Leveraging Local Image Redundancy for Efficient Virtual Machine Provisioning

Andrzej Kochut, Alexei Karve
IBM T.J. Watson Research Center
19 Skyline Drive, Hawthorne, NY, 10532, USA
{akochut,karve}@us.ibm.com

***Abstract*—Virtualized data centers became ubiquitous and led to invention of new delivery model called Cloud computing. Cloud service providers pursue high degree of automation for the management processes in order to make their services responsive and inexpensive. One of such critical processes is virtual machine provisioning. This article proposes and evaluates a provisioning model that leverages virtual machine image similarity to reduce the data volume transferred from the storage server to the hypervisor on which the virtual machine is being instantiated. We also present an analytical model of such a provisioning process for a single hypervisor shedding light on the impact of degree of image similarity, system utilization, and hypervisor capacity on the performance of the system. The model is validated using discrete event simulator. The algorithm has been implemented on a testbed system and also extensive simulations of virtualized server clusters were conducted. The proposed provisioning scheme can offer significant (up to 80%) reduction in data transfer between storage server and hypervisors thus significantly reducing provisioning time while also decreasing cost.**

I. Introduction

Cloud Computing [13], [2], [17] is becoming a popular IT delivery model [10] in which computing resources are made available on-demand through a shared physical infrastructure. The expectation is that sharing hardware, software, network resources, system management, and labor costs will reduce per unit compute cost both for individual users and enterprises. Compute Cloud is elastic - it provides the capability to add and remove capacity as needed with small lead times by leveraging virtualization technology [7], [12], [11], [22], [1], [24], [6].

The key consideration for Cloud service providers is to ensure high level of service while minimizing the cost. One of the critical processes in Cloud management, affecting both user experience and infrastructure costs, is provisioning new virtual machine (VM) instances. Typically this function is implemented using image-based provisioning - a deployment and activation mechanism that clones a "golden" read-only virtual machine image residing in an image library to create a new virtual machine instance. In order to minimize cost, virtual machines' disks usually reside on inexpensive Direct Attached Storage (DAS), while user data is stored on managed Network Attached Storage (NAS). One of the problems with image-based provisioning is copying the image from a storage server hosting the library to a hypervisor (when new VM instance is requested). It creates a long latency for instance creation and significant demand on storage and network throughput (therefore increasing cost). An approach partially addressing this problem is OS streaming where the image is copied progressively starting with blocks that are most likely to be needed first. This approach allows the VM to be started with low latency; however the user may experience unpredictable delays waiting for blocks that were not pre-fetched. This approach also requires transferring the full image therefore does not address the network and storage bandwidth problems.

978-1-4673-0269-2/12/$31.00 © 2012 IEEE

This paper introduces and analyzes a virtual machine provisioning algorithm that significantly improves VM provisioning time for systems using direct attached storage to store VM ephemeral data. The approach leverages the redundancy across images thus reducing load on storage server and consumed network bandwidth. When a fraction of an image to be provisioned is already present on the host (because of other VMs currently present on the host having common content sections with it), the redundancy driven approach locally reconstructs this portion of the image (without the need to transfer data from storage server). Note that the locally running VMs use local image instances that can get modified. In order to protect the blocks that can be used to reconstruct new instances from being modified by running instances, copy-on-write (e.g., [15]) approach is used. It ensures that local VM instances do not modify underlying source image (a local copy is created when blocks are modified). Note that if one or more instances of the image to be reconstituted are already present on the host, then the approach can immediately instantiate a copy-on-write VM instance based on read-only image. In this case, the fraction of image available locally is 100%. Intuitively, degree of similarity among images will tend to improve the fraction of local reconstruction. Also having highly loaded (in terms of number of virtual machines allocated to it) host increases the probability of local reconstruction by increasing the odds that other images required for the reconstruction are present on the host.

In order to understand and explain the above relationships this article proposes an analytical model that allows to quantify expected fraction of the image that can be reconstituted based on the local hypervisor state. It sheds light on how this quantity depends on key system parameters such as average utilization, degree of similarity among images, capacity of the hypervisor, and image provisioning frequencies. The model is validated using extensive simulations based on both synthetically generated similarity matrices as well as using 77-image library obtained from VMware Marketplace [23]. The first allows us to explore parameter space while the latter represents a typical image catalog of a Cloud service provider.

We have implemented the hypervisor agents as well as provisioning manager tracking the state of the system. It allows to reconstitute images at target hypervisors by copying locally available image segments from direct attached storage and fetching the remaining data from the storage server.

In order to evaluate the performance of the system for large hypervisor clusters we have performed detailed simulations. The effect of several VM placement policies is explored and a new policy based on amount of locally available content is proposed and evaluated. The approach is shown to give as much as 80% reduction in terms of image transfer time for highly loaded systems.

Roadmap

The remainder of this article is organized as follows. Section II introduces similarity representation method for image libraries. Section III describes the details of the system model and derives analytical formula for expected fraction of the image that can be reconstituted based on the local hypervisor state. Section IV presents results of the model validation and also discusses insights from the model. Sections V and VI introduce and evaluate local reconstitution algorithm, respectively. Finally, Sections VII and VIII present related work and conclude, respectively.

II. Image Similarity Representation

Assume a library consisting of L images. Each image is a sequence of blocks (e.g., each of size of 512 bytes) related to block device representation of the image. We represent similarity across the library by defining a set of non-overlapping clusters of blocks. The union of all clusters is equal to all unique data blocks contained in the library, i.e., each of the images can be re-created based on a subset of the clusters. Note that if an image has internal redundancy then a block of a cluster may represent multiple identical blocks of the image.

