# Cost-efficient Datacentre Consolidation for Cloud Federations

Gabor Kecskemeti[1], Andras Markus[2] and Attila Kertesz[2]
[1] *Department of Computer Science, Liverpool John Moores University, United Kingdom*
[2] *Software Engineering Department, University of Szeged, 6720 Szeged, Dugonics ter 13, Hungary*
*g.kecskemeti@ljmu.ac.uk, Markus.Andras@stud.u-szeged.hu, keratt@inf.u-szeged.hu*

Keywords:
Cloud Computing, Datacentre consolidation, Simulation

Abstract:
Cloud Computing has become mature enough to enable the virtualized management of multiple datacentres. Datacentre consolidation is an important method for the efficient operation of such distributed infrastructures. Several approaches have been developed to improve the efficiency e.g. in terms of power consumption, but only a few attention has been turned to combining pricing methods with consolidation techniques. In this paper we discuss how we introduced cost models to the DISSECT-CF simulator to foster the development of cost efficient datacentre consolidation solutions. We also exemplify the usage of this extended simulator by performing cost-aware datacentre consolidation. We apply real world traces to simulate cloud load, and propose 7 strategies to address the problem.

## 1 Introduction

Cloud computing enabled the virtualized management and sharing of software and hardware solutions, including computing and storage resources and application runtimes. The elasticity of Infrastructure as a Service (IaaS) clouds allows commercial providers to better exploit their datacentres as well as increase their incomes. Datacentre consolidation is a technique that helps achieving these goals. Related works in datacentre load balancing and consolidation have already shown that multi-objective proposals can hinder performance and increase the problem complexity, therefore innovative solutions are needed to deal with multiple and complex aims.

In this paper, we investigate novel, cost-aware Virtual Machine (VM) consolidation methods for cloud datacentres using the DISSECT-CF simulator [Kecskemeti, 2015], which is a generic tool for investigating infrastructure clouds. We discuss how to introduce cost models to the DISSECT-CF simulator to investigate cost efficient datacentre consolidation techniques. We also propose 7 different strategies to perform VM consolidation, and present how to use the extended simulator by performing cost-aware datacentre consolidation by evaluating these algorithms. We apply real world traces to simulate cloud load during the experiments.

The structure of the paper is the following. First, in Section 2, we discuss state of the art approaches in the field. In Section 3, we introduce the cost models applied in the DISSECT-CF simulator. Section 5 discusses our experiments and measurement results achieved with the extended simulator. Finally, Section 6 concludes our work.

## 2 Related Work

Ahmad et al. presented a survey on datacenter consolidation solutions in [Ahmad et al., 2015]. They argued that virtual machine (VM) migration and dynamic voltage frequency scaling (DVFS) methods are generally used to achieve server consolidation, which help to achieve resource management goals like load balancing and power management, though it also affects application performance. They concluded the survey that the unpredictable nature of workloads and the inability

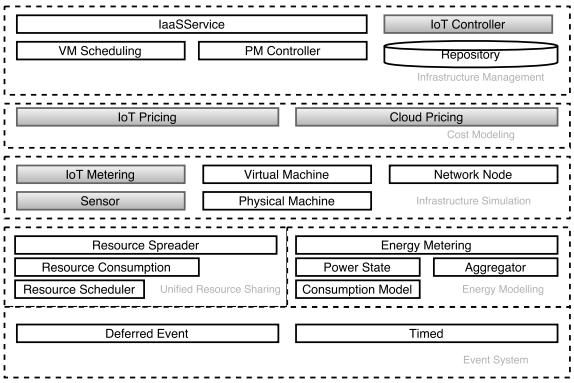


Figure 1: The architecture of the DISSECT-CF simulator

to accurately predict application demands call for dynamic, lightweight and adaptive VM migration designs to improve application performance. Filho et al. [Filho et al., 2018] published another survey in this field. They state that VM placement and migration are the major challenging issues in management of virtualized datacenters, and many proposals apply different approaches ranging from linear programming, to genetic algorithms. They also showed that multi-objective proposals can reduce performance and increase the problem complexity, therefore innovative solutions are needed to deal with multiple and complex aims. Our proposed simulation environment aims at providing a way for investigating certain policies to achieve these goals.

