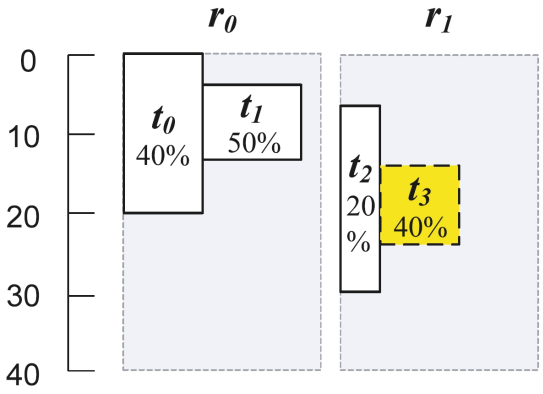

Figure 5: scheduling task t_3 on r_1 [8]

Equation 9 shows the cost function of ECTC where $\tau_0 = \tau_1 + \tau_2$, τ_1 is the time at which the task runs alone on the machine, τ_2 is the time at which the task does not run alone on the machine, $p_\Delta = p_{max} - p_{min}$, p_{min} is the energy consumption of an idle machine, p_{max} is the maximal energy consumption of the machine.

For the assumed homogeneous hardware basis this equation can be simplified to equation 10.

$$f_{i,j} = \tau_0 - \tau_1 \tag{10}$$

From the shortened equation one can determine the worst and best case. Best case is achieved when the task is running with other tasks for its whole execution time. The worst case is again when the task is the only one running on the machine.

As before t_0, t_1 and t_2 are already running and task t_3 arrives. This yields the values $f_{0,3} = 10 - 4 = 6$ and $f_{1,3} = 10 - 0 = 10$. This time r_1 has the higher value and t_3 gets scheduled on r_1.

The real difference between the two cost functions can be seen when the decision where task t_4 should be scheduled has to be made. If ECTC is used the utilization of both machines is greater or equal to 40%, so in order to run t_4,

which requires 70% utilization, a new machine has to be powered up. Contrary to using MaxUtil where t_4 fits on r_1 which utilizes only 20% of its CPU.

Evaluation ECTC and MaxUtil are compared to random Task allocations. The test sets where built based on three different parameters. Number of tasks (100 to 5000 in intervals of 100), mean inter-arrival time (10 to 100 in steps of 10) and resource usage patterns (random, low (mean 30%), high (mean 70%)). Task processing times are exponential distributed and task arrival times are based on a Poisson process.

For all tests ECTC is shown to be more efficient than MaxUtil. The highest energy savings can be achieved with the low resource utilization (25% for MaxUtil and 33%for ECTC).

3.2 Virtual Machines

3.2.1 Thresholds [1]

Preconditions No assumptions on VMs or PMs are made.

Method Two Steps are needed in VM migration: First it has to be decided if VMs need to be migrated and if so which VMs and where they have to be migrated to. The authors suggest for the first step a threshold system. Thresholds are straight forward solution to keep some resources reserved in order not to run into immediate trouble once the utilization increases. In its simplest form only one fixed threshold is sufficient. It defines an upper limit of utilization, as long as its not violated new VMs can be allocated to the host. A more sophisticated approach is to use two fixed thresholds as upper and lower limit. If one of the thresholds is breached VMs need to be migrated to resolve the situation. If too many VMs are allocated some have to migrate to different hosts. If too few are allocated all VMs should be migrated to other hosts in order to shut down this machine.

Dynamic thresholds open the possibility to adapt to the changing behavior of the VMs. The utilization of one VM is assumed to be a random variable u_j, and different VMs not necessarily have the same distribution for u_j. The host utilization is the sum of all u_js of the VMs allocated to it. If the u_js are distributed differently the host utilization can be assumed to be normal and be modeled with the t-distribution. Based on the inverse cumulative probability function an interval which will be reached with a low probability can be calculated. This interval together with the standard deviation can be used to set the upper threshold for each host individually. The idea is to have a higher threshold for hosts with VMs which have a behavior that does not change often than for hosts with VMs where the behavior changes constantly. The lower utilization threshold is calculated as a global value across all hosts to identify the machines where the utilization is lower than the average. As the u_js are collected for each VM and only later summed up this data can also be used after a VM has been migrated to allow a more accurate prediction.