Assume that the library has C of such non-overlapping clusters. Define a similarity matrix S of size $C \times L$ that captures degree and type of content overlap across the images. Each row of the matrix relates to a cluster and column to an image. S is a binary matrix defined as follows:

$$S_{k,l} = \begin{cases} 1 & \text{if cluster } k \text{ is part of image } l \\ 0 & \text{otherwise} \end{cases} \qquad (1)$$

for $k = 1, \ldots, C$ and $l = 1, \ldots, L$. In addition, define $cl\_size(c)$ for $c = 1, \ldots, C$ to be a size of cluster c. To reconstitute an image one has to obtain clusters having "1" in the column corresponding to the image being reconstituted. The content of each of the shared clusters can be obtained from any image that contains this cluster. The system maintains mapping files specifying block locations of data in the image (the details are omitted for brevity). Note that actual image size could be larger than the sum of cluster sizes due to identical blocks within the image. TABLE I shows an example of such a similarity matrix for a library consisting of 10 images and 20 clusters. The similarity matrix has been divided into two sections: the upper part is the set of 10 unique clusters CL-01 to CL-10 (each containing blocks present in only one of the images) while the lower part is that for shared clusters. The rightmost column shows cluster sizes (in MB). For example, image type 1 consist of clusters CL-01, CL-11, CL-14 and CL-20.

Cluster	Image type identifier										Cluster
identifier	1	2	3	4	5	6	7	8	9	10	size (MB)
CL-01	1	0	0	0	0	0	0	0	0	0	238
CL-02	0	1	0	0	0	0	0	0	0	0	314
CL-03	0	0	1	0	0	0	0	0	0	0	182
CL-04	0	0	0	1	0	0	0	0	0	0	256
CL-05	0	0	0	0	1	0	0	0	0	0	476
CL-06	0	0	0	0	0	1	0	0	0	0	458
CL-07	0	0	0	0	0	0	1	0	0	0	317
CL-08	0	0	0	0	0	0	0	1	0	0	358
CL-09	0	0	0	0	0	0	0	0	1	0	70
CL-10	0	0	0	0	0	0	0	0	0	1	303
CL-11	1	0	0	0	1	0	0	0	0	0	419
CL-12	0	0	0	1	0	0	0	0	1	0	141
CL-13	0	0	0	1	0	0	1	0	0	0	174
CL-14	1	1	0	1	0	0	0	0	0	0	187
CL-15	0	0	0	0	1	0	1	0	0	0	319
CL-16	0	0	0	0	0	0	1	0	1	0	294
CL-17	0	1	0	0	0	0	0	0	1	0	440
CL-18	0	0	0	1	0	1	0	0	0	0	84
CL-19	0	0	1	1	0	0	0	0	0	0	116
CL-20	1	0	0	0	0	0	0	0	1	0	260

TABLE I
SIMILARITY MATRIX S FOR THE EXAMPLE LIBRARY.

Image type identifier									
1	2	3	4	5	6	7	8	9	10
0.14	0.07	0.13	0.14	0.16	0.03	0.06	0.13	0.05	0.09

TABLE II
RELATIVE PROVISIONING FREQUENCIES FOR 10 IMAGE TYPES IN THE EXAMPLE.

In addition to the degree of overlap an important characteristic of the library is relative image popularity (defined as the relative frequency of an image being requested for provisioning). Denote these probabilities by α_i for $i = 1, \ldots, L$ and note that $\sum_{i=1}^{L} \alpha_i = 1$. TABLE II shows relative provisioning frequencies (image type popularity) for 10 image types considered in the example. Image type 5 is the most popular with probability of being chosen to be provisioned of 0.16, and image type 6 is the least popular with provisioning probability 0.03.

In order to allow a compact characterization of the degree of similarity across the image library that would also account for relative image popularity, we define a similarity index. We first define this metric for a single image type. It should reflect not only the degree of similarity with other image types (i.e. what fraction of the image can be rebuilt based on other images in the library), but also how popular are the images the overlap is with. I.e., if a given image has an overlap with another image, it is more beneficial if the other image has high provisioning probability since it results in higher probability of finding it in the hypervisor. Therefore, similarity metric $Sim(i, S)$ for

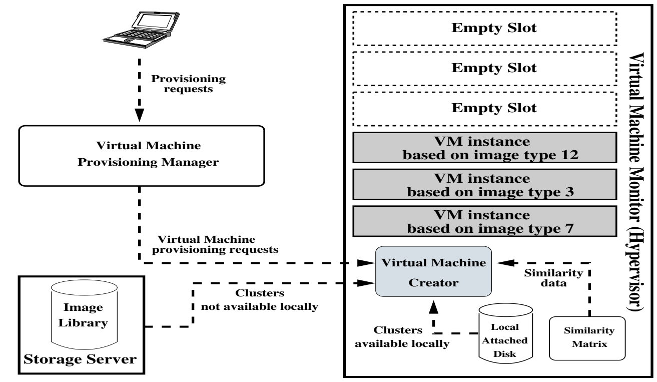

Provisioning requests
Virtual Machine Provisioning Manager
Virtual Machine provisioning requests
Image Library
Storage Server
Clusters not available locally
Empty Slot
Empty Slot
Empty Slot
VM instance based on image type 12
VM instance based on image type 3
VM instance based on image type 7
Virtual Machine Creator
Similarity data
Clusters available locally
Local Attached Disk
Similarity Matrix
Virtual Machine Monitor (Hypervisor)

Fig. 1. Virtual Machine provisioning system architecture (case of a single hypervisor with capacity of 6 virtual machines). Virtual machine instances are created using copy-on-write technology and based on read-only images. Read-only images, originally obtained from the storage server, can be used to create other images (leveraging image similarity information).

