Optimal virtual machine scheduling with Anvik

Andrea Sansottera
Politecnico di Milano
sansottera@elet.polimi.it

Paolo Cremonesi
Politecnico di Milano
paolo.cremonesi@polimi.it

ABSTRACT

In Infrastructure-as-a-Server (IaaS) clouds, the provider has to decide on which server the virtual machines (VMs) requested by the users should be provisioned. This is an online scheduling problem, in which incoming VMs, typically with different resource requirements, have to be scheduled to one of several heterogeneous servers with limited capacity. For the provider of a public cloud, the objective is to maximize profit, the difference between the operating cost of the servers and the revenue due to running the VMs. In some cases, it might be advantageous to perform VM admission control and reject low-profit VM if low-cost servers are unavailable. We model this problem as a continuous-time Markov decision process and present a tool, *anvik*, for the computation of the optimal scheduling and admission control policy. *Anvik* is released as open-source.

Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: Optimization; C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Algorithms, Design, Economics

1. INTRODUCTION

In this paper we formulate a continuous-time Markov Decision Process (MDP) to determine the optimal policy for scheduling and admission control of virtual machines in IaaS clouds. The policy is a table that maps the system state with actions to take upon VM arrivals (i.e., rejection or admission to a specific server). For a given state, the policy might suggest to take different actions based on the VM characteristics (e.g., resource consumption, revenue). To define instances of the MDP and efficiently compute the optimal policy we present a newly developed tool, *anvik*, that we make freely available under an open-source license[1].

[1]https://github.com/asansottera/anvik

ValueTools'13, December 10 – 12 2013, Turin, Italy

Anvik helps to answer the following questions, all relevant to the management of cloud infrastructure: (i) what is the optimal policy for allocating VMs of different classes on the available, possibly heterogeneous, servers? (ii) which type of servers should be provisioned in order to improve profit? (iii) how will my revenue change if the arrival rate for a given class of VMs increases or decreases? (iv) will the revenue increase by introducing a different class of VMs?

The paper is organized has follows. The model is presented in Section 3. Related work is covered in Section 2. The tool is described in Section 4. In Section 5, we discuss some scenarios to assess the correctness and the performance of the implementation. We draw conclusions in Section 6.

2. RELATED WORK

A broad and interesting class of dynamic resource allocation problems, with direct applications in cloud computing, is studied in [2]. The paper does not consider the constraints and requirements of each individual server and virtual machine. The work presented in [7] addresses the problem of finding an optimal placement of VMs between private and public clouds. An heuristic for the deployment of VMs in a cloud SaaS infrastructure is presented in [5]. [4] focuses on minimizing the energy cost associated with migration of VMs in cloud environments. The problem of optimal allocation of VMs across different public cloud providers is presented in [1]. The work presented in [6] describes an evaluation methodology for the comparison of different VM placement algorithms in cloud infrastructures.

3. MODEL

In this section we describe the proposed virtual-machine scheduling continuous-time Markov Decision Process, defining the states, the costs and revenues associated with each state, the set of actions and the transitions between states.

States Servers provide resources such as CPU cores and main memory that are allocated to virtual machines. Let r be the number of different resource types. In this work, the resources required by one virtual machine are reserved exclusively to that virtual machine. Moreover, we do not take into account the overhead of the virtual machine monitor and assume that a sufficient amount of resources has been reserved to run it. We consider k groups of servers, where servers in the same group are identical. The i-th group contains n_i servers and is characterized by a capacity vector $\vec{c}_i \in \mathbb{R}^r$, where the h-th element c_{ih} represents the amount of resource h present on each server of group i. The cloud provider supports m classes of virtual machines. A virtual