Concerning simulations, the CloudSim toolkit has been widely used to propose and evaluate certain heuristics for datacenter consolidation, such as in [Abdullah et al., 2017] and [Kertesz et al., 2016]. Though these solutions provide load balance improvements, they do not take into account and do not apply provider pricing.

# 3 Our Proposed Cost Model for Cloud Datacentre Management

DISSECT-CF is a compact open source [DISSECT-CF, 2017] simulator focusing on the internals of IaaS systems. Figure 1 presents its architecture including our extensions (denoted with grey colour). There are six subsystems (encircled with dashed lines) implemented, each responsible for a particular functionality: (i) event system – the primary time reference; (ii) unified resource sharing – models low-level resource bottlenecks; (iii) energy modelling – for the analysis of energy-usage patterns of resources (e.g., NICs, CPUs) or their aggregations; (iv) infrastructure simulation – for physical/virtual machines, sensors and networking; (v) cost modelling – for managing IoT and cloud provider pricing schemes, and (vi) infrastructure management – provides a cloud like API, cloud level scheduling, and IoT system monitoring and management.

In a recent work [Markus et al., 2017], we introduced the following new components to model IoT Cloud systems: Sensor, IoT Metering and IoT Controller. Sensors are essential parts of IoT systems, and usually they are passive entities (actuators could change their surrounding environment though). Their performance is limited by their network gateway's (i.e., the device which polls for the measurements and sends them away) connectivity and maximum update frequency. Our network gateway model builds on DISSECT-CF's already existing Network Node model, which allows changes in connection quality as well. In our model, the Sensor component is used to define the sensor type, properties and connections to a cloud system. IoT Metering is used to define and characterize messages coming from sensors, and the IoT Controller is used for sensor creation and management.

To incorporate cost management, we enabled defining and applying provider pricing schemes both for IoT and cloud part of the simulated environments. These schemes are managed by the IoT and Cloud Pricing components of the Cost modeling subsystem of DISSECT-CF, as shown in Figure 1.

## 3.1 Configurable Cost Models based on Real Provider Schemes

In order to enable realistic datacentre consolidation simulations, we considered four of the most popular, commercial cloud providers, namely: Amazon, MS Azure, IBM Bluemix and Oracle. Most providers have a simple pricing method for VM management (beside thaditional virtual machines, some provide containers, compute services or application instances for similar purposes). The pricing scheme of these providers can be found on their websites. We considered the Azure's application service [Azure, 2017], the Bluemix's runtime pricing sheet under the Runtimes section [IBM, 2017], the Amazon EC2 On-

Demand prices [Amazon, 2017], and the Oracle's compute service [Oracle, 2017] together with the Metered Services pricing calculator [Oracle Calculator, 2017]. The cloud-related cost is based on either instance prices (Azure and Oracle), hourly prices (Amazon) or the mix of the two (Bluemix) provider uses both type of price calculating. For example, Oracle charges depending on the daily uptime of our application as well as the number of CPU cores used by our VMs.

```
<cloudproviders>
 <amazon>
  <medium>
    <ram>8589934592</ram>
    <cpucores>2</cpucores>
    <instance-price>18.15</instance-price>
    <hour-per-price>0.094</hour-per-price>
  </medium>
 </amazon>
 <oracle>
  <medium>
    <ram>16106127360</ram>
    <cpucores>2</cpucores>
    <instance-price>139</instance-price>
    <hour-per-price>0</hour-per-price>
  </medium>
  <large>
    <ram>16106127360</ram>
    <cpucores>4</cpucores>
    <instance-price>268</instance-price>
    <hour-per-price>0</hour-per-price>
  </large>
 </oracle>
 <bluemix>
  <large>
    <ram>4294967296</ram>
    <cpucores>8</cpucores>
    <instance-price>0</instance-price>
    <hour-per-price>0.296</hour-per-price>
  </large>
 </bluemix>
</cloudproviders>
```

Figure 2: Cost model of Cloud providers

Figure 2 shows the XML structure and the cost values for the applied categories we designed to be used in the simulator. This configuration file contains some providers (for example the amazon element starting in the second line), and the defined values are based on the gathered information from the providers' public websites discussed before. We specified 3 different sizes for applicable VMs (named small, medium and large).

This XML file has to contain at least that size category to be used for the experiments. As we can see from the fourth line to the seventh line, a category defines a virtual machine with the given ram and cpucores attributes, and we state the virtual machine prices with the instance-price and hour-per-price attributes. If we select the amazon provider with small category, then in the scenarios a virtual machine will have 1 CPU core and 2 GB of RAM, and the usage of this virtual machine will cost 0.296 Euro per hour.

