When Cloud Virtual Machine Images Need to Be Updated

Marco A. S. Netto, Marcos D. Assunção
Lakshminarayanan Renganarayana, Chris Young

IBM Research

{mstelmar,marcosda}@br.ibm.com, {lrengan,ccyoung}@us.ibm.com

***Abstract*—Updates to cloud images typically come in the form of patches and either correct bugs and security vulnerabilities or introduce new functionality. The complexity and effort required to patch an image is much higher than what is required to patch an instance. This is due to the risk of incorrectly modifying configurations, breaking the cloud provisioning for the image or preventing the correct operation of the management stack. In a managed cloud, if a patch is not applied to an image it must be applied to each instance of the image. This process results in wastage of compute resources and causes the customer to receive an initial instance that has not been tested by the cloud provider. This paper proposes an algorithm to identify when an image should be updated based on the frequency of instantiation requests and the outstanding patches as actually experienced in a production data centre.**

I. Introduction

Infrastructure as a Service (IaaS) providers typically offer a set of generic starter images which provide an effective base for further customisation to a consumers' workload. Cloud consumers typically have the option to leverage community constructed images or to build their own custom images. The consumers' goal is to find an image which most closely matches their workload requirements, therefore reducing manual effort associated with further image customisation.

The complexity of image management and governance increases as each customer introduces their own unique requirements [1]. The currency of software within an image provides a challenging problem for both cloud provider and consumer as neither party benefits from using outdated or insecure software versions. Hence, a method is needed to assist parties responsible for image currency to determine the most appropriate frequency and method to apply software patches. This paper helps address this challenge for the benefit of cloud provider and consumer; an open issue not well explored in the cloud image management literature [1]–[11].

This work introduces an algorithm that assists the parties responsible for image currency to determine which images should be updated with software patches at a point in time. The algorithm considers several aspects of cost involved with updating an image or dynamically updating each instance at instantiation time. The algorithm is evaluated against historic request data from a production data centre.

II. Image Updating Algorithm

The process and costs associated with updating an instance versus an image are quite different—as errors updating images may impact all provisioned virtual machines. These two aspects are included in the proposed algorithm to determine when images should be updated.

Algorithm 1: Pseudo-code for determining the maximum time for updating an image.

```
   Input: reqList, patchList, image
   Output: maximumUpdateTime
1  iPatchList ← patchList.getPatches(image)
2  patchICost ← iPatchList.getImageCost(image)
3  patchRCost ← iPatchList.getReqCost(image)
4  nReqs ← 0
5  maximumUpdateTime ← 0
6  foreach time unit t do
7      maximumUpdateTime ← maximumUpdateTime + t
8      nReqs ← nReqs + reqList.getFutureReqs(image, t)
9      if patchICost < nFutureReqs * patchRCost * k then
10         return maximumUpdateTime
11 return -1
```

Algorithm 1 represents the pseudo-code for determining the maximum update time when a set of updates need to be applied onto an image, which runs in the system depicted in Figure 1. The algorithm can be executed at fixed time intervals or dynamically upon the management system receiving a new patch. This alleviates the necessity to attempt to predict the arrival times of future patches. The algorithm has the following input variables:

- **reqList:** previous requests for provisioning instances. Each request contains the required operating system, software stack, and additional configuration parameters;
- **patchList:** description and links to download and apply operating system and software updates;
- **image:** image to be evaluated.

The algorithm starts by initialising $iPatchList$ (Line 1), which includes the patches from $patchList$ that are deemed relevant for the image. After that, the two update costs are set (Lines 2–3): $patchICost$ and $patchRCost$ which represent the cost of applying the patches $iPatchList$ into the image and into an instance provisioned from that image. Then a loop is used to compare the image and instance update costs according to the advancement of time (Lines 6–10)—such time can be