VALUETOOLS 2013, December 10-12, Torino, Italy

DOI 10.4108/icst.valuetools.2013.254403

machine of class j requires a certain amount of each resource, that we represent with a requirement vector $\vec{l}_j \in \mathbb{R}^r$, where the h-th element l_{jh} represents the amount of resource h required by a virtual machine of class j. The arrival process of virtual machines of class j is a Poisson process with rate λ_j. The expected time a virtual machine of class j remains active (i.e., the service time) has an exponential distribution with parameter μ_j. We represent the state of a server as a vector $\vec{w} \in \mathbb{N}_0^m$, where the j-th element w_j represents the number of VMs of class j running on the server. A feasible state for a server of the i-th group is a vector $\vec{w}$ such that

$$\sum_{j=1}^{m} w_j l_{jh} \leq c_{ih} \quad \forall h \in \{1, \ldots, r\} \tag{1}$$

We denote the set of feasible states for servers of the i-th group as $\mathcal{W}_i$. For convenience of notation we define $\vec{e}_j$ as the m-vector with the j-th element equal to one and all other elements equal to zero. We observe that $\vec{w} + \vec{e}_j$ does not necessarily belong to $\mathcal{W}_i$, since server capacity might be exceeded. Moreover, $\vec{w} = \vec{0}$ is always a feasible server state and denotes a server that can be powered-off.

Example 1 Consider servers providing 6 CPU cores and 16 GB of RAM. Two virtual machine types are made available. The first type of virtual machines requires 2 CPU cores and 8 GB of RAM, while the second type requires 4 CPU cores and 8 GB of RAM. The set of feasible allocations is $\mathcal{W} = \{[0,0],[1,0],[2,0],[0,1],[1,1]\}$.

In order to reduce the number of states required to describe the system, we take advantage of the fact that servers within a group are identical, i.e., indistinguishable. Therefore, we do not track the state of each server in the i-th group, which would require a set of group states equal to the cartesian product of all the server states $\mathcal{S}_i = \mathcal{W}_i^{n_i}$. Instead, a group state is characterized by the number of servers in each state. Formally, we define a group state s as a function from the set of server states $\mathcal{W}_i$ to the set of non-negative integer numbers satisfying the constraint $\sum_{\vec{w} \in \mathcal{W}_i} s(\vec{w}) = n_i$. The set of group states for group i is therefore

$$\mathcal{S}_i = \left\{ s \in \mathcal{W}_i \to \mathbb{N}_0 : \sum_{\vec{w} \in \mathcal{W}_i} s(\vec{w}) = n_i \right\} \tag{2}$$

which is the set of combinations with repetition of n_i elements from a set of $\mathcal{W}_i$ items. The cardinality of $\mathcal{S}_i$ is

$$|\mathcal{S}_i| = \text{multichoose}(|\mathcal{W}_i|, n_i) = \binom{|\mathcal{W}_i| + n_i - 1}{n_i} \tag{3}$$

Example 2 Consider servers and virtual machines as described Example 1. For $n = 3$ servers, the naive formulation requires $|\mathcal{W}|^n = 5^3 = 125$ states. The more compact formulation only requires $\binom{|\mathcal{W}|+n-1}{n} = \binom{7}{3} = 35$ states.

The overall state space of the system is given by the cartesian product of the state spaces of each server group. Hence, $\mathcal{Z} = \mathcal{S}_1 \times \mathcal{S}_2 \times \ldots \mathcal{S}_k$. A state $z \in \mathcal{Z}$ is a k-tuple where z_i is the state of group i when the system is in state z. Hence, $z_i(\vec{w})$, $\vec{w} \in \mathcal{W}_i$, is the number of servers of group i in state $\vec{w}$ when the system is in state z.

Example 3 Consider the first group of $n_1 = 3$ servers described in Example 2, whose state space $\mathcal{S}_1$ has cardinality 35. Moreover, consider a second group of $n_2 = 2$ servers with 4 CPU cores and 16 GB of RAM. For the same virtual machine classes in Example 1, the set of feasible allocations is $\mathcal{W}_2 = \{[0,0],[1,0],[2,0],[0,1]\}$. For this second group of servers we need $\binom{|\mathcal{W}_2|+n_2-1}{n_2} = \binom{5}{2} = 10$ states. The state space $\mathcal{Z}$ of the system has cardinality $|\mathcal{S}_1 \times \mathcal{S}_2| = 350$.