## 4 Consolidator algorithms

Data-centre consolidation techniques are heavily used in commercial clouds. Consolidation is built on the migration capabilities of virtual machines, where virtualised workload is moved around in the data-centre according to the cloud operator's goals. In the past years, there were several approaches proposed for consolidating the virtualised workloads of clouds. Most of them were evaluated with simulations. When analysing cost models, the effects of consolidation could not be avoided. Although, the foundations for these consolidator algorithms were laid down in our DISSECT-CF simulator from the beginning [Kecskemeti, 2015]. Even with the addition of more precise live-migration modelling [Maio et al., 2016], the consolidation algorithms were not present in the simulator.

There are two distinct approaches possible to implement a consolidation algorithm in DISSECT-CF: (*i*) create an alternative physical machine (PM) controller which utilizes consolidator related techniques as well or (*ii*) create an independent consolidator which builds on top of the other infrastructure management components of the simulator. While both approaches could apply the same policies and enact the same goals of a cloud provider, they should be implemented differently. In the first case, the PM controller should extend its possible actions from switching on/off PMs to migrating VMs as well. In the second case, the consolidator is dedicated to only decide on migration related actions. This is beneficial as the consolidator algorithm could collaborate with multiple PM controller strategies without the need for a complete rewrite of the consolidation approach. As this second approach is more generic, thus we present it in this paper in more detail. Note, the source of the presented approach is publicly available in the source repository of DISSECT-CF [DISSECT-CF, 2017].

Figure 3 shows how the extension was implemented. The main addition of the simulator is

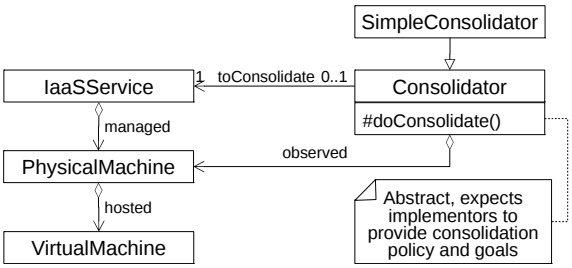

Figure 3: Consolidation related extension of DISSECT-CF

the `Consolidator` class, which is to be extended by any new consolidation policies in the future. This abstract class handles the basic connection of the future consolidators to the `IaaSService` by monitoring the VM related activities on the cloud. It is also responsible for managing the frequency with which the consolidation policy is run (to be implemented by third parties in the `doConsolidation()` function). In general, it ensures that the custom consolidator policy is only invoked if there are any VMs in the cloud at any particular time. To do so, the consolidator monitors the PMs and observes how they are managed by PM controllers and utilised by the VM schedulers.

The simulator also offers a consolidation policy called `SimpleConsolidator`. This policy packs the VMs to the smallest amount of PMs as follows.

1. Creates an ordered list ($P := \{p_1, p_2, ...p_n\}$) of the PMs (e.g., $p_1$) currently running in the IaaS (where the number of running PMs in the IaaS is $n$). This list has the least used PMs in the front and the heaviest used ones at the tail: $u(p_1) \leq u(p_2) \leq ... \leq u(p_n)$. Where we denote the utilisation of a PM with the function : $u : P \rightarrow \mathbb{R}$. Note: the utilisation is determined solely on the resource allocations for the VMs hosted on each PM and it is not dependent on the instantaneous resource usage of any of the VMs in the cloud.
2. Picks the least used not yet evaluated PM ($p_i$). If there are no more PMs to evaluate, we terminate the algorithm.
3. Picks a VM ($v_x$) hosted by $p_i$. Where $v_x \in h(p_i)$ and the function $h : P \rightarrow 2^V$ defines the set of VMs which are hosted by a particular PM. This set is a subset of all VMs ($V$) in the IaaS service.
4. Picks the heaviest used (but not completely utilised) and not yet tested PM ($p_k$). Where we have the following limits for $k$: $i+1 \leq k \leq n$.
5. Checks if the new PM has enough free resources to host the VM: $r_f(p_k, t) \geq r(v_x, t)$, where $r_f : P \times \mathbb{R} \rightarrow \mathbb{R}^3$ and $r : V \times \mathbb{R} \rightarrow \mathbb{R}^3$. The $r_f$ function tells the amount of free resources available at the specified host at the specified time instance $t$. Also, the $r$ function tells the amount of resources needed by the virtual machine at the specified time instance. The resource set is modelled by a triplet of real numbers: ($i$) number of CPU cores, ($ii$) per core processing power and finally ($iii$) memory.
   - If the check was successful, then the VM is requested to be migrated from the host $p_i$ to $p_k$. Then continue on with a new VM pick.
   - If the check fails, we repeat with all untested PMs. If no more PMs are around to test, we pick another VM from the list of $h(p_i)$. If there are no more VMs to pick, we return to step 2.