To answer the question of which VMs to migrate the authors propose three different selection policies:

- **Minimization of Migrations (MM)** – migrate the least number of VMs to minimize migration overhead

- **Highest Potential Growth (HPG)** – migrate VMs that have the lowest usage of CPU relatively to requested in order to minimize total potential increase of the utilization and SLA violation
- **Random Choice (RC)** – choose the necessary number of VMs randomly

The second step in VM Migration is called VM Placement. The chosen algorithm is a Modified Best Fit Decreasing where the modification is the consideration of power usage. VMs get sorted by their CPU usage in a decreasing order. VMs are allocated to the host where the least increase of power consumption after allocation occurs. This has the effect of preferring the power efficient hosts over less efficient ones.

Evaluation The test environment was a simulated data center with heterogeneous hardware. For workload data traces of real world workloads were taken.

For the dynamic thresholds several parameters were evaluated to find the best ones. This one were compared with the single and multiple fixed thresholds and a random algorithm. Compared to the most energy efficient method, one of the single fixed thresholds, the dynamic thresholds have higher energy consumptions but only a fraction of SLA violations. The multiple fixed threshold methods are in the middle between the two extremes.

3.2.2 Algorithms [2]

Preconditions No assumptions on VMs or PMs are made.

Method The authors use a knowledge base that has all the information about VMs and PMs current and past resource utilization and the Service Level Agreements (SLA). This knowledge base decides, when the current (VM,PM) matching does not fit anymore and recommends actions for changing. Every resource may have its own two thresholds. Since VMs are not allocated with their SLA but with a bit more than they currently need a rise or decay in its demand requires a reconfiguration of the VM. If after a reconfiguration of a VM a host has reached its limits a migration is necessary. This paper introduces several algorithms for migration: First Fit, Monte Carlo and Vector Packing.

- First Fit: Allocation takes place in a first fit manner. For reallocation half of the most loaded machine and all of the least loaded machine is distributed with first fit.
- Monte Carlo: The VMs are allocated with Round Robin so they are equally distributed. For reallocation a cost function is used to tell the cost of the current allocation. Several alternative allocations are created with the Monte Carlo method and the cost function is calculated for the alternatives too. The allocation with the lowest costs will be taken. Some parameters of the cost function are cost for migration, cost for overloaded hosts and costs for empty hosts. Setting positive values for the first two parameters and negative values for the last one, solutions with little migration, few overloaded hosts and high number of empty hosts will be preferred.

- Vector Packing: First of all the VMs get sorted with highest resource needs first. Then they get allocated in a First Fit manner but trying not to imbalance the resources of the PM. Reallocation works the same way but only the most loaded PMs are taken into account.

Since some of the PMs are getting empty after reallocation, the authors also propose a way to power down as shown in equation 11. At every step only a fraction of the empty PMs shall be turned off, so a sudden peak can easier be handled. Also when there are many empty machines, they will be powered down in an exponential manner influenced by parameter a. At least one PM shall always run.

$$\text{Number of PMs to switch off} = \frac{\text{Number of empty PMs}}{a} \tag{11}$$

For powering up machines again power efficient hosts are considered first.

Evaluation The test were carried out with a simulation of 100 homogeneous physical machines. Three different workload characteristics were tested: a workload with little changes, a workload with medium changes and a workload based on real data, which needed more CPU than the other workloads. For the little workload changes and the real test set Monte Carlo is the best choice. For medium workload changes First Fit works best. Even though Vector Packing is never the best it is best on average of all workloads. While Monte Carlo does deliver good results, it takes the longest to compute and does not scale well with a higher number of VMs.

3.2.3 Decentralized VM migration [10]

Preconditions A homogeneous hardware basis, where each host is capable of holding the same amount of same-sized VMs is assumed. The protocol can easily be extended to handle heterogeneous hardware and VMs.

Method The authors present a decentralized approach which can cope with a varying amount of hosts. They propose a gossiping protocol which decides how many VMs are allocated onto a PM.

The protocol includes two threads running on each PM, one is called the active thread the other is passive. In the following algorithms H_i is the current amount of VMs allocated to the host i and C is the maximum capacity of a host.