Cost and Revenue Our aim is to maximize the profit, defined as the difference between the revenue generated by running the VMs and the cost of operating the servers. The revenue rate generated by running a VM of class j is $h_j \in \mathbb{R}^+$ and the cost rate of operating a server of type i is $g_i \in \mathbb{R}^+$. The profit rate of a system in state z is therefore:

$$\rho(z) = \sum_{i=1}^{k} \sum_{\vec{w} \in \mathcal{W}_i \setminus \vec{0}} z_i(\vec{w}) \left(h_i - \sum_{j=1}^{m} g_j w_j \right) \tag{4}$$

Powered-off servers, i.e. servers in state $\vec{0}$, are excluded from the summation since they have a null cost rate.

Actions An action a specifies, for each VM class, whether a new VM should be dropped or not and, in the latter case, where it should be provisioned. Hence, an action a is an m-tuple $\{a_1, \ldots, a_m\}$, where a_j belongs to the set:

$$A = \{\text{drop}\} \bigcup \{(i, \vec{w}) : 1 \leq i \leq k, \vec{w} \in \mathcal{W}_i\}$$

When, for a class j, $a_j = \text{drop}$, virtual machines of class j are not admitted to the cloud. The set of actions is $\mathcal{A} = A^m$ and its cardinality is $|\mathcal{A}| = |A|^m = (1 + \sum_{i=1}^{k} |\mathcal{W}_i|)^m$. For a given state, not all actions are feasible. We denote with $\mathcal{A}_z$ the set of feasible actions in state z. An action $\{a_1, \ldots, a_m\}$ is feasible in state z if, for any VM class j such that $a_j = (i_j, w_j)$, a_j is feasible in state z. Action $a_j = (i, \vec{w})$ is feasible in state z if and only if there is at least one server of group i in state $\vec{w}$, i.e. $z_i(i, \vec{w}) > 0$, and such a server has enough capacity to host another VM of class j, i.e. $\vec{w} + \vec{e}_j \in \mathcal{W}_i$. For the special case $a_j = \text{drop}$, we consider two different problem formulations. In the first formulation, $a_j = \text{drop}$ is feasible in state z if and only if there is no room to allocate a VM of class j, i.e. $\forall (i, \vec{w})$ such that $\vec{w} + \vec{e}_j \in \mathcal{W}_i$, $z_i(\vec{w}) = 0$. In the second formulation, $a_j = \text{drop}$ is always feasible and the problem becomes a joint admission-control and scheduling problem, which has potential for higher revenue, since VMs that generate a small revenues can be rejected to reserve space for higher-revenue VMs.

Transitions To complete the definition of the continuous-time MDP we need to define transitions between states. Observe that transitions only occur at the arrival and departure times of virtual machines. Arrival rates of the virtual machines are independent from the system state. In fact, recall that each class of virtual machines has a Poisson arrival process with rate λ_j. Taking action a on a arrival of class j leads to the system state $z' = \chi_A(z, a, j)$. We have to consider two cases: $a_j = \text{drop}$ and $a_j = (i^*, \vec{w}^*)$, which means that VMs of class j are allocated on a server of group i^* in state $\vec{w}^*$. In the former case we have for $z' = z$, while in the second case we have a system state such that the group states $z'_i = z_i$ for all $i \neq i^*$ and

$$z'_{i^*}(\vec{w}) = \begin{cases} z_{i^*}(\vec{w}) - 1 & \vec{w} = \vec{w}^* \\ z_{i^*}(\vec{w}) + 1 & \vec{w} = \vec{w}^* + \vec{e}_j \\ z_{i^*}(\vec{w}) & \text{otherwise} \end{cases} \tag{5}$$

The departure rate $\delta(z, i, \vec{w}, j)$ of virtual machines of class j from servers of group i in state $\vec{w}$ is

$$\delta(z, i, \vec{w}, j) = z_i(\vec{w}) w_j \mu_j \tag{6}$$

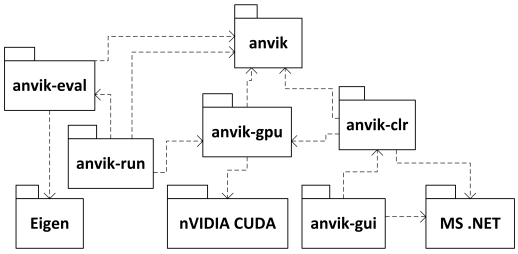

Figure 1: Dependencies between the modules of the tool and third-party libraries.