Thus we can summarize the algorithm as packing the VMs to the heaviest loaded PMs with a first fit approach. This approach is efficient with the PM controller called `SchedulingDependentMachines` which switches off all unused machines once they become freed up (in this case once all their VMs migrate away).

## 5 Evaluation

During our implementation and evaluation, where applicable, we used publicly available information to populate our experiments. In the next subsection we introduce the applied workloads, then discuss the proposed algorithms and scenarios, and the achieved results.

### 5.1 Workloads

Though virtual machine management log-based traces would be the best candidates for analysing cloud characteristics, traces collected from other large-scale infrastructures like grids could also be appropriate. Generally two main sources are used for this purpose: the Grid Workloads Archive (GWA [GWA, 2017]) and the Parallel Workloads Archive [PWA, 2017]. For this study we used traces downloadable from GWA (namely: Au-

verGrid, DAS2, Grid5000, LCG, NorduGrid and SharcNet).

We used the `JobDispatchingDemo` from the DISSECT-CF examples project[1], to transform the jobs listed in the trace to VM requests and VM activities. This dispatcher asks the simulator to fire an event every time when the loaded trace prescribes. Also, the dispatcher maintains a list of VMs available to serve job related activities (e.g., input & output data transfers, cpu and memory resource use). Initially the VM list is empty. Thus the job arrival event is handled with two approaches: $(i)$ if there is no unused VM in the VM list that has sufficient resources for the prescribed job, then the dispatcher creates a VM according to the resource requirements of the job; alternatively, $(ii)$ if there is an unused VM with sufficient resources for the job, then the job is just assigned to the VM. In the first approach, the job's execution is delayed until its corresponding VM is spawned. In both cases, when the job finishes, it marks the VM as unused. This step allows other jobs to reuse VMs pooled in the VM list. Finally, the VMs are not kept for indefinite periods of time, instead they are kept in accordance with the billing period applied by the cloud provider. This ensures, that the VMs are held for as long as we paid for them but not any longer. So if there is no suitable job coming for a VM within its billing period, then the VM is terminated and it is also removed from the VM list.

## 5.2 Scenarios

In the following we list the pricing strategies available at the moment. They are applicable alone or in combination as required.

**S1 - Fixed pricing.** It uses a constant price for every VM request. This pricing strategy does not consider any factors in its price:

$$M_{fix} = m_c, \tag{1}$$

where $M_{fix}$ is the price (i.e money) returned, and $m_c$ is the constant base price which is configurable for every simulation.

**S2 - Resource constraints aware pricing.** It implements a linear relationship between the price of a VM and the amount of resources the VM needs. The higher the resource needs are, the more the user should pay.

$$M_{rcaw}(r_{cores}, r_{mem}, r_{proc}) = m_c \frac{r_{cpu} * r_{proc} * r_{mem}}{r_{cpu}^{MAX} * r_{proc}^{MAX} * r_{mem}^{MAX}}, \tag{2}$$

where the triple $< r_{cores}, r_{mem}, r_{proc} >$ represents the resources requested by the customer for its VM. The triple $< r_{cpu}^{MAX} * r_{proc}^{MAX} * r_{mem}^{MAX} >$ represents the properties of the largest resource amount any PM has in the cloud provider. Note that all the resource values are represented as the provider sees them fit, for the purpose of the paper we assumed they are all positive real numbers (e.g., $r_{cores} \in \mathbb{R}^+$). Thus, this pricing model, charges $m_c$ if the user requests the largest still serviceable resource set.