Algorithm 1 Procedure ACTIVE THREAD

```
1: loop
2:   Wait Δ
3:   for all j ϵ GetNeighbors(i) do
4:     Send (H_i) to j
5:     Receive (H'_i) from j
6:     H_i ← H'_i
7:   end for
8: end loop
```

The active thread sends messages to its neighbors with its current count of virtual machines. It gets a new value from the neighbor and updates its count with this value and continues the protocol by sending the next message to the next neighbor.

Algorithm 2 Procedure PASSIVE THREAD

```
1: loop
2:    Wait for message H_j from j
3:    if H_i ≥ H_j then
4:       D ← min(H_j, C − H_i)
5:       Send (H_j − D) to j
6:       H_i ← H_i + D
7:    else
8:       D ← min(H_i, C − H_j)
9:       Send (H_j + D) to j
10:      H_i ← H_i − D
11:   end if
12: end loop
```

The passive thread on another machine listens for the messages of an active thread. Upon arrival it needs to decide whether the count of VMs on the sending machine needs to be in- or decreased. The idea is that the machine with the higher number of VMs tries to gather as much VMs as possible from the other machine. The answering machine in the passive thread updates its own VM counter and sends the updated VM counter of the other machine back to the requesting active thread.

This process needs to run in an endless loop. If at the end of one iteration the host has no VMs allocated it can be put into a lower power consumption mode.

As there is no global knowledge involved the set of neighbors is only a subset of all machines. Therefore the active thread talks only to a few hosts. To always have an up-to-date view of the neighborhood hosts exchange messages with their known neighbors and merge this information with their own view.

This protocol can handle that hosts appear and disappear suddenly which is a reasonable assumption in a cloud environment. Although the algorithms are simple, they do not cover the decision which VMs to migrate, only the decision on the number of VMs to migrate is made. If heterogeneous hardware and VMs need to be handled the algorithms will become more complex as C will not be constant but needs to be determined for each host independently and calculating remaining VMs needs to consider different VM sizes.

Evaluation Results of this method are not analyzed in energy efficiency but rather in how many hosts need to be powered up. Simulations are run with different set-ups to show how the approach reacts to different situations. In a static situation where hosts and number of VMs are not changed it is shown that the protocol leads fast to a near optimal solution. If a bigger neighborhood is used solutions converge faster. If the VM count is not static but varies each time step the protocol keeps the utilization near the optimum. If the protocol is turned off it deviates from it. In another experiment it is shown, that sudden

hardware outages or additions can be handled.

3.2.4 Artificial Intelligence Approach [3]

Preconditions A homogeneous hardware basis is assumed. The VMs may vary in their needs.

Multidimensional-Bin-Packing (MDBP) The authors define the problem as a MDBP, where the bins are hosts, the items are the VMs to be placed on the PM and the goal is to use as few PMs as possible. A description of the MDBP-Problem can be found in [7].

> Bin packing problems involve the packing of objects of given sizes into bins of given capacity. In the case of one-dimensional bin packing the size of each object is a real number between 0 and 1, and each bin is of capacity one. It is required that the sum of the sizes of the objects packed into any given bin may not exceed 1. The problem of finding a packing using a minimum number of bins is known to be NP-hard [...]. [7, Page 289]

This can be generalized to a multidimensional bin packing problem where limits in each dimension must not be exceeded.

Method The authors choose an AI approach for VM migration. They propose an ant colony as model to solve the MDBP to get an optimized workload placement.

Ants communicate through pheromones. The pheromone is emitted by an ant and evaporates after some time. When food is searched the shortest path has the highest concentration of pheromones because the ants on this path return faster and the pheromone therefore has less time to evaporate. Ants tend to prefer paths with higher concentration, thus the best way to a new food source is found after some time through indirect communication.
As for the technical implementation the authors describe the Ant Colony Optimization (ACO) algorithm as

> artificial ants act as multi-agent system and construct a complex solution based on indirect low-level communication. [3, Page 27]

Each ant knows about all running VMs and all hosts. An ant allocates a VM i on a PM v based on a probabilistic decision rule p as shown in equation 12, which takes into account the current pheromone level τ and heuristic information η. α and β are parameters to control the influence of heuristic and pheromone level. N_v is the set of remaining VMs which are not yet allocated and fit on host v.

