Towards a SLA-compliant Cloud Resource Allocator for N-tier Applications

Aaron McConnell, Gerard Parr, Sally McClean, Philip Morrow and Bryan Scotney
School of Computing and Information Engineering
University of Ulster
Coleraine, Northern Ireland
Email: a.mcconnell@ulster.ac.uk, gp.parr@ulster.ac.uk, si.mclean@ulster.ac.uk, pj.morrow@ulster.ac.uk, bw.scotney@ulster.ac.uk

***Abstract*—Cloud vendors commonly offer users IaaS where virtual machines (VMs) can be created and run on cloud resources. The resource allocation for each VM is defined by the user and the VM is created on a physical machine (PM) where ample resources exist to support the VM's operation at its maximum capacity. There are a number of opportunities for improvement when allocating host resources to VMs. VM-resident applications are often n-tier, with different VMs responsible for parts of the distributed application. It may be important that these VMs are placed within a given network proximity to one another. The network proximity to the user may also be an issue for some applications. Resource allocation to VMs should also be such that, rather than a user over-provisioning the VM, the VM's minimal operational requirements are specified so that the VM can be resource-throttled at times of heavy load. This paper presents an outline for a system called Innkeeper, which aims to allocate resources to a VM in a way that ensures the VM will always function adequately, but where the VM is not over-provisioned. Innkeeper also aims to place VMs so that a VM "family" are kept within a necessary network proximity to one another and where the proximity to the user is also considered when placing VMs.**

Keywords-cloud computing; resource allocation; virtualisation.

I. INTRODUCTION

Cloud Computing offers virtualised data centre resources to users as a service. Organisations can use cloud platforms to outsource their IT infrastructure, resulting in reduced Capital Expenditure (CAPEX) and Operating Expediture (OPEX) and dynamically scaling capacity. Cloud providers offer pay-per-use pricing schemes where users pay an amount to reserve certain resources and a further amount for the amount of a resource used, e.g., bandwidth. Allocating physical host resources to users and their VMs is carried out on-the-fly and is often achieved using greedy algorithms. It is necessary for a set of operational constraints to be established before any optimisation method can be successful. PMs have finite sets of resources and VMs have operational requirements in terms of resources. A VM should only be placed on a physical host where the physical host has sufficient resources to satisfy the demands of all resident VMs. VMs also have a set of network requirements that are often ignored when defining a VM's resource needs. Cloud applications are often n-tier applications [1], with one element of the application residing within a separate VM from other elements. The network distance (in terms of bandwidth and latency) between application elements should be short enough that the entire application functions to the user's requirements. This is especially a concern in the situation where a cloud provider has resources in more than one geographic location, or where the user migrates part of his organisation's application to the cloud, whilst retaining other parts within the organisation.

The work presented in this short paper is at an early stage and is ongoing. The proposed solution provides a system, which 1) provides a Service Level Agreement (SLA) framework for n-tier cloud applications, 2) provides an automated scalable, three-tiered approach for assessing the suitability of distributed resources for VM placement, 3) considers network links between all application entities and the user.

The next section discusses related work, followed by a description of the proposed model and its design. Results from some initial experiments are presented in section IV.

II. RELATED WORK

Existing literature details that cloud computing offers different models for VM consumption in cloud environments [2], reservation, on-demand and spot market [3], with each model catering for a different need. Placing VMs in each model requires that a decision is made about placing the VM on an appropriate host [4], [5] or migrating a VM from one host to another in order to provide some kind of optimisation (e.g., performance increase, financial cost saving). The current state-of-the-art focuses heavily on optimisation of VM placement with various types of focus on different constraints; there are systems aimed at cloud resource provisioning for existing VMs [6], [7], [8], but there is no work on providing an open, scalable platform for assessing host, cluster and cloud capability, particularly for SLA-compliant n-tier VM placement.