an image type i given similarity matrix S is defined as:

$$Sim(i,S) = \frac{\sum_{k=1}^{C}\left[S_{k,i}cl\_size(k)\sum_{l=1}^{L}S_{k,l}\alpha_l\right]}{\sum_{k=1}^{C}S_{k,i}cl\_size(k)} \tag{2}$$

Finally, we define combined measure for the entire library:

$$Sim(S) = \sum_{l=1}^{L} Sim(l,S)\alpha_l \tag{3}$$

III. Analytical Model of Local Content Availability for a Single Hypervisor

Architecture overview of the system we model is presented in Fig. 1. The system consists of four major components: virtual machine provisioning manager, storage server, virtualized host running hypervisor (in this example, with capacity for 6 virtual machines) and virtual machine creator component. Virtual machine provisioning manager receives virtual machine creation and deletion requests. Each creation request contains image type identifier that the new VM should be based on. The provisioning manager accepts the request and forwards it to the virtual machine creator. Virtual machine creator checks which parts of the image that is needed to instantiate the VM can be obtained from the local attached storage in the hypervisor. It can determine that using the similarity matrix that contains information about overlap across images in the library. For example, if virtual machine creator receives a request to instantiate VM based on image type 1 and the hypervisor contains images 2, 5, and 9 which can provide 78% of the content required to reconstitute image 1, it will copy the locally available blocks and then request the remaining 22% of the content from the storage server. Recall that due to using copy-on-write the original images obtained from the storage server are preserved.

We are interested in finding out what fraction of content required to instantiate a new virtual machine is available in the hypervisor's locally attached storage upon the request's arrival. This is the key metric that evaluates the quality of such provisioning scheme. The value depends on several aspects, such as utilization of the system, degree of similarity across images in the library, popularity (i.e., provisioning frequencies) of image types, and also number of VMs the hypervisor can concurrently execute. The key parameters and variables used in the model are presented in TABLE III.

Parameter	Description
λ	Mean arrival rate of provisioning requests
μ	Mean lifetime of virtual machine instances
M	Capacity of the hypervisor (in terms of number of virtual machines it can concurrently host
ρ	System utilization defined as $\frac{\lambda}{\mu * M}$
L	Number of image types in the library
α_i	Image provisioning frequency for image type i
S^i	Image similarity matrix for image type i
F_i	Random variable representing fraction of image available locally when request for image i starts being served

TABLE III
Parameters used in the analytical model.

Consider a set of L image types with associated provisioning probabilities denoted by α_i for $i = 0, \ldots, L$. Note that $\sum_{i=1}^{L} \alpha_i = 1$. An image i consists of n_i non-overlapping clusters s_k^i for $k = 0, \ldots, n_i$. In addition, for each image i we have a similarity matrix S^i of size n_ixL such that:

$$S_{k,l}^i = \begin{cases} 1 & s_k^i \text{ is part of image } l \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Note that the image similarity matrix S^i is a subset of overall library similarity matrix S (introduced in Section II) with only the rows having values of "1" in column i. The introduced notation helps in presenting the analytical model.

Define fraction of blocks required to rebuild image i that are available in the hypervisor (as parts of other image instances) as $F_i \in [0,1]$. We are interested in steady state expected value of F_i, denoted by $E[F_i]$. Denote the host's capacity by M. It represents the maximum number of virtual machines that can fit on the host. Denote the event that upon request to provision image of type i entering service (i.e., being provisioned on the hypervisor), the hypervisor contains cluster s_k^i as A_k^i. Then:

$$E[F_i] = \sum_{k=1}^{n_i} \frac{|s_k^i| P(A_k^i)}{\sum_{l=1}^{n_i} |s_l^i|} \tag{5}$$

where $|s|$ denotes the size of cluster s.

The probability of availability of a specific part of the image is a function of two key elements: number of images currently hosted on the hypervisor and also the number of images this fragment occurs in. Therefore, we first condition on the number of images on the hypervisor, obtaining:

$$P(A_k^i) = \sum_{q=0}^{\infty} P(A_k^i | Q = q) P(Q = q) \tag{6}$$

Since the maximum number of virtual machine images on a hypervisor upon request entering service is $M - 1$ and the probability depends only on the number of virtual images in service rather than in the queue, we can further simplify:

$$\begin{aligned} P(A_k^i) &= \sum_{q=0}^{M-2} P(A_k^i|Q=q)P(Q=q) \\ &+ \; P(A_k^i|Q=M-1)P(Q \geq M-1) \end{aligned} \tag{7}$$

The conditional availability $P(A_k^i|Q=q)$ can be computed as Bernoulli trial having no successes in q trials with success probability equal to sum of probabilities a given image contains fragment k of image i. Therefore:

$$P(A_k^i|Q=q) \;=\; 1-\left[1-\sum_{l=1}^{L} S_{k,l}^i \alpha_l\right]^q \tag{8}$$

The last element to compute is the distribution of number of virtual machines in the hypervisor. This depends on arrival and service processes (in our case virtual machine provisioning request arrivals and lifetimes of virtual machine instances). We continue the computation assuming M/M/m queuing system. If the constraints of the Poisson arrivals and exponential virtual machine lifetimes are not realistic, the model can be turned into an approximation by replacing M/M/m Equation 9 with, for example, diffusion approximation [3] or matrix-geometric approach [16]. This will enable us to model variance of arrival and lifetime processes. This extension is not presented in this article and left as a future work. Observe that in such case the model becomes an approximation not only because of the $P(Q=q)$ approximation but also because of the fact that PASTA property does not hold anymore with non-Poisson arrival process.