since there are w_j virtual machines on each server and the departure time is the minimum of $z_i(\vec{w})w_j$ exponential distributions with rate μ_j. The state reached when a virtual machine of class j departs from a server of group i^* in state $\vec{w}^*$ is $z' = \chi_D(z, i, \vec{w}, j)$, where $z'_i = z_i$ for all $i \neq i^*$ and

$$z'_{i^*}(\vec{w}) = \begin{cases} z_{i^*}(\vec{w}) - 1 & \vec{w} = \vec{w}^* \\ z_{i^*}(\vec{w}) + 1 & \vec{w} = \vec{w}^* - \vec{e}_j \\ z_{i^*}(\vec{w}) & \text{otherwise} \end{cases} \quad (7)$$

4. TOOL

We developed a tool to solve the virtual machine scheduling MDP described in Section 3 under the average cost criterion [8]. The algorithm of choice is the value iteration algorithm [8] which, given the problem structure, has time complexity $O(|\mathcal{Z}|\cdot\left(1+\sum_{i=1}^{k}|\mathcal{W}_i|\right)^m)$, where $\left(1+\sum_{i=1}^{k}|\mathcal{W}_i|\right)^m$ is an upper bound on the size of the feasible action set $\mathcal{A}_z$. We have two different implementations. The first one is written in C++ and runs on the CPU, using OpenMP to take advantage of multiple cores. The second one is written in CUDA and runs on the GPU.

Particular care has been taken to avoid storing the state description for all states, reducing the amount of memory required by the optimizer. In fact, memory is the most limited resource when tackling large problems. Even for medium sized problem, reducing the amount of memory greatly improves cache utilization on the CPU. Hence, our optimized versions of the value iteration algorithm do not require the explicit enumeration of system states. Instead, we have developed efficient algorithms to generate state information on the fly given the state index as well as methods to quickly compute the index of the state reached after a departure or arrival. For this reason, only two floating point numbers and one integer number are stored for each system state, which is the minimum allowed by the value iteration algorithm.

The GPU implementation of the optimization algorithm auto-tunes to the available amount of video memory. Hence, if the description of all the system states can be fitted in the video memory, states are explicitly enumerated. This reduces the iteration time by about 50%. On the other hand, if the amount of video memory is insufficient, the memory-optimized algorithm is run. For a problem similar to the Scenario 4 described in Section 5, we were able to fit about 10 million states on a GeForce GTX 760 GPU with 4 GB of memory. The amount of memory required to enumerate all states, however, depends not only on the number of system states but also on the problem structure.

The tool, code-named *anvik*, consists of several components:

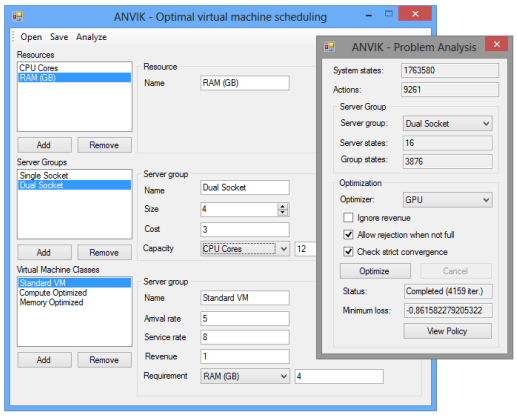

Figure 2: Screenshot of the graphical user interface.

- the *anvik* library, which contains the C++ classes with the problem description, the model analysis, the optimizer interface and the CPU optimizer;
- the *anvik-gpu* library, which contains the C++ class and the CUDA code of the GPU optimizer;
- the *anvik-evaluator* library, written in C++, evaluates the Markov chain induced by a policy, using a freely-available sparse linear solver [3];
- the *anvik-clr* library, written in C++/CLI, which provides a bridge from CLR languages such as C# to the native code in the *anvik* and *anvik-gpu* libraries;
- the *anvik-gui* application, written in C#, which provides the user interface for MS Windows;
- the *anvik-run* cross-platform command-line application, written in C++, computes the optimal policy for a given problem instance or evaluates a policy.