III. MODEL DESCRIPTION

Innkeeper is designed to provide three brokers, one for each host, one for each cluster of hosts and one for the cloud of clusters (see Fig. 1). The function of each broker is as follows:

a) Host Innkeeper: Each Host Innkeeper (HIk) responds to a request for VM placement with either *accept* or *refuse* depending on whether or not the host can accommodate the VM. This decision is based upon a VM SLA, which defines

 ISBN: 978-1-61208-216-5

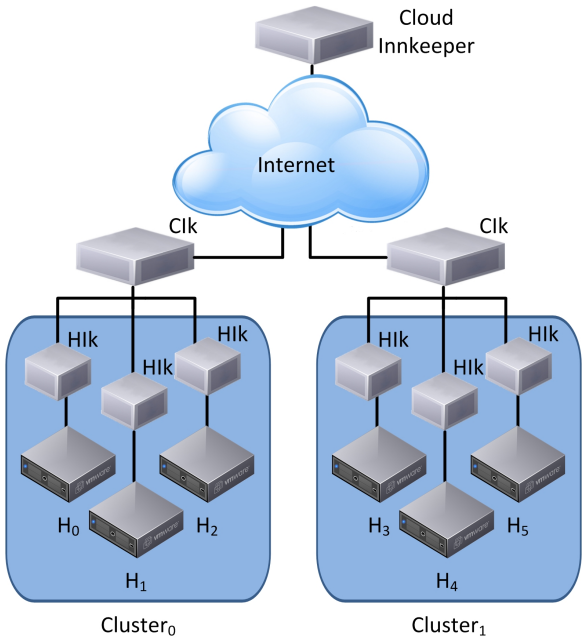

Fig. 1. The Three-tier Innkeeper structure

metric thresholds for minimal VM performance, e.g., a VM may require a minimum of 800 Mhz of CPU share and 1 GB of virtualised memory in order to perform adequately. For each metric, the HIk subtracts the SLA value from the currently available resources for the host, e.g., if the host has 4 GB of memory remaining and the VM's SLA is requesting 1 GB then the HIk would calculate that accepting the VM means 3 GB of free memory would remain for the host. The HIk would define host-level thresholds as well, meaning that it would never be the case that 100% of any host-level metric is ever consumed. In the case that some metric request in a to-be-placed VM caused the host to consume levels of that metric at host level, then the VM would not be placed and a *refuse* response would be generated.

b) Cluster Innkeeper: Each Cluster Innkeeper (CIk) acts as a placement broker at the cluster level. The CIk aims to place VMs on hosts, in the cluster it is responsible for, by interfacing with each HIk and querying whether or not the host can accept a VM with a given SLA. Three outcomes are possible when a CIk attempts to place a VM. Either one of the HIks accepts the VM, more than one HIk accepts the VM or none of the HIks accepts the VM. The first case is straight-forward and the VM is placed via the accepting HIk. In the second scenario, where there are multiple hosts on, which the VM may be placed, there is room for employing some intelligence when choosing which host to place the VM with, e.g., one host may offer more of a desired resource than another. The third scenario, where no hosts can accommodate the VM, presents a situation where either another CIk is used or an optimisation is attempted in order better place existing VMs and free up capacity so that a new VM can be added to a host. This type of optimisation is discussed in sub-section III-B. CIks also provide knowledge about the network proximity, in terms of bandwidth and latency, between CIks. This is necessary in order to ensure that VMs are placed within adequate proximity to other VMs they communicate heavily with, and with the end-user, as defined in the VM's SLA.

c) Cloud Innkeeper: The Cloud Innkeeper (CLIk) is a central system, which acts as a broker for the entire cloud. The CLIk is presented with a user's VM SLA and attempts to place the VM on a cluster by interfacing with each CIk. There are again three possible scenarios, one CIk can accommodate that VM, multiple CIks can accommodate the VM, or no CIks can accommodate the VM. The second scenario again presents an opportunity to place the VM at a host where some benefit may be had over placement at other hosts, e.g., an important resource is more abundant. The third scenario, where no CIks can accommodate the VM, presents a further opportunity to optimise the placement of existing VMs at one or more CIk. It is also possible that the CLIk will be presented with a n-tier set of VMs to place, each with constraints regarding the network proximity to others. An optimisation problem is created in this instance, which may be solved with the implementation of a greedy algorithm.

These brokers provide a highly dynamic and scalable hierarchy for SLA-compliant VM placement. A common Application Programming Interface (API) is shared between the HIk, CIk and CLIk as shown in Table I. An API relevant to network metrics is unique only to CIks, where bandwidth and latency values between a queried CIk and another address, e.g., another CIk or an end-user I.P. address, are returned.