Assume that virtual machine provisioning requests arrive with mean rate λ and mean lifetime of virtual machine instances is μ resulting in $\rho = \frac{\lambda}{M\mu}$. Then, the queue size probabilities [14] are:

$$P(Q=q) = \begin{cases} p_0 & q=0 \\ p_0 \dfrac{(M\rho)^q}{q!} & 0 < q \leq M \\ p_0 \dfrac{M^M(\rho)^q}{M!} & q \geq M \end{cases} \tag{9}$$

where $p_0 = \left[\sum_{k=0}^{M-1} \frac{(M\rho)^k}{k!} + \frac{(M\rho)^M}{M!}\frac{1}{1-\rho}\right]^{-1}$.
Equation 5 with formulas from Equations 7, 8, and 9 form the final solution for $E[F_i]$ which we explore in empirical studies.

IV. Model verification and empirical studies

Analytical model presented in Section III provides a closed form expression (Equation 5) for the expected fraction of virtual machine image available at the local storage upon the provisioning request's arrival (in case of a single hypervisor). We now verify the model's correctness by comparing its predictions with results of the discrete event simulations. Next, we explore the model parameter space to gain insights into the circumstances under which the expected gain is substantial and therefore the use of such a provisioning scheme is beneficial.

Discrete Event Simulator

In order to evaluate accuracy of the analytical model we have implemented a discrete event simulator capable of generating sample evolutions of a stochastic system representing virtual machine hypervisor, virtual machines currently available (running) on this hypervisor, as well as details related to what fraction of content required to create a new virtual machine is available at the hypervisor's local storage. The simulator represents hypervisor as having M slots (as described in Section III) with each of the slots either empty or occupied by a virtual machine. The input to the simulator includes: mean arrival rate (λ), mean lifetime of a virtual machine (μ), number of virtual machines that can concurrently be hosted on the hypervisor (M), the number of images in the library (L), image provisioning frequencies (α_i for $i = 1, \ldots, L$), and image similarity matrix S (as introduced in Section II) representing the parts of each of the images that are shared with other images. The simulator is written in C++. Virtual machine provisioning request arrivals are generated using pseudo-random number generator and follow Poisson distribution. Virtual machine type (number from 1 to L) is also chosen randomly with frequencies matching α_i. Virtual machine lifetimes (i.e. amount of time virtual machine stays on the hypervisor) are also randomly generated and follow exponential distribution. For each provisioning request the simulator outputs fraction of the virtual machine image that was available locally. The values are reported together with image type. Based on this information we compute mean value based on large number of simulated arrivals.

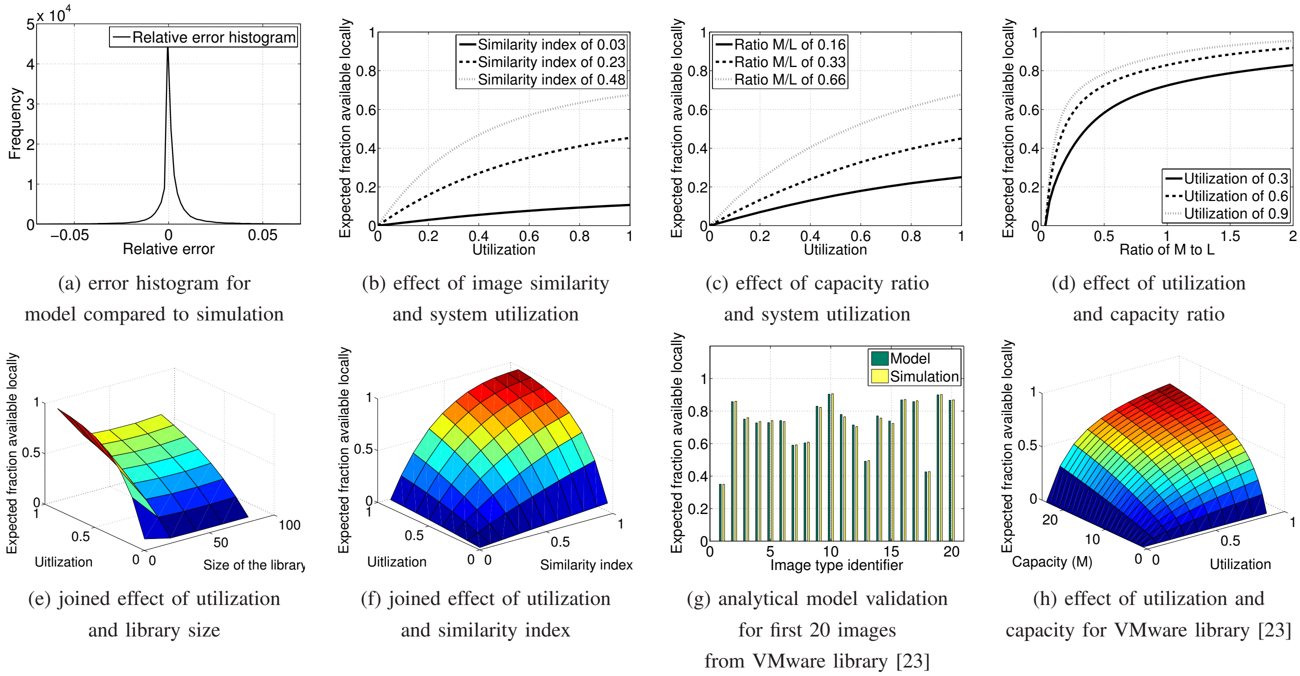
Model Evaluation

Model evaluation uses the discrete event simulator described above. We have generated 2750 configurations, that represent the parameter space, and compared the mean value obtained by averaging 10 runs of the simulator with the model. Each simulator run contains 10,000 request arrivals (therefore each configuration has been run with 100,000 arrivals given 10 simulation runs). We explored the following model parameters: $L = 10, 20, 30, 40, 50$, $M = 5, 10, 15, 20, 25$, $\rho = \frac{\lambda}{\mu} = 0.1, 0.2, 0.3, \ldots, 0.9$, number of rows in the similarity matrix was $3L$ (first L rows representing unique clusters with following $2L$ rows representing shared clusters), and probability that a given position in the overlap part of the matrix is 1 $pr = 0.0, 0.1, 0.2, \ldots, 1$. The last parameter simulated varying degrees of overlap among images.