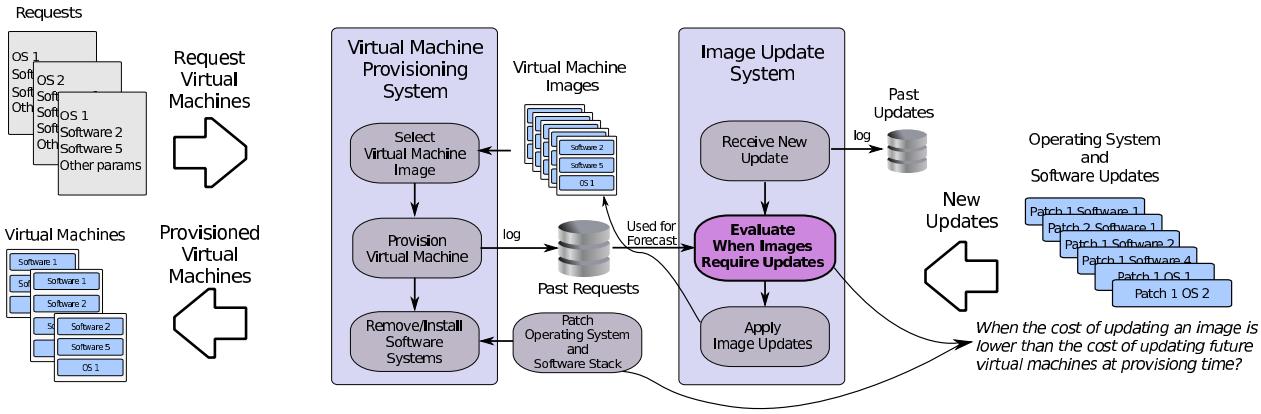

Fig. 1. The challenge from the cloud provider's perspective is to determine when the cost of patching an image is less than the cost of dynamically patching future cloud instances at provisioning time.

hours, days, weeks and so on. The number of future requests $nReqs$ is updated for each loop iteration (Line 8). Then a test is executed to determine if the the cost for updating an image is lower than the cost of updating all future instances from that image (Line 9). If the cost is lower then return the maximum update time (Line 10). A k factor is used to compensate for the fact that the cost may vary for each instance provisioned (Line 9).

It is important to note that the algorithm considers the evaluation of a single image. However, a likely scenario would be a system administrator who wants to know the maximum update time for a set of images. For this scenario, two extra variables for storing the total cost of updating all images and the total cost for updating all instances could be compared, and when the first is lower than the second, the maximum update time is returned.

III. Evaluation

The rationale behind the image updating algorithm is that by predicting requests and using information on the costs associated with updating an image or an instance, it is possible to determine when an image should to be updated. This knowledge reduces the time associated with applying patches to the image and the impact on future instances based on the image. The experimental results presented in this section demonstrate that the principle is sound. We describe the experiment setup, the metrics used to evaluate the image updating process, and present the result analysis.

A. Experiment Setup and Metrics

The first key input parameter for the evaluation is the arrival time of requests. In order to setup this value, we analysed several requests for multiple Linux images of a production data centre. The requests for each image have considerable fluctuations, presenting some seasonality over the months, and show higher demand in the second half of the year. Based on this data, we setup the number of requests per day following a Zipf distribution.

The second key input parameters are the arrival time and size (in MBs) of the software updates. These values vary according to the type of update (OS and software in general). Therefore, based on the collected data of patches for linux distributions and software stack such as DB2, IBM HTTP Server, and Websphere, we modelled the patch arrival time as a Zipf distribution and patch size as a normal distribution.

We also considered the risk of a patch breaking an image (*i.e.* the resource provider needs to fix an instance created from the image when it does not work as the customer expected). As the image update algorithm depends on how accurate the prediction of future requests is, we considered two scenarios where the prediction (i) overestimates or (ii) underestimates the number of future requests. The values of these and other parameters are summarised in Table I.