A. Monitoring

Host and VM monitoring is carried out by accessing the monitoring Web services of the underlying virtualisation platform. This monitoring is carried out by each HIk. CIks and the CLIk access monitoring information via the HIk API. Network monitoring, of the links between CIks and with end-users, is carried out by using Bwping [9] to acquire bandwidth and monitoring statistics. Host and VM real-time monitoring statistics are stored in a database within each HIk with network monitoring statistics stored with each CIk.

B. Optimisation

Optimisation opportunities exist when a scenario occurs where a HIk, CIk or the CLIk cannot accommodate a new VM. It may be possible to free up resources at a host by attempting to reconfigure the hosting of existing VMs so that the overall capacity utilisation of a given host is higher while not breaching utilisation thresholds for any given host-level metric. This type of optimisation can be viewed as the Multi-objective Knapsack Problem (MKP), which is a combinatorial optimisation problem [10], [11], [12]. The MKP requires that a compromise or trade-off is made when considering multiple optimisational sub-objectives. Therefore MKP cannot guarantee an optimal solution for each sub-objective. Chu et Beasley [13] formulate the MKP as:

 ISBN: 978-1-61208-216-5

TABLE I
THE INNKEEPER COMMON API

Interface	Description
GetAggCPU (Return Integer)	Returns the aggregated CPU capacity for the broker's hosts (MHz)
GetAggMem (Return Integer)	Returns the aggregated memory capacity for the broker's hosts (MB)
GetAggDisk (Return Integer)	Returns the aggregated disk capacity for the broker's hosts (GB)
GetMaxCPU (Return Integer)	Returns the single largest amount of CPU available at a host (MHz)
GetMaxMem (Return Integer)	Returns the single largest amount of Memory available at a host (MB)
GetVmAllocation (Return Integer)	Returns the current allocation of VMs for the broker's hosts
CanHostVM(VMSLA *sla) (Return Boolean)	Is passed VM's SLA and returns true or false indicating the ability to host the VM
AddVM(VMID *vmId) (Return Boolean)	Is passed an ID for the VM so an attempt can be made to place it at the appropriate host
RemoveVM(VMID *vmId) (Return Boolean)	Is passed an ID for the VM so an attempt can be made to remove the VM from cloud/cluster/host

$$\text{maximise} \sum_{j=1}^{n} p_j \, x_j,$$

$$\text{subject to} \sum_{j=1}^{n} r_{ij} \, x_j \leq b_i, \quad i = 1, ..., m,$$

$$x_j \in \{0, 1\}, \quad j = 1, ..., n,$$

where each of the m constraints is defined as a *knapsack* constraint. The MKP attempts to place a subset of n items such that the total value of all items is as high as possible within given constaints. The value of each item is defined as p, with x defining whether or not item p_j is placed (x is assigned a value of 1 if the item is placed and 0 if it is not). The constraint value for p_j is r_{ij}, with i referring to a given constraint, e.g., for placing a VM it might be CPU or memory. The total constraint value for the sum of constraints for all placed items must not exceed the maximum allowed value for that constraint b_i. This problem is often solved using a greedy algorithm, e.g., a genetic algorithm.

IV. INITIAL RESULTS

Some initial experimentation was carried out in order to assess host metric utilisation as VMs are placed on a host. These experiments were carried out using a Fujitsu Siemens Celsius R550 servers, with two Intel Xenon E5440 processors (2.8 GHz, 6 MB L2 cache) and with 8 GB of main memory. This host ran VMware ESX 4.0 [14] and was used to host VMs, containing a standard Joomla v1.5.20 Web Server [15], each with a resource allocation of 2 virtual CPUs, 512 MB of main memory and an 8 GB thin-provisioned virtual hard drive. Load was placed on each VM's application (two page requests per second) using OpenLoad [16] on the Web Server over port 80 (the standard HTTP port). Fig. 2 illustrates the resource usage for both the host and the first VM placed. The host's CPU runs at 100% utilisation when four loaded VMs (each one had a VM CPU load of $\approx 95\%$) are placed on it. This host CPU load causes resource contention between VMs. It is interesting to note that, after 3 heavily-loaded VMs are placed on the host, the CPU resource consumption of the monitored VM appears to drop. This decrease in CPU consumption by the VM is because it is starved of access to the underlying