The results of the evaluation are presented in Fig. 2a. It is a histogram of relative error between simulation results and values predicted by the model, i.e., for configuration i with simulation result sim_i and model result mod_i X-axis represents the ratio $\frac{sim_i - mod_i}{sim_i}$ and Y-axis represents frequency with which such difference occurs. The model is very accurate, with relative error well under 5%.

Insights from the Model

Analytical models, such as developed in this article, can be used to explore parameter space of the problem and discover



(a) error histogram for model compared to simulation

(b) effect of image similarity and system utilization

(c) effect of capacity ratio and system utilization

(d) effect of utilization and capacity ratio

(e) joined effect of utilization and library size

(f) joined effect of utilization and similarity index

(g) analytical model validation for first 20 images from VMware library [23]

(h) effect of utilization and capacity for VMware library [23]

Fig. 2. Analytical model validation (a), empirical studies based on synthetically generated similarity matrices (b-f), and empirical studies based on a set of images from VMware Marketplace library [23] (g) and (h).

quantifiable relationships among the parameters. In our case the key parameters affecting the expected fraction of content available locally are: utilization of the system, size of the library, similarity index, and capacity of the hypervisor (in terms of number of virtual machines that can be concurrently instantiated on the hypervisor). We examine the key relationships among these parameters. We chose three representative levels of low to high similarity, capacity ratio and utilization to illustrate sensitivity of gain.

First, consider the effects of utilization and similarity index. Fig. 2b shows results for varying utilization (X-axis) and three different levels of similarity index (0.03, 0.23, and 0.48). The other parameters of the model are fixed at $M = 4$ and $L = 100$. In all cases increasing utilization increases expected available fraction. Configurations with higher similarity index have significantly higher expected availability.

Another consideration is effect of capacity of the hypervisor M, in terms of number of virtual machines that can be run concurrently, and the size of the image library (i.e. number of image types L). Example results for $\frac{M}{L} = 0.16, 0.33$, and 0.66 are shown in Fig. 2c. Parameter $L = 30$. Again, utilization increases the expected availability and ratio of $\frac{M}{L}$ has also positive impact. The higher the M, the larger fraction of the image library is likely to be represented in the local storage of the hypervisor. Therefore, the odds of finding required content increase.

Next, consider effect of varying ratio of $\frac{M}{L}$ for three different utilization levels (Fig. 2d). As in the prior case, large ratio results in high expected availability of content, while the impact of utilization is most important for low ratios.

Fig. 2e shows joined effect of utilization and library size while keeping similarity index fixed. The highest expected content availability occurs for high utilization and small libraries with increasing library size having detrimental effect on availability. Similarly, Fig. 2f shows joined effect of utilization and similarity index. When both values approach 1, almost all of the content is available locally.

Application to an Example Image Library

We now focus on applying the model to the actual image library (rather than synthetically generated data as was the case in the previous subsection). We have obtained a representative sample of virtual machine images from a well known virtual appliance repository [23]. The images utilize several releases of operating systems: OpenSuse 11.1, Ubuntu 10.10, Ubuntu 10.2, and OpenSuse 11.3. Moreover, each virtual machine contains an installed application stack, such as Coppermine, Alfresco, DokuWiki, etc. These represent a broad range of open source applications that are a good sample of what a cloud service provider would have in the catalog.

We have derived a similarity matrix S for this set. A SHA1 hashcode of each of the 512 byte blocks of each image has been computed and then blocks with matching hashcodes were marked as identical (additional byte-by-byte checking would be needed in production system, but since the probability of SHA1 collision is very small it has not be done for the purpose of this study). Based on this computation 7611 clusters have been identified. The similarity matrix S derived this way became input to our model and the simulator. The similarity index for the entire library is 0.17 (given we did not have

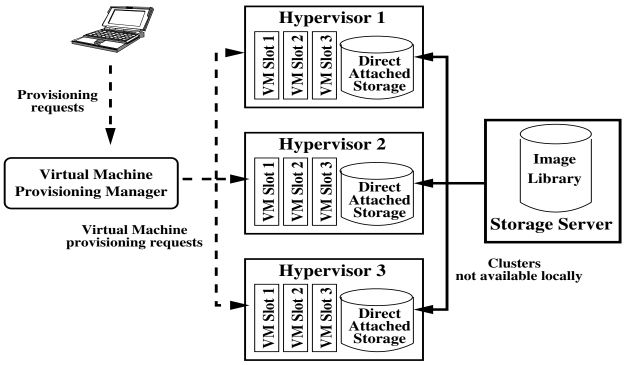


Fig. 3. Virtual Machine provisioning system architecture for an example cluster of 3 hypervisors with direct attached storage and capacity of 3 virtual machines.

image popularity probabilities, uniform popularity distribution was assumed for this computation).

To validate the accuracy, we have run both the simulator and analytical model assuming equal frequency of provisioning of each of the images and also exploring several values of parameters M and ρ. Fig. 2g shows results for an example run with parameters $M = 24$,$\rho = 0.87$. Fig. 2h shows effects of hypervisor capacity and system utilization for the example library.

V. Virtual Machine Provisioning Algorithm

The preceding analysis of a single hypervisor case suggests very significant opportunity for reduction of amount of network transfer from storage server while provisioning and therefore reduction of provisioning time and improvement in cost and runtime performance of the system (as compared to using copy-on-write access to network storage server). This section formulates a provisioning algorithm that uses image redundancy information to provision virtual machines more efficiently.