$$p_v^i := \frac{[\tau_{i,v}]^\alpha \times [\eta_{i,v}]^\beta}{\sum_{u \in N_v} [\tau_{u,v}]^\alpha \times [\eta_{u,v}]^\beta}, \quad \forall i \in N_v \tag{12}$$

Equation 13 determines the load b of the host v as the sum of the resources r used by the allocated VMs.

$$\vec{b}_{i,v} := \sum_{i \in B_v} \vec{r}_i \tag{13}$$

The heuristic information as shown in equation 14 is necessary for the ants to choose VMs so that the hosts are better utilized. It is the scalar value of the inverse of the capacity $\vec{C}$ minus the future load.

$$\eta_{i,v} := \frac{1}{|\vec{C_v} - (\vec{b_v} + \vec{r_i})|_1} \tag{14}$$

When all ants have found their workload placement, the pheromone level is calculated by decreasing the old pheromone level by a factor $(1 - \rho)$ and adding pheromones from the best ant in this round $\Delta\tau_{i,v}^{\text{best}}$.

$$\tau_{i,v} := (1 - \rho) \times \tau_{i,v} + \Delta\tau_{i,v}^{\text{best}}, \quad \forall(i, B_v) \in I \times B \tag{15}$$

Where I is the set of items and B is the set of bins.

The concentration must always lay between certain thresholds $[\tau_{min}, \tau_{max}]$ in order to prevent early stagnation. Termination is assured by restricting the algorithm to a fixed amount of iterations. The output is chosen to be the global best solution so far.

Algorithm 3 shows the complete algorithm for finding the best solution. After the setup, lines 6 to 20 show how the mapping of VMs is done. Then the Best solution is updated if necessary. Lines 25 to 34 show the update of the pheromone level. This is done for a fixed number of cycles until the global best solution is returned.

Evaluation The test were run on a simulated cluster of homogeneous hosts. There where 600 hosts and as many VMs. The evaluation period was 24 hours. The energy consumption was estimated as a linear function based on the CPU utilization. The idle machines were not taken into account for the energy consumption since they were assumed to be turned off. The energy consumption were not only taken from the workload but also considered for the algorithm. The optimal parameters were found through experimental testing. The measured values are the amount of hosts with VMs, total energy consumption and average execution time for the algorithm. The results are compared to a simple greedy algorithm and to an optimal solution. The proposed algorithm is near to the optimal solution but needs more time for computation. Computation time could be reduced with some optimization.

4 Conclusion

We have presented six different methods to reduce energy consumption in a cloud computing environment. Each approach originates from its own application scenario and has therefore different requirements, which makes comparing them difficult. In addition each method used its own test sets, simulated hardware and VM configurations. Some compared their solution to the optimum others compared it to competitors or to the simplest approach like random or greedy allocation. Most results show energy usage while some show utilization. Furthermore some methods only approximate energy usage instead of measuring it.

Algorithm 3 Energy-Aware ACO-based Workload Consolidation

Input: Set of items I and set of bins B with their associated resource demand vectors $\vec{r}_i$ and $\vec{C}_v$ respectively, Set of parameters
Output: Global best solution S_{best}

Initialize parameters, Set pheromone value on all item-bin pairs to τ_{max}
for all $q \in \{0 \dots nCycles - 1\}$ **do**
  for all $a \in \{0 \dots nAnts - 1\}$ **do**
    $IS := I; v := 0$
    $S_a := [x_{i,j} := 0], \forall i \in \{0, \dots, m-1\}, \forall j \in \{0, \dots, n-1\}$
    while $IS \neq \emptyset$ **do**
      $N_v := \{i | \sum_{j=0}^{n-1} x_{i,j} = 0 \wedge \vec{b}_v + \vec{r}_i \leq \vec{C}_v\}$
      if $N_v \neq \emptyset$ **then**
        Choose item $i \in N_v$ stochastically according to probability $p_v^i := \frac{[\tau_{i,v}]^\alpha \times [\eta_{i,v}]^\beta}{\sum_{u \in N_v} [\tau_{u,v}]^\alpha \times [\eta_{u,v}]^\beta}$
        $x_{i,v} := 1$
        $IS := IS - i$
        $\vec{b}_v := \vec{b}_v + \vec{r}_i$
      else
        $v := v + 1$
      end if
    end while
  end for
  Compare ants solutions S_a according to the objective function $f \rightarrow$ Save cycle best solution as S_{cycle}
  if $q = 0 \vee IsGlobalBest(S_{cycle})$ **then**
    Save cycle best solution as new global best S_{best}
  end if
  Compute τ_{min} and τ_{max}
  for all $(i, B_v) \in I \times B$ **do**
    $\tau_{i,v} := (1 - \rho) \times \tau_{i,v} + \Delta\tau_{i,v}^{\text{best}}$
    if $\tau_{i,v} > \tau_{max}$ **then**
      $\tau_{i,v} := \tau_{max}$
    end if
    if $\tau_{i,v} < \tau_{min}$ **then**
      $\tau_{i,v} := \tau_{min}$
    end if
  end for
end for
return Global best solution S_{best}