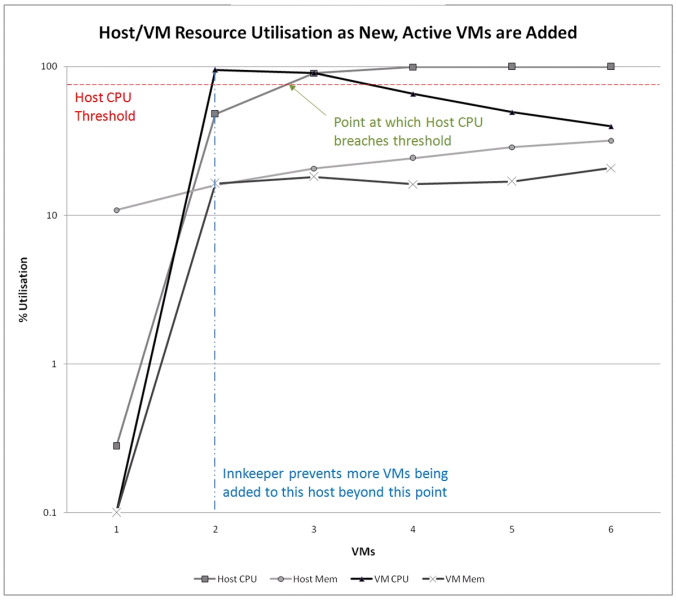

Fig. 2. Host and VM Resource Consumption

host resources and is forced to queue for CPU resources. This creates the illusion, in the reported VM performance metrics, that the VM is not consuming resources due to lack of demand on the VM when in fact the drop in CPU consumption is because the VM does not have the opportunity to consume the host's CPU. This is backed up the graph in Fig. 3 where OpenLoad reports an increasing response time, as the number of VMs on the host is increased, for the monitored VM. The horizontal dashed line on Fig. 2 illustrates a potential host CPU utilisation threshold beyond which no further VMs should be added. With this threshold defined within a HIk, the HIk would prevent further VMs being added beyond the second VM, as illustrated by the vertical dashed line in Fig. 2.

V. CONCLUSIONS AND FUTURE WORK

This model attempts to provide a system for SLA-compliant placement of n-tier VMs in a cloud computing environment. Initial experimentation illustrates a need for such a system, to ensure near-optimal use of distributed cloud resources while enforcing SLA constraints. There is also a need to ensure

 ISBN: 978-1-61208-216-5

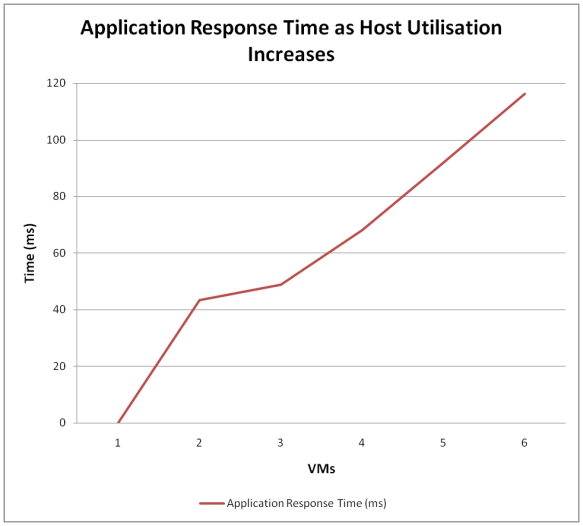

Fig. 3. VM Response Times

that communicating VMs are placed within a relatively close network proximity, in order that they perform adequately. A key requirement for cloud-based systems is the ability to provide scalability. The Innkeeper design offers this scalability and ensures that each of the three tiers is agnostic to the function of the other tiers. This design reduces complexity and provides a relatively simple but powerful means by which hosts can be monitored and VMs placed. One of the key motivators for this work is to create a platform for cloud monitoring and orchestration where intelligent optimisation, with various focuses, e.g., power-saving, reduction in network load, can be easily implemented.