In Figure 1, we represent the dependencies between the components and the dependencies to third-party software, namely the .NET Framework for the graphical user interface, the Eigen mathematical library for the policy evaluator and the CUDA framework for the GPU optimizer. Both the graphical user interface *anvik-gui* and the command line interface *anvik-run* load problem files in a simple text format. A screenshot of the GUI is represented in Figure 2. The main window allows the specification of a problem by defining the set of resources, the server groups and the virtual machine classes. Clicking the analysis button leads to the analysis window, which shows the number of states in the Markov decision process, the number of actions as well as the number states required for each server group and for each individual server. When the user clicks the optimize button, the computation of the optimal policy starts asynchronously. A drop-down list allows the choice of the optimizer.

5. RESULTS

Scenario 1 The simplest scenario we can consider is a single server and a single class of VMs, with arrival rate $\lambda = 0.5$ VM/s and service rate $\mu = 1$ VM/s. The server can host at most one VM and has unary cost. The server state space is therefore $\mathcal{W} = \{0, 1\}$ and the system state space is $\mathcal{Z} = \mathcal{S} = \{[1, 0], [0, 1]\}$. There are two actions: $\mathcal{A} = \{\{\text{drop}\}, \{(1, 1)\}\}$. The first action drops arriving VMs, while the second one allocates them on a server of type 1 in state 1. The only feasible action in system state

$[0, 1]$ is to drop the arriving VMs. In system state $[1, 0]$, since we do not allow admission control, the only feasible action is $\{(1, 1)\}$. The algorithm converges in few iterations to the average loss rate 0.3333. The same result can be obtained by observing that, in this trivial scenario, under the optimal policy, the system behaves as an M/M/1/1 queueing system. By enabling admission control, since VMs do not generate revenue, the optimal policy is simply to always drop VMs, achieving an average loss rate 0.

Scenario 2 We consider a first group of two servers as in Scenario 1 and a second group with a single server of larger capacity and higher cost. This server is twice as expensive as the servers of the first group and it can host a maximum of three virtual machines. The states for the server of the second group are therefore $\mathcal{W}_2 = \{0, 1, 2, 3\}$. We have a system with states a total of $|\mathcal{Z}| = |\mathcal{S}_1| \times |\mathcal{S}_2| = 24$ states:

$$\mathcal{S}_1 = \{[2, 0, 0], [1, 1, 0], [1, 0, 1], [0, 2, 0], [0, 1, 1], [0, 0, 2]\}$$
$$\mathcal{S}_2 = \{[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]\}$$

The set of actions considers both groups of servers and the two corresponding sets of server states:

$$\mathcal{A} = \{\text{drop}, (1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3), (2, 4)\}$$

The solution of this problem yields a minimum cost of 0.4192. An interesting observation about the optimal policy is that in the state with the two small servers turned off and a single virtual machine running on the big server, the next virtual machine is allocated on one of the empty servers. This is counter-intuitive and the opposite of what a greedy policy would do: use the spare room on the big server for the new VM. The greedy choice costs less in the short term, but in the long term it is more expensive than the optimal policy that, since we are in a low-load scenario, aims to turn off the big server as soon as possible, in order to reduce its cost.

