**DEVELOPING ALLOCATION RESOURCES IN DATA CENTER BY USING VIRTUAL MACHINES**

Samandarov B.G’., Palvonova S. B.

***Annotation***. Continuous adjustment of the horizontal scaling of the application, located in the data centers, looks like a good candidate for automatic control distribution that distributes resources in an enclosed room given their current workload. Despite several attempts, the real applications of these methods in cloud computing infrastructures face some difficulties. Some of them basically return to the basic concepts of automatic control: controllability, inertia of the controlled system, amplification and stability. In this paper, taking into account our recent work on the creation of a management structure, designed for automatic distribution in virtualized applications, we are trying to identify sources of instability in managed systems. As examples, we analyze two types of policies: threshold and reinforcement learning methods for dynamically scaling resources. experiments show that both ways are complex, and that trying to implement a controller without looking at the way of a controlled system of reactions to actions, both in time and in amplitude, is doomed to failure.

***Keywords***-Cloud computing, Application hosting, Resource allocation, Closed loop systems, Controllability, Hysteresis.

We now attempt to identify from experiments and the knowledge of the cloud and of the applications, the sources of instabilities discussed in the section, striving for simple yet efficient rules to obey in order for cloud managers to get good resource management policies. The application consists of three main components: Apache load balancing server, JBoss application server and MySQL database server. In the initial setup, each the component is run on a separate virtual machine (VM). The same profile is used for all virtual machines during the experiments: one computational unit (equivalent to 2.66 GHz), 256 MB of memory and 10 GB of memory. Figure 1 presents an overview of the RUBI architecture and determined for our experiments. VirtRL framework makes the application level RUBiS scalable to evaluate two scaling strategies (static threshold and Q-Learning), which were discussed below. VirtRL Control API allows you to collect data at constant time intervals (20 seconds here). Each information is a state system during the time interval and consists of: average query speed (workload), average response time RUBiS (performance) and the number of virtual machines at the scaling point (resource usage).



Figure 1. RUB is architecture and its scaling point

Our implementation The SLA (as an average response time) meet performance goals (SLOs). This policy has five parameters: upper threshold ut, lower threshold lt, fixed the number of virtual machines n that must be allocated or released, and two durations of inertia: inUp for scaling and in Down for decreasing. With a careful selection of these parameters conceal the delay phenomenon (see section II-C), in allocating / freeing virtual machines, initializing JBoss servers, and rebalancing client requests among existing servers (after adding or removing a resource). At each iteration, if performance violates ut, n virtual machines are requested and registered at the zoom point. The controller then blocks itself for inUp seconds. If the performance is below lt, n VMs are removed and their resources are released. Again, the controller locks itself for a few seconds. In our experiment, we set the following values: ut = 0,8, lt = 0,4, n = 1, inUp = 120 and inDown = 60. SLO: " the average response time of the end user should not exceed ut. " Figure 2 illustrates the results of our first experiment. Several phenomena can be explained. The first is about complex configuration of policy settings. First, the service The provider must scan the target system for its performance model. Then, given the high-level objective to achieve a certain workload, he can choose correct parameters to meet this SLO. In the download figure, The second and third sinusoidal oscillations show The case where the parameters are well suited.



Figure 2. Result of a 7.5 hours experiment with the static threshold-based policy. The top graph shows the SLA objective and the average response time evolution. The bottom graph shows the workload and the VM usage. The figure shows that the policy configuration is well-suited for the second and the third oscillations.

However, when the control power and the slope differ too much, there are two sources of instability. First fluctuations show what happens when the control is stronger than the slope. Unnecessary servers are distributed, and then released at the next iteration. Be too reactive The system deceives its performance and causes instability. On the other hand, when the control is weaker than the slope, some of these characteristics are also inherited. On the fourth and fifth fluctuations, decisions are made too late which causes another form of instability and actually serves unavailability. Because the power of control (parameter n) is selected once and for all, the current threshold politicians are particularly susceptible to these types of instability. During these periods based on the response time of submitted requests and rejected they are not taken into account.

Finally, Figure 3 shows what happens when latency not properly hidden from the decision module. We outplayed second swing with a policy using some closer thresholds. This setup demonstrates the impossibility of capturing phenomenon of latency. When violations occur, a new virtual machine is requested and added to the zoom point. Then, as soon as productivity increases, the policy finds the last created VM is useless because of the near lower threshold. Thus, it tends to release it even if this leads to SLA violations in the future.



Figure 3. Result of the second oscillation replayed with closer thresholds: ut = 0.8 (SLO) and lt = 0.6. The policy constantly allocates and deallocates VMs. Because of the latency in these actions, SLA violations occur which cause service unavailability.

At time t, taking into account the state of st (the current workload and the use of resources), this looks for the best action to perform. Possible actions: (i) add n new virtual machines, (ii) release n virtual machines, or (iii) do nothing. Because they have consequences for the target application time, a scalar measurement is calculated, called reward at the moment (t + 1), to appreciate the kindness and badness specific in. In our implementation, a high award returns when SLO is performed (minus a certain penalty if the zoom point is reassigned) and negative otherwise. Therefore, unlike our threshold policy, Q Learning records this history information in a value table. This compares all states of the st system with their best actions at and can be initialized with selected values ​​or based on specific, but simple performance model. Then study phase is required to capture a real model before convergence to an optimal policy. The formula used in our Q-Learning implementation is:

Q(st, at) ← Q(st, at)(1 − αt(st, at)) +

αt(st, at) (r(st+1, at) + γ maxa Q(st+1, a))

Here (st, at) are the state and action at time t, r(st+1, at) is the delayed reward for at evaluated at time (t + 1), maxa Q((st+1, a) denotes the Q value of the best action into the next state at time (t + 1), the constant γ ∈ [0, 1] is a “discount parameter” expressing the present value of expected future reward and αt is a “learning rate” parameter defining the impact of the new data on the Q function. Apart from the reward function, four parameters can be customized: learning rate α, discount factor γ and two inertia durations: in Up for scaling up, in Down for scaling down. Both durations hide the latency phenomenon, as in the threshold case. In this experiment, we set the following values: α = 0.05, γ = 0.5, in Up = 60 and in Down = 60. Beware the SLO remains the same as before and is explicitly expressed through the reward function.



Figure 4. Result of the Q-Learning policy. Notice that the two graphs do not cover the training phase. The figure shows that once the performance model has been learned, the two sources of instability (power of the control and oscillations in the inputs) tend to disappear since the policy adapts its power to the oscillation slopes.

This Heuristics tend to accelerate performance of the performance model, but limits the space development of the state. Figure 4 illustrates the results of this second experiment after the training phase. Then there can be several phenomena explained. At first, this shows that with good marked performance model, violations of SLA are minimized. The state, the policy of Q-Learning converges to the values Virtual machines that need to be allocated or released. With different sinusoidal It helps to adapt the control power to the workloads. Then two sources of instability (power and fluctuations in inputs), as a rule, are solved.

REFERENCES

1. “Rightscale,” <http://www.rightscale.com/>.
2. “Google App Engine,” <http://code.google.com/intl/fr/appengine/>.
3. F. Golnaraghi and B. Kuo, Automatic Control Systems, 9th ed.

 Wiley, 2010.

4. M. L. Littman, “Algorithms for Sequential Decision Making,”

 Ph.D. dissertation, Brown University, 1996.