Fig. 3 presents virtualized cluster with hypervisors that are using direct attached storage to run virtual machine instances while leveraging storage server to retain image library. Each hypervisor can host a fixed number of virtual machines (as discussed in Section III). Provisioning requests are issued to the provisioning manager which routes them to a chosen hypervisor. The hypervisor reconstitutes image of the virtual machine and then creates a copy-on-write virtual machine instance based on the reconstituted image. In a typical system the reconstitution can be accomplished by copying image from the storage server to the hypervisor's DAS. The proposed algorithm instead of doing that attempts to rebuild the required image by copying data from other images already present in the hypervisor's DAS therefore reducing network transfer and provisioning latency. The algorithm is presented in Figures 4, and 5. Global variables used are listed in TABLE IV. The selection of hypervisor can be done in multiple ways. We have implemented and explored three policies: "first fit", "random", and "highest local fraction". The "first fit" policy selects the first hypervisor with an empty slot. The "random" policy selects a randomly chosen hypervisor among all having empty slots. Finally, "highest local fraction" policy selects hypervisor that has the highest fraction of content required to create the image being provisioned. It does so by using library similarity matrix (as described in Section II) and knowledge of which hypervisor has which images available. Runtime provisioning algorithm overhead for routing the provisioning request (using bit-set representation for images in a cluster) is equal to the number of hypervisors times the number of clusters on hypervisors ($O(H * C)$). To reconstitute the required image on the selected hypervisor, the algorithm selects all the clusters that are contained in the image. For each cluster, if there is an image available locally on the hypervisor containing this cluster, the algorithm selects the first such image and copies the blocks from that image into the target image (that is being

Variable	Description
L	number of images in the library
H	number of hypervisors in the cluster
C	number of clusters
$Policy$	routing policy
$AvailImages[L][H]$	true at $[l, h]$ iff image l is deployed on hypervisor h
$SimMatrix[C][L]$	true at $[c, l]$ iff cluster c is part of the image l

TABLE IV
Global variables used in the algorithm in Fig. 4.

```
let $t$ be identifier of image to be provisioned

procedure RouteProvisioningRequest($t$)
  switch($Policy$)
    case FIRST_FIT:
      for each hypervisor $h$ s.t. $empty\_slot(h)$ == true do
        $selectedHypervisor = h$;
      break;
    case RANDOM_SELECT:
      $selectedHypervisor$ = random_having_free_slot;
      break;
    case HIGHEST_LOCAL_FRACTION:
      let $highestLocalFraction = -1$;
      for each hypervisor $h$ s.t. $empty\_slot(h)$ == true do
        let $availableContent = 0$;
        let $requiredContent = 0$;
        for each cluster $c$ s.t. $SimMatrix[c, t]$ == true do
          for each image $i$ s.t. $AvailImages[i, h]$ == true do
            $availableContent$ += $cl\_size(c)$;
            break;
          end for;
          $requiredContent$ += $cl\_size(c)$;
        end for;
        $availableFraction = \frac{availableContent}{requiredContent}$;
        if $availableFraction > highestLocalFraction$ then
          $highestLocalFraction = availableFraction$;
          $selectedHypervisor = h$;
        end if;
      end for;
      break;
  end switch;
  $ProvisionVirtualMachine(t, selectedHypervisor)$;
  $AvailImages[t, selectedHypervisor]$ = true;
end procedure;
```

Fig. 4. Pseudo-code of hypervisor selection algorithm as executed by the provisioning manager. Pseudo-code of the $ProvisionVirtualMachine()$ procedure is shown in Fig. 5.

```
let $t$ be identifier of image to be provisioned;
let $h$ be identifier of hypervisor to provision to;

procedure ProvisionVirtualMachine($t$,$h$)
  if $Available[t, h]$ == false then
    // image not available locally, reconstruct
    for each cluster $c$ s.t. $SimMatrix[c, t]$ == true do
      $found\_local$=false;
      for each image $i$ s.t. $Available[i, h]$ == true do
        if $SimMatrix[c, i]$ == true then
          $found\_local$=true;
          $source\_image$=$i$;
          break;
        end if;
      end for;
      if $found\_local$ == true then
        $CopyBlocksFromLocal(c, source\_image, t)$;
      else
        $CopyBlocksFromStorage(c, t)$;
      end if;
    end for;
  end if;
  $CreateCopyOnWriteVritualMachine(t, h)$;
end procedure;
```

Fig. 5. Pseudo-code of virtual machine provisioning algorithm that reconstructs image on local storage using redundancy information.

created). For any clusters not available locally, the system copies the required blocks from the relevant image on the storage server. When using linked list representation of clusters in an image with a bit-set representation for hypervisors to which a cluster belongs the computation complexity is $O(C)$.

We have implemented the proposed algorithm as well as extended the discrete event simulator (discussed in Section IV) by adding capability to simulate multiple hypervisors and provisioning manager including three hypervisor selection policies described in prior sub-section. Each hypervisor runs a provisioning agent (written in Java) that tracks the current set of instances active on the hypervisor. An active instance on a hypervisor means that the image from which the instance was started is locally available for reconstitution when new instances with common clusters are required. The Virtual Machine Provisioning Manager keeps track of the similarity matrix that contains clusters common among images as well as the information from the agents about images locally available on each of the hypervisors. The similarity matrix computation consists of the following steps: 1) Computation of SHA1 hashcode for each of the 512 byte blocks of each image; 2) Identifying matching hashcodes and marking the related blocks as identical; and 3) Creation of a set of non-overlapping clusters to produce similarity matrix described in Section II. When images are added to or removed from the storage server, the similarity matrix is incrementally updated.