Metrics. We considered the following metrics when evaluating the proposed algorithm with the input parameters described in this section:

- **Wasted Time:** Percentage of the wasted time compared to the best approach, that is update the image or update the future instances, as defined in Equation 1;
- **Number of Days:** Number of days in the future where the image needs to be updated;
- **Recovery Waste Cost:** percentage of the amount of time the instance was unable to be used due to a bad image patch in relation to the maximum time when the patches for the image need to be applied.

$$\text{Wasted Time} = \frac{\left| \sum_{i=1}^{n} ImgP_i - \sum_{j=1}^{m} VMP_j \right|}{min\left(\sum_{i=1}^{n} ImgP_i, \sum_{j=1}^{m} VMP_j \right)} \quad (1)$$

Equation 1 is defined as the time cost difference between updating the image and updating future instances divided by their minimum value, where: n is the number of patches of the image, m is the number of future requested instances that will use the image, $ImgP$ is the cost for applying a patch on an image, and VMP cost of applying a patch on an instance.

TABLE I. SUMMARY OF EXPERIMENT PARAMETERS AND VALUES.

Parameter	Value	Description
ReqArrival	Zipf (1.5)	Request arrival time
VMPatchTime	Normal (50,30)	VM Patch time (secs)
ImgPatchTime	Normal (100,30)	Image Patch time (secs)
BreakRisk	0.05–0.5%	Probability of image break
PredAccOver	0–300%	Pred. accuracy overest.
PredAccUnder	0–99%	Pred. accuracy underest.

B. Result Analysis

The first analysis considers the accuracy of the prediction for future requests. As discussed earlier, this is a key parameter as accuracy of the predictions is highly dependent on the data centre and its workload. We analysed both overestimation and underestimation of the number of future requests.

Overestimation of future requests. This is a scenario of cautious management, where the value of the future number of requests is overestimated. The higher the overestimation, the earlier the day an image needs to be updated. This behaviour is illustrated in Figure 2, which shows the average number of days in the future (including the standard deviation bars) when an image is updated as a function of the overestimated number of requests that will use the image. The number of days in the future when to update the image decreases with the overestimation until it reaches a point where the algorithm determines that the image needs to be updated on the *"next day"*. Hence, with overestimation of requests, even with few requests the image is updated more often.

The early image update can waste a considerable amount of time and resources, as depicted in Figure 3. This figure shows the wasted time of updating an image as a function of the underestimated number of requests that will use the image. We notice that when the overestimation is high, similar to the number of days metric, the wasted time reaches a threshold. This can be observed by comparing the standard deviation bars under low and high overestimation. This happens because the number of days in the future when the image needs to be updated stops to reduce, getting closer to the *"next day"* phenomenon. The results also show a tolerance of approximately 10% for the overestimation to determine the maximum time to update the image.

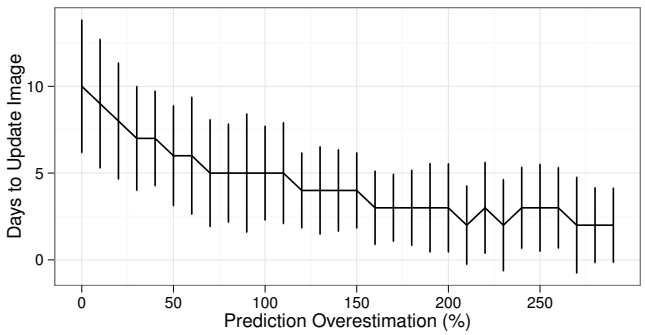

Fig. 2. Number of days in the future when an image is updated as a function of the overestimated number of requests that will use the image.

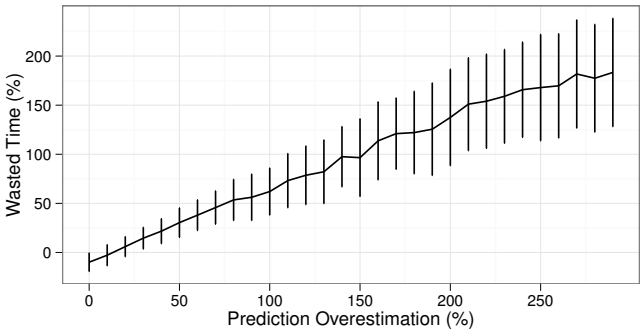

Fig. 3. Wasted time of updating an image as a function of the overestimated number of requests that will use the image.