**S3 - Quantized pricing.** It applies a pricing strategy similar to $M_{rcaw}$. But instead of scaling the price by a continuous function, we apply a transformation which transforms ($\mathfrak{T} : \mathbb{R}^3 \to \mathbb{R}^3$) the original request from the user to some preset values. When defining a quantized pricing, one must define this transformation only, then we can apply the $M_{rcaw}$ model to find out the actual price.

$$M_{quant}(r_{cores}, r_{mem}, r_{proc}) = M_{rcaw}(\mathfrak{T}(r_{cores}, r_{mem}, r_{proc})) \tag{3}$$

This is the technique that is used by most of cloud providers nowadays. In those cases, the providers are often restricting the amount of resources one can request as well. An example transformation function could be:

$$\mathfrak{T}_{ex} = \begin{cases} \text{if } r_{mem} <= 2 \wedge r_{cores} <= 1 \\ \quad r'_{mem} = 2, r'_{cores} = 1, r_{proc} = 1 \\ \text{if } 2 < r_{mem} <= 8 \wedge 1 < r_{cores} <= 2 \\ \quad r'_{mem} = 8, r'_{cores} = 2, r_{proc} = 1 \\ \text{otherwise} \\ \quad r'_{mem} = 32, r'_{cores} = 8, r_{proc} = 1 \end{cases} \tag{4}$$

The simulator implements a pricing model which can be configured to load a particular transformation function for a particular cloud provider. The limits for the transformation functions are stored in an XML file representing certain commercial provider cost models. Later in the measurements we apply the cost model presented in Section 3.

**S4 - PM Utilization aware pricing.** This strategy also offers a linear pricing approach.

[1] `https://github.com/kecskemeti/dissect-cf-examples`

In contrast to the resource constraints aware pricing model, this time, we adjust the price based on the number of PMs in use at cloud provider:

$$M_{utaw} = m_c \frac{|P_U|}{|P|}, \tag{5}$$

where the $P_U$ is the set of PMs that host any VMs: $P_U = \{\forall p_x \in P : u(p_x) \neq 0\}$. Thus the more exploited the cloud provider is, the more the user should pay.

**S5 - Load dependent pricing.** This works similarly to the PM utilization aware pricing. At the cost of additional monitoring requirements, it implements the same policy with a more fine grained utilization calculation:

$$M_{ld} = m_c \frac{\sum_{\forall p \in P} R(p)}{\sum_{\forall p \in P} R^{MAX}(p)}, \tag{6}$$

where $R(p)$ represents the average amount of resources utilised in the last hour from particular physical machine, while $R^{MAX}(p)$ defines the total amount of resources the PM could offer in the same hour. Thus, this pricing model considers how well the VMs actually use the resources and if the VMs are not highly used (even though they are hosted at the cloud at the moment), then the prices will be lowered (this will attract further users and enable the provider to use under provisioning for those VMs that are just paid for but not used at the moment).

**S6 - Reliability aware pricing.** It alters the price based on the ratio of successfully and unsuccessfully hosted VMs at the cloud. A VM is classified unsuccessfully hosted if it is terminated because of a physical machine failure, and not because of a user's request.

$$M_{rel} = m_c \frac{|V_f|}{|V|}, \tag{7}$$

where $V_f$ is the set of VMs which failed due to a hardware issue at the provider side.

**S7 - Profit margin focused pricing.** It tries to price resources so the profit margin index ($i$) of the cloud provider stays in a predefined range (i.e. $I_{min} < i < I_{max}$).

$$M_{margin}(t_0) = m_c \tag{8}$$

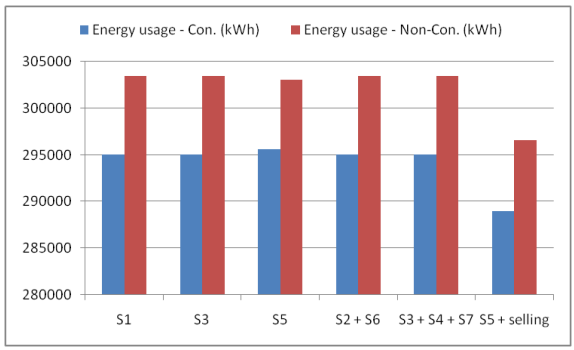


Figure 4: Energy consumption of experiments with the Grid5000 trace

$$M_{margin}(t_x) = M_{margin}(t_{x-1}) \cdot \begin{cases} 0.9, & \text{if } i(t_x) < I_{min} \\ 1.1, & \text{if } i(t_x) > I_{max} \\ 1, & \text{otherwise} \end{cases}, \tag{9}$$

where the function $i(t)$ determines the current (or at a given time, represented by $t$) profit margin index of a provider. This technique tries to adopt the prices to make sure the provider is profitable even in competitive environments.