VI. Provisioning Algorithm Evaluation

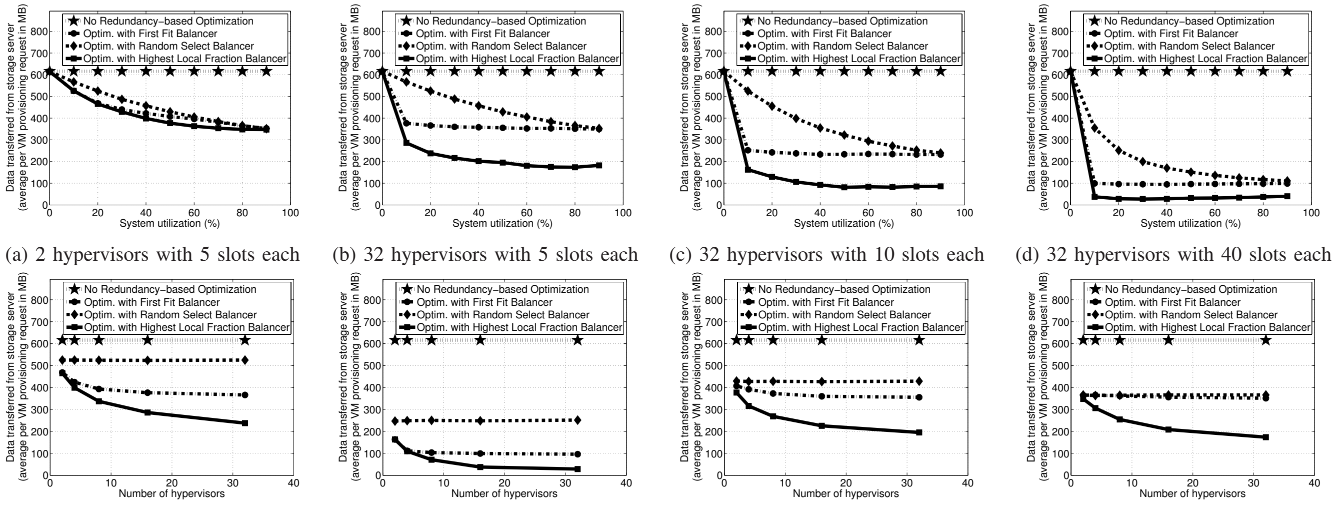
In order to compare the performance of the proposed algorithm with copy based provisioning that does not use redundancy information as well as to explore impact of VM placement policies we have performed an extensive simulation study. All simulations use 77 VMware Marketplace images [23] and explore various configurations in terms of number of hypervisors in the cluster, capacity of hypervisors (in terms of the number of virtual machines it can concurrently host), as well as system utilization. Popularity of images was set to uniform distribution (i.e., each of the images equally likely to be provisioned). Provisioning request arrivals and duration of time between provisioning and deletion of virtual machines were generated using Poisson and exponential distributions. Each simulation run contained 10,000 provisioning requests.

The results are presented in Fig. 6. Y-axis on each of the plots represents average amount of data transferred from the storage server to hypervisor per a single provisioning request. This value is critical to both provisioning performance as well as cost of the system (the more data needs to be transferred the more storage server bandwidth has to be used for provisioning requests). Gain in VM provisioning time is directly proportional to the fraction of data locally available (instead of network copy, local storage copy is utilized). In our experimental setup we found the network transfer to be 6 times slower than local storage transfer. Exact quantification requires modeling of storage subsystem, including ratio of local storage to network storage transfer bandwidth. This is out of scope of this paper and left as a future work.

The X-axis on plots (a)-(d) represent overall system utilization (in terms of time-averaged fraction of taken virtual machine slots on hypervisors). The X-axis on plots (e)-(h) represent number of hypervisors in the cluster.

The ”No Redundancy” method always copies all blocks from storage server regardless of whether they are available locally. Since average size of image (in sparse file format) for the library is 603MB, this policy results in constant average amount of data transferred (per request) independent of system utilization and cluster configuration (denoted by dashed lines with stars on all of the plots). This results in high network bandwidth usage and therefore highest provisioning time and represents “naive” copy based provisioning.

The three mechanisms that use redundancy elimination result in performance gains during provisioning because of partial local reconstitution of target image. The methods use local reconstitution from clusters available locally and only copy the blocks from storage server when blocks are not available locally. The “random select” option has the weakest performance. The reason is it is spreading the virtual machines across all hypervisors even in low-load conditions. In such case the odds of finding required content locally decrease. “First-fit” balancer attempts to keep the used hypervisors at full capacity (by provisioning to first available slot). Because of that hypervisor utilization is kept higher than in “random select” and results in higher odds of finding redundant content. The difference between the two policies decreases as system utilization grows. Finally, the “highest local fraction balancer” policy performs best. It is due to the fact that as the system operates, hypervisors become “specialized” in hosting images that form similarity cliques. Note that this is “emergent performance” behavior, where similar virtual machines tend to be provisioned to same hosts. Overall, the reduction in transfers from storage server can be as high as 80% for highly



(a) 2 hypervisors with 5 slots each (b) 32 hypervisors with 5 slots each (c) 32 hypervisors with 10 slots each (d) 32 hypervisors with 40 slots each

(e) 20% utilization with 5 slots (f) 20% utilization with 40 slots (g) 50% utilization with 5 slots (h) 80% utilization with 5 slots

Fig. 6. Results of experimental evaluation of redundancy-based virtual machine provisioning algorithm described in Section V.

utilized large clusters with large number of virtual machine per hypervisor slots.

VII. Related Work

The key area of related work is redundancy detection allowing to identify parts of virtual machine images that are shared between multiple images in the library. This is similar to data compression in the sense that only one copy of the data is retained and duplicate data is removed leaving references to the first copy. This approach reduces the required storage capacity and also allows to manage the library as data therefore leading to simplification of library management processes. A very good example of such approach is Mirage system [18].