Underestimation of future requests. This is a scenario of more relaxed management, where the number of future number of requests is underestimated. In this case, the higher the underestimation, the later an image is updated. This behaviour is illustrated in Figure 4, where it shows the number of days in the future when an image is updated as a function of the underestimated number of requests that will use the image. Note the difference from the overestimation scenario; here we varied underestimation up to 99%, as the value of 100% represents that there will be no future requests. Underestimation has a key difference of overestimation: depending on the quality of the request arrival prediction, the algorithm may determine that it will take months until the update is necessary; that is, the impact on the number of days is very high, as we can observe in the curve shown in Figure 4. Depending on the scenario, this may be acceptable, but system administrators may consider including a threshold for the days to update the image.

The possibility of having long periods when the image is not updated causes a considerable amount of time to be wasted as depicted in Figure 5. This figure shows the wasted time of updating instances as a function of the underestimated number of requests that will use such an image. Similar to the number of days metric, the wasted time follows an exponential curve due to incorrect prediction of very small number of requests, which results in many more instances to be updated, instead of only updating the image that will generate such instances. The results also show a tolerance of approximately 5% for the underestimation to determine the maximum time to update the image.

Risk of performing bad patches. Another factor we investigated is the impact of bad patches applied to images on the subsequent provisioning time of instances. Figure 6 presents the recovery wasted time as a function of image update probability failure. We varied the probability from 0.5 to 1.0. We notice that when there is a high risk of an image update to compromise future instances, the time to recover those instances may be considerably high. The results presented here were obtained considering one day to detect the error on an instance due to a bad patch on the image. The recovery wasted time may be higher and more costly if the time to detect a problem is long and the instances need to be recreated.

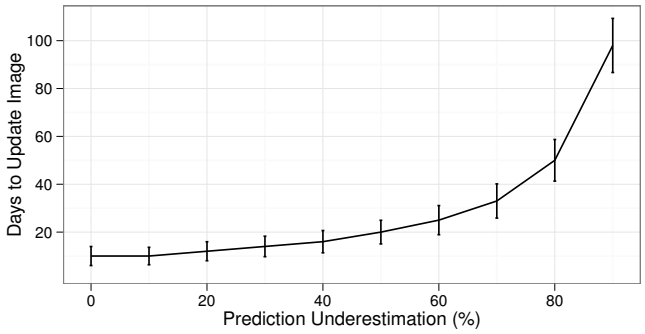

Fig. 4. Number of days in the future when an image is updated as a function of the underestimated number of requests that will use the image.

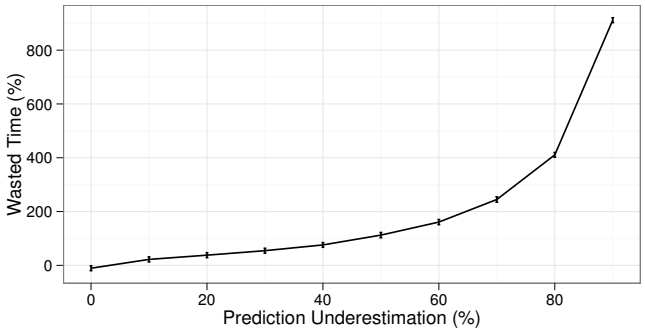

Fig. 5. Wasted time of updating instances as a function of the underestimated number of requests that will use the image.

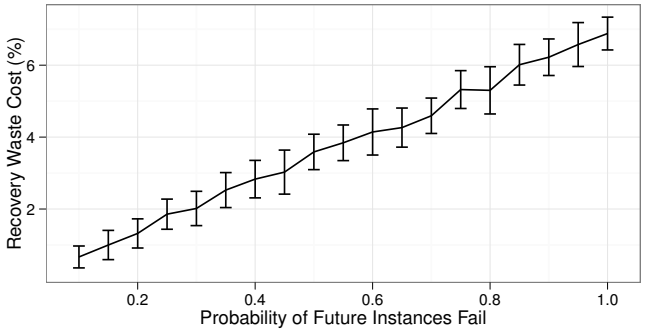

Fig. 6. Recovery wasted time as a function of image update probability failure.