## 5.3 Results

As mentioned before, we investigated how policies considering pricing information can affect consolidation processes. We used 6 different trace files from real world distributed systems to simulate load on the cloud datacentres we aim to consolidate. We also designed 7 different strategies to perform cost-aware consolidation. In overall, the consolidation algorithms succeeded to balance the load over the system, and in most cases energy and money can be saved by applying them. In the following we highlight the most interesting results.

We have performed numerous experiments by executing the above listed strategies for all previously mentioned trace files. Concerning experiments run on the *Grid5000* load, Figure 4 and Figure 5 depict the tradeoff of energy gains and runtime expansions for the given strategies ("S2 + S6" means we applied both strategies, "S5 + selling" means we applied the S5 strategy and sold the shut down PMs to gain money). By migrating certain VMs to other physical machines to balance the load, we managed to reduce the

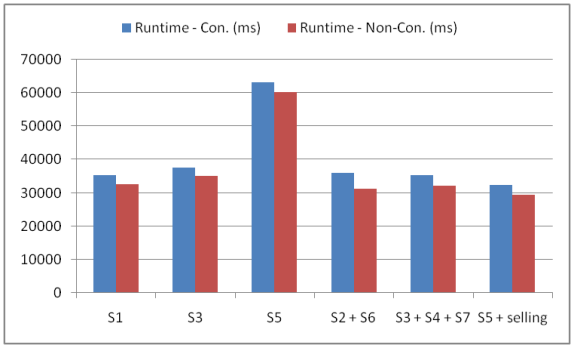


Figure 5: Runtime of experiments with the Grid5000 trace

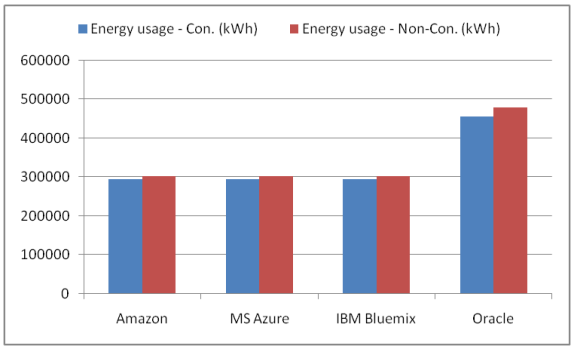


Figure 6: Energy consumption of experiments with Grid5000 for different cloud provider pricing with the S3 strategy

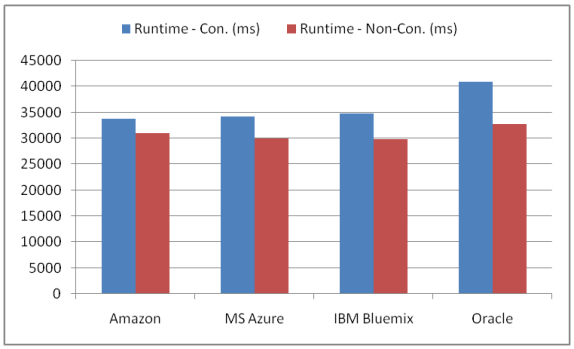


Figure 7: Runtime of experiments with Grid5000 for different cloud provider pricing with the S3 strategy

power consumption, however the migration processes took some time which appears in the overall runtime. From the results we can see that the S6 strategy is the most efficient for reducing power consumption, and still it is the fastest solution.

Figure 6 and Figure 7 depicts the results of our S3 strategy that enables to load and apply different provider pricing schemes. From these results we can see that the highest energy gains could be achieved with the Amazon pricing scheme for this load condition, while the worst result came from applying the Oracle pricing.