A. Future Work

This body of work presents challenges and opportunities, all of which will be explored in the near future so that a live alpha prototype system is in place before the end of the calendar year. The impending implementation of the prototype system requires that the means of monitoring live VMs, hosts and network links are in place within a cloud test-bed environment. This will most likely be carried out on a VMware test-bed with a number of clusters of industry-standard, multi-core hosts. Network degradation will be created between these clusters (and between clusters and emulated end-users) using WANem [17]. A standardised means of n-tier VM SLA definition remains a challenge, with an associated problem of optimally placing multiple, communicating VMs. An associated issue with n-tier VM placement is that heavy VM load, on existing VMs, may force SLA failure despite the SLA metric thresholds being set for a given amount of maximum load. Failure to ensure that a VM's load does not exceed that for which its SLA is defined may result in the HIk's resource provisioning calculations becoming pointless. A methodology must be developed to ensure that a VM is resource-throttled when its load causes it to consume resources to the extent that resource contention is caused for other VMs on the same host. The other perspective is that this throttling shouldn't be a constant - if host resources are unused then they should be available to VMs, in the hope of increasing the quality of user experience, rather than the host experiencing low utilisation. However, optimisation opportunities exist to place VMs such that, with VM resource-throttling in place, low utilisation on hosts does not occur.

REFERENCES

[1] P. Xiong, Z. Wang, S. Malkowski, Q. Wang, D. Jayasinghe, and C. Pu, "Economical and robust provisioning of n-tier cloud workloads: A multi-level control approach," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, June 2011, pp. 571 –580.

[2] K. Mills, J. Filliben, and C. Dabrowski, "Comparing vm-placement algorithms for on-demand clouds," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, 29 2011-Dec. 1 2011, pp. 91 –98.

[3] I. Fujiwara, K. Aida, and I. Ono, "Applying double-sided combinational auctions to resource allocation in cloud computing," in *Applications and the Internet (SAINT), 2010 10th IEEE/IPSJ International Symposium on*, July 2010, pp. 7 –14.

[4] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *INFOCOM, 2010 Proceedings IEEE*, March 2010, pp. 1 –9.

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

[6] N. Bonvin, T. G. Papaioannou, and K. Aberer, "Autonomic SLA-driven Provisioning for Cloud Applications," *2011 11th IEEEACM International Symposium on Cluster Cloud and Grid Computing*, pp. 434–443, 2011. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5948634

[7] R. N. Calheiros, R. Ranjan, and R. Buyya, "Virtual Machine Provisioning Based on Analytical Performance and QoS in Cloud Computing Environments," *2011 International Conference on Parallel Processing*, pp. 295–304, 2011. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6047198

[8] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek, "Adaptive resource provisioning for read intensive multi-tier applications in the cloud," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 871–879, 2011. [Online]. Available: http://dx.doi.org/10.1016/j.future.2010.10.016

[9] O. Derevenetz, "Bwping - open source bandwidth measurement tool," http://bwping.sourceforge.net/, Jun. 2011, [retrieved: April 2012].

[10] C. Cheng, Y. Huang, Z. Chen, X. Zhang, and J. Xu, "Solving the 0-1 multi-objective knapsack problem using self-assembly of dna tiles," in *Bio-Inspired Computing, 2009. BIC-TA '09. Fourth International Conference on*, Oct. 2009, pp. 1 –9.

[11] Z. Shurong, W. Jihai, and Z. Hongwei, "Multi-population cooperative ga and multi-objective knapsack problem," in *Management and Service Science (MASS), 2010 International Conference on*, Aug. 2010, pp. 1 –4.

[12] D. Vianna and J. Arroyo, "A grasp algorithm for the multi-objective knapsack problem," in *Computer Science Society, 2004. SCCC 2004. 24th International Conference of the Chilean*, Nov. 2004, pp. 69 – 75.

[13] P. C. Chu and J. E. Beasley, "A genetic algorithm for the multidimensional knapsack problem," *Journal of Heuristics*, vol. 4, no. 1, pp. 63–86, Jun. 1998. [Online]. Available: http://dx.doi.org/10.1023/A:1009642405419

[14] Vmware, "Virtualization overview," whitepaper, http://www.vmware.com/pdf/virtualization.pdf, Jan. 2010, [retrieved: April 2012].

[15] Turnkey-Linux, "Joomla 1.5 appliance - cutting edge content management — turnkey linux virtual appliance library," online, http://www.turnkeylinux.org/joomla15, Nov. 2011, [retrieved: April 2012].

[16] P. Johnsen, "Openload," online, http://freecode.com/projects/openload, Jun. 2001, [retrieved: April 2012].

[17] WANem, "Wanem - the wide area network emulator," online, http://wanem.sourceforge.net, Dec. 2009, [retrieved: April 2012].

 ISBN: 978-1-61208-216-5