IV. Final Remarks

Cloud images need to be maintained by applying patches regularly in order to, for instance, avoid security vulnerabilities that can compromise a customer's resources and information; or add new software features. The decision of when to patch an image is not trivial as the additional complexity and time required to patch an image must be balanced against that of updating an instance.

This paper introduced an algorithm to assist the parties responsible for image currency to determine which images should be updated with software patches. The algorithm takes into consideration several aspects of cost associated with performing the update to either the image or to an instance during its provisioning process.

Based on the evaluation results using data from a production data centre and associated software configurations, we identified the importance of knowing the actual costs involved to update images and instances. As image updates are more costly than instance updates, predicting future requests is fundamental to determine how long the images can remain out-of-date in order save work updating them. However, the wasted cost by performing a bad decision on either updating an image or updating an instance is highly dependent on the prediction of future requests. The results presented in this paper are representative for the workloads we selected to evaluate the image update algorithm. However, we discussed the steps necessary to perform the analysis that can be leveraged by image managers with different settings to perform more effective decisions on when images need to be updated. This is particularly important for managed clouds where customers expect high levels of quality-of-service and quality of images to provision their instances.

References

[1] D. Reimer, A. Thomas, G. Ammons, T. Mummert, B. Alpern, and V. Bala, "Opening black boxes: Using semantic information to combat virtual machine image sprawl," in *Proceedings of ACM SIGPLAN/SIGOPS VEE*, 2008, pp. 111–120.

[2] T. Garfinkel and M. Rosenblum, "When virtual is harder than real: Security challenges in virtual machine based computing environments," in *10th USENIX HotOS*, Berkeley, USA, 2005, pp. 20–20.

[3] W. Zhou, P. Ning, X. Zhang, G. Ammons, R. Wang, and V. Bala, "Always up-to-date: Scalable offline patching of vm images in a compute cloud," in *ACSAC 2010*, New York, NY, USA, 2010, pp. 377–386.

[4] A. Kochut and A. Karve, "Leveraging local image redundancy for efficient virtual machine provisioning," in *Proceedings of the 13th IEEE/IFIP Network Operations and Management Symposium (NOMS'12)*, 2012.

[5] R. Filepp, L. Shwartz, C. Ward, R. D. Kearney, K. Cheng, C. C. Young, and Y. Ghosheh, "Image selection as a service for cloud computing environments," in *Proceeding of the IEEE Service-Oriented Computing and Applications (SOCA'10)*, Perth, Australia, Dec. 2010, pp. 1–8.

[6] A. Ganguly, J. Yin, H. Shaikh, D. Chess, T. Eilem, R. Figueiredo, J. Hansom, A. Mohindra, and G. Pacifici, "Reducing complexity of software deployment with delta configuration," in *Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management (IM'07)*, 2007, pp. 729 –732.

[7] M. D. Assuncao, M. A. S. Netto, B. Peterson, L. Renganarayana, J. Rofrano, C. Ward, and C. Young, "Cloudaffinity: A framework for matching servers to cloudmates," in *Proceedings of the 13th IEEE/IFIP Network Operations and Management Symposium (NOMS'12)*, 2012.

[8] W. A. Jansen, "Cloud hooks: Security and privacy issues in cloud computing," in *44th Hawaii International Conference on System Sciences (HICSS 2011)*, Jan. 2011, pp. 1–10.

[9] Y. Chen, V. Paxson, and R. H. Katz, "What's new about cloud computing security?" Berkeley, USA, Jan. 2010.

[10] G. Ammons, V. Bala, T. Mummert, D. Reimer, and X. Zhang, "Virtual machine images as structured data: the mirage image library," Proceedings of the USENIX HotCloud 2011, Jun. 2011.

[11] R. Wartel, T. Cass, B. Moreira, E. Roche, M. Guijarro, S. Goasguen, and U. Schwickerath, "Image distribution mechanisms in large scale cloud providers," in *Proceedings of the IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom'10)*, 2010.