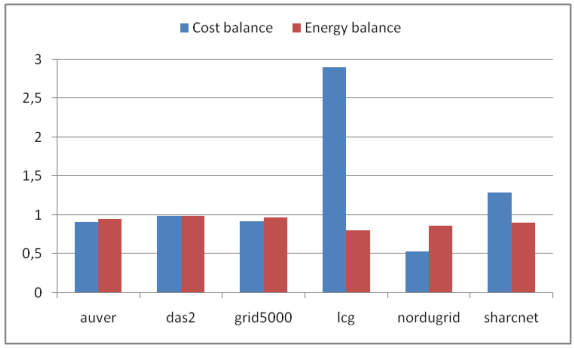


Figure 8: Load and energy balance for different load conditions with the S2 + S6 strategy

We also experienced that the load types represented by the traces highly affect the results. Figure 8 presents measurements performed under different load conditions with the combined S2 + S6 strategy. The depicted balance represents the possible gains of using consolidation in terms of cost (i.e. money) and energy.

## 6 Conclusions

In this paper we addressed the problem of datacentre consolidation. Though several approaches have been developed to improve the utilization efficiency of datacentres, only a few attention has been turned to combining pricing methods with consolidation techniques.

We presented an extension of the DISSECT-CF simulator to foster the development of cost efficient datacentre consolidation solutions. We also showed how to apply real world traces to simulate cloud load, and proposed 7 different cost-based strategies to address the problem. Our results have approved that cost-aware datacentre consolidation is a valid approach and can result in significant cost and energy gains.

## Acknowledgements

The research leading to these results was supported by the UNKP-17-4 New National Excellence Program of the Ministry of Human Capacities of Hungary, and by the Hungarian Government and the European Regional Development Fund under the grant number GINOP-2.3.2-15-2016-00037 ("Internet of Living Things").

## REFERENCES


[Amazon, 2017] Amazon pricing website. Online: https://aws.amazon.com/ec2/pricing/on-demand/.

[DISSECT-CF, 2017] DISSECT-CF website. Online: https://github.com/kecskemeti/dissect-cf. Accessed at January, 2017.

[GWA, 2017] Grid Workloads Archive website. Online: http://gwa.ewi.tudelft.nl/. Accessed at September, 2017.

[IBM, 2017] IBM Bluemix pricing sheet. Online: https://www.ibm.com/cloud-computing/bluemix. Accessed at January, 2017.

[Azure, 2017] MS Azure price calculator. Online: https://azure.microsoft.com/en-gb/pricing/calculator/. Accessed at January, 2017.

[Oracle, 2017] Oracle pricing website. Online: https://cloud.oracle.com/en_US/opc/compute/compute/pricing. Accessed at January, 2017.

[Oracle Calculator, 2017] Orcale Metered Services pricing calculator. Online: https://shop.oracle.com/cloudstore/index.html?product=compute. Accessed at January, 2017.

[PWA, 2017] Parallel Workloads Archive website. Online: http://www.cs.huji.ac.il/labs/parallel/workload/. Accessed at September, 2017.

[Abdullah et al., 2017] Abdullah, M., Lu, K., Wieder, P., and Yahyapour, R. (2017). A heuristic-based approach for dynamic vms consolidation in cloud data centers. *Arabian Journal for Science and Engineering*, 42(8):3535–3549.

[Ahmad et al., 2015] Ahmad, R. W., Gani, A., Hamid, S. H. A., Shiraz, M., Yousafzai, A., and Xia, F. (2015). A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *Journal of Network and Computer Applications*, 52(Supplement C):11 – 25.

[Filho et al., 2018] Filho, M. C. S., Monteiro, C. C., Inacio, P. R., and Freire, M. M. (2018). Approaches for optimizing virtual machine placement and migration in cloud environments: A survey. *Journal of Parallel and Distributed Computing*, 111(Supplement C):222 – 250.

[Kecskemeti, 2015] Kecskemeti, G. (2015). DISSECT-CF: a simulator to foster energy-aware scheduling in infrastructure clouds. *Simulation Modelling Practice and Theory*, 58P2:188–218.

[Kertesz et al., 2016] Kertesz, A., Dombi, J. D., and Benyi, A. (2016). A pliant-based virtual machine scheduling solution to improve the energy efficiency of iaas clouds. *Journal of Grid Computing*, 14(1):41–53.

[Maio et al., 2016] Maio, V. D., Kecskemeti, G., and Prodan, R. (2016). An improved model for live migration in data centre simulators. In *2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)*, pages 108–117.

[Markus et al., 2017] Markus, A., Kertesz, A., and Kecskemeti, G. (2017). Cost-aware iot extension of dissect-cf. *Future Internet*, 9(3).