Scenario 4 We now describe a more realistic scenario with two server groups, with n_1 and n_2 servers each. The first server group is characterized by 8 CPU cores, 8 GB of RAM and a cost rate $h_1 = 0.5$. The second group contains servers with 12 CPU cores, 16 GB of RAM and a cost rate $h_2 = 1.5$. We consider two workload classes with arrival rates λ_1 and λ_2 and service rates $\mu_1 = 2.8$ VM/s and $\mu_2 = 1.2$ VM/s. The first class of virtual machines requires 4 CPU cores and 4 GB of RAM, while the second class requires 6 CPU cores and 8 GB of RAM. The profit rates associated with the two classes are $g_1 = 0.5$ and $g_2 = 1$. In Table 1, we evaluate the profit achieved by the optimal policy for different choices of n_1, n_2 such that $n_1 + n_2 = 20$. We consider two problem formulations, without admission control (MDP 1) and with admission control (MDP 2) and set $\lambda_1 = \lambda_2 = 3$ VM/s. We observe that the revenue increases by 26% when increasing n_1 from 5 to 10, while it is almost unchanged when increasing n_1 from 10 to 15. Moreover, we observe that admission control yields a 11% increase in profit when $n_1 = 5$, while it provides no advantage when $n_1 = 15$. In Table 2, we evaluate the impact of the mix of VM arrivals. We set $\lambda = \lambda_1 + \lambda_2 = 6$ VM/s, $n_1 = 5$ and $n_2 = 15$. Increasing the fraction of arrivals of the smallest VMs the profit decreases quickly and the benefit of admission control grows thinner: from 14% to 6% when the arrivals of the first class increase from 30% to 70% of the total arrivals. To give an indication of the execution time, the last problem instance in Table 2, characterized by over 3×10^6 states, was solved in 4m18s on a GTX 760 GPU and in 12m51s by a i7 3930k CPU. While the problem is computationally challenging, it is important to stress that the optimal policy can be computed offline. For the online scheduler, applying the optimal policy takes negligible time, since it only requires a table lookup.

Table 1: Results for $\lambda_1 = 3$ VM/s, $\lambda_2 = 3$ VM/s.

n_1	n_2	MDP 1	MDP 2
5	15	1.09549	1.22216
10	10	1.37915	1.38038
15	5	1.38158	1.38158

Table 2: Results for $n_1 = 5$, $n_2 = 15$.

λ_1	λ_2	MDP 1	MDP 2
1.8 VM/s	4.2 VM/s	1.30538	1.48607
3.0 VM/s	3.0 VM/s	1.09549	1.22216
4.2 VM/s	1.8 VM/s	0.85615	0.91131

6. CONCLUSIONS

We described a Markov Decision Process tool for the optimal scheduling of multiple classes of virtual machines on an heterogeneous cloud. We have shown that the optimal policy often takes counter-intuitive actions that would not be taken by a simple greedy approach. The tool, released under an open-source license, is memory-optimized and tuned to run on both multi-core CPUs and on GPUs.

7. REFERENCES

[1] R. N. Calheiros, R. Ranjan, and R. Buyya. Virtual machine provisioning based on analytical performance and qos in cloud computing environments. In *Int. Conf. on Parallel Processing*, pages 295–304. IEEE, 2011.

[2] X. Gao, y. Lu, M. Sharma, M. S. Squillante, and J. W. Bosman. Stochastic optimal control for a general class of dynamic resource allocation problems. *SIGMETRICS Perform. Eval. Rev.*, 41(2):3–14, Aug. 2013.

[3] G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[4] J.-C. Huang, H.-H. S. Lee, and M. M. Hossain. Migration energy-aware workload consolidation in enterprise clouds. In *Conference on Cloud Computing Technology and Science*, pages 405–410. IEEE, 2012.

[5] T.-J. Lee, K.-C. Huang, B.-J. Shen, H.-Y. Chang, Y.-H. Tung, and P.-Z. Shih. Resource allocation and dynamic provisioning for service-oriented applications in cloud environment. In *Int. Conf. Cloud Computing Technology and Science*, pages 839–844. IEEE, 2012.

[6] K. Mills, J. Filliben, and C. Dabrowski. Comparing vm-placement algorithms for on-demand clouds. In *Int. Conf. on Cloud Computing Technology and Science*, pages 91–98. IEEE, 2011.

[7] D. Niyato. Optimization-based virtual machine manager for private cloud computing. In *Int. Conf. on Cloud Computing Technology and Science*, pages 99–106. IEEE, 2011.

[8] L. I. Sennott. *Stochastic dynamic programming and the control of queueing systems*. Wiley, New York, 1999.