Most approaches need global knowledge to work properly. We assume that in reality in a huge data center this might be impractical. One major difference in the VM approaches is that some need periodical checks to update the placement while others react to changes in VM usages. A combination of both seems useful. Reacting on changes reduces the amount of violations of service level agreements, while periodical checks can be used to rebuild an optimal placement which might have been destroyed by reactive actions.

We identified some possibilities for future work.

Performance aspects are almost completely ignored. On the one hand it is understandable to focus solely on the energy part, on the other hand it would be interesting to have an approach which considers for example placing cooperating VMs on the same host or in the same rack. With this placement not only the performance would be improved but also the power needed for switching the communication between these VMs, since less switches are involved.

We think that some approaches might be possible to combine. For example to solve the decision problem in [10] the different approaches in [1] could result in reasonable results. The reconfiguration of VMs could be implemented in approaches which now only consider the maximum capacity of a VM, this should, in theory, lead to even better results.

Since energy awareness in cloud computing is a relatively new topic, there are a lot of other research possibilities.

References

[1] Anton Beloglazov and Rajkumar Buyya. Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers. In *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science*, MGC '10, pages 4:1–4:6, New York, NY, USA, 2010. ACM.

[2] Damien Borgetto, Michael Maurer, Georges Da-Costa, Jean-Marc Pierson, and Ivona Brandic. Energy-efficient and SLA-aware management of IaaS clouds. In *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*, e-Energy '12, pages 25:1–25:10, New York, NY, USA, 2012. ACM.

[3] Eugen Feller, Louis Rilling, and Christine Morin. Energy-Aware Ant Colony Based Workload Placement in Clouds. In *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing*, GRID '11, pages 26–33, Washington, DC, USA, 2011. IEEE Computer Society.

[4] Google Inc. The Big Picture FAQs - Google Green. http://www.google.com/intl/en/green/bigpicture/#/intro/infographics-1, 2013-07-11.

[5] Google Inc. The Big Picture FAQs - Google Green. http://www.google.com/intl/en/green/bigpicture/references.html, 2013-07-11.

[6] Qingjia Huang, Sen Su, Jian Li, Peng Xu, Kai Shuang, and Xiao Huang. Enhanced Energy-Efficient Scheduling for Parallel Applications in Cloud. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, CCGRID '12, pages 781–786, Washington, DC, USA, 2012. IEEE Computer Society.

[7] Richard M. Karp, Michael Luby, and A. Marchetti-Spaccamela. A probabilistic analysis of multidimensional bin packing problems. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, STOC '84, pages 289–298, New York, NY, USA, 1984. ACM.

[8] YoungChoon Lee and AlbertY. Zomaya. Energy efficient utilization of resources in cloud computing systems. *The Journal of Supercomputing*, 60(2):268–280.

[9] Liang Luo, Wenjun Wu, Dichen Di, Fei Zhang, Yizhou Yan, and Yaokuan Mao. A resource scheduling algorithm of cloud computing based on energy efficient optimization methods. In *Green Computing Conference (IGCC), 2012 International*, pages 1–6, 2012.

[10] M. Marzolla, O. Babaoglu, and F. Panzieri. Server consolidation in Clouds through gossiping. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a*, pages 1–6, 2011.