The library redundancy is investigated here for a different reason. We want to evaluate the impact of redundancy elimination on virtual machine provisioning speed and network bandwidth consumption. File level de-duplication is intended to eliminate redundant files on a storage system by saving only a single instance of the file. Another option is segment level de-duplication which eliminates partial duplicate segments or pieces of the file thus storing only new segments and indexes to the original segments to reconstruct the file. Splitting of data into segments can be done either using fixed length blocks or variable length segments. A very good discussion of the redundancy elimination can be found in [8]. Other interesting sources on this topic are [9], [4].

Another related area is resource management. Static allocation approach is used in [19] where authors apply vector bin-packing heuristic to minimize the number of servers required to host a given web traffic. [5] introduces a concept of resource economy in the web server hosting center. Resource overbooking is advocated in [21] as a means of increasing the revenue generated by available resources in a shared hosting platforms. Work of [20] presents an integrated resource management framework for quality of service in network services. In case of this article the focus is on provisioning latency while not introducing resource over-booking at runtime, rather than on the placement problem itself.

VIII. Conclusions and Future Work

We have presented and evaluated a virtual machine provisioning system that uses similarity across virtual machine images to minimize amount of data that has to be transmitted from the storage server to the hypervisor upon VM instantiation. A single-hypervisor analytical model is formulated and validated. It provides important insights into the circumstances in which such a scheme can be beneficial. The key parameters affecting the performance are utilization, hypervisor capacity, degree of similarity among the virtual machines, and image provisioning frequencies. Higher level of utilization increase likelihood that there are other VMs in the hypervisor and therefore increases odds of finding content required to create a new VM. Degree of image similarity has similar effect - the higher the overlap the more benefit from the redundancy driven provisioning scheme. Cloud environments often use local storage because it is cost efficient. Proposed scheme can be utilized for efficient transfer of VM disks during live migration.

Algorithm is implemented in a testbed and also validated using extensive discrete event simulations based on a library representative of typical Cloud provider's catalog. New virtual machine placement policy based on degree of image similarity is proposed and evaluated. The system achieves up to 80% reduction in amount of data transferred from the storage server to hypervisors. It is especially effective for large and highly utilized hypervisor clusters.

Our future research plans include considering lazy deletion, thus taking advantage of caching of images. Here the read-only source for selected instances is retained locally even when the instances are deprovisioned. This further increases odds of finding required clusters when provisioning new instance. We plan to explore the potential effect of this improvement and also perform larger scale experiments on a test bed.

REFERENCES

[1] Kernel Virtual Machines. Online. http://sourceforge.net/projects/kvm.

[2] Amazon Inc. Amazon Elastic Compute Cloud. Online, 2009. http://aws.amazon.com/ec2/.

[3] H. Baruch and W. R. Franta. A Diffusion Approximation to the Multi-Server Queue. MANAGEMENT SCIENCE Vol. 24, No. 5, January 1978, pp. 522-529.

[4] J. Bonwick. ZFS Deduplication. Online, 2009. http://blogs.sun.com/bonwick/entry/zfs_dedup.

[5] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin Vahdat, and Ronald P. Doyle. Managing Energy and Server Resources in Hosting Centres. *in Proc. of OSDI*, 2001.

[6] Microsoft Corp. Microsoft Virtualization. Online, 2011. http://www.microsoft.com/virtualization/.

[7] R. Creasy. The Origin of the VM/370 Time-Sharing System. *IBM Journal of Research and Development*, 1981.

[8] Fred Douglis, Jason Lavoie, John M. Tracey, Purushottam Kulkarni, and Purushottam Kulkarni. Redundancy elimination within large collections of files. In *In USENIX Annual Technical Conference, General Track*, pages 59–72, 2004.

[9] EMC. Data Domain Replicator Software, Network-efficient replication for backup and archive data. Online, 2011. http://www.datadomain.com/pdf/DataDomain-Rep-Datasheet.pdf.

[10] Gartner Inc. Special Report on Cloud Computing. Online, 2011. http://www.gartner.com/technology/research/cloud-computing/.

[11] R. Goldberg. Survey of Virtual Machine Research. *in IEEE Computer Magazine*, 1974.

[12] P. Gum. System/370 Extended Architecture: Facilities for Virtual Machines. *IBM Journal of Research and Development*, 1983.

[13] IBM. IBM SmartCloud. Online, 2011. http://www.ibm.com/cloud-computing/us/en/.

[14] L. Kleinrock. *Queueing Systems. Volume 1: Theory.* , 1975.

[15] KVM. Qemu Image Formats. Online, 2011. http://en.wikibooks.org/wiki/QEMU/Images.

[16] M. Neuts. *Matrix-geometric solutions in stochastic models, an algorithmic approach.* The Johns Hopkins University Press, 1981.

[17] Rackspace. Rackspace Cloud. http://www.rackspace.com/cloud/, 2011.

[18] 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 Proc. of USENIX Virtual Execution Environments Workshop*, 2008.

[19] J. Shahabuddin, A. Chrungoo, V. Gupta, S. Juneja, S. Kapoor, and A. Kumar. Stream-Packing: Resource Allocation in Web Server Farms with a QoS Guarantee. *Lecture Notes in Computer Science*, 2001.

[20] K. Shen, H. Tang, T. Yang, and L. And. Integrated Resource Management for Cluster-based Internet Services. *in Proc. of the OSDI*, 2002.

[21] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource Overbooking and Application Profiling in Shared Hosting Platforms. *in Proc. of OSDI*, 2002.

[22] VMware. Online. http://www.vmware.com.

[23] VMware Inc. VMware Virtual Appliance Marketplace. Online, 2011. http://www.vmware.com/appliances/.

[24] Xen. Online. http://www.xensource.com.