Probabilistic Approach to Network-Based Virtual Computing

Chung-Ping Hung* and Paul S. Min†
Department of Electrical and Systems Engineering, Washington University in St. Louis
One Brookings Drive, St. Louis, MO 63130, USA
Email: *chung23@wustl.edu †psm@wustl.edu

***Abstract*—In this paper, we propose a probabilistic approach to desktop virtualization. Like the application streaming, applications that are stored and managed in the server are uploaded to and ran on remote devices in speculative orders to achieve high performance computing. To ensure the compatibility, the required portion of the operating system is packed and uploaded together with the applications.**

A look-up table (LUT), which reflects the probabilistic behaviors of code blocks during the execution of applications, is used. In order to reduce the computational complexity associated with the LUT (and thus the latency), the initial LUT with a very large number of entries is successively substituted with smaller LUTs while preserving approximately the same quality of information. We also propose the use of hierarchical LUTs whereby a large LUT can be separated into multiple smaller LUTs to improve performance and/or reduce cost.

***Index Terms*—virtual computing, memory management, computer network, information technology.**

I. Introduction

Today, people use computers on daily basis whether they realize it or not. Other than traditional computers, cell phones, PDAs, portable media players, GPS, automotive electronics, and household appliances also provide IT resources.

Companies around the world have developed hardware and software to make standards to ease the integration of those resources [9], but rapid changes in technology make it virtually impossible for ordinary users to keep up. There are significant challenges in providing information technologies that are easy to use for ordinary users.

This paper aims at proposing a novel server-client architecture that reduces the complexity of information technology while providing high performance. While several claims have been made on this front, the solutions proposed thus far adopted piece-meal approaches for specific applications [6], [7], [9] limiting the applicability and scope. The standards lack the generality needed to make them adaptable with all kinds of operating systems and processor types. Many of these attempts resulted in poor performance, not acceptable in today's computing environment.

The particular paradigm this paper is focused on is based on desktop virtualization [6], [7]. Departing from the 1960s-style terminal architecture, desktop virtualization adds a layer of function known as the virtual machine monitor [6], [7]. The virtual machine monitor interacts with the operating system in the server to display a desktop remotely on the client machines. The virtual machine monitors also manage input/output (IO) functions at the client machines executed by keyboards, mice, and video displays. On a client machine, the user sees a desktop like the one that appears when an ordinary computer is turned on, but this desktop is emulated by the server interacting with the virtual machine monitor. The client machine acts like a simple display device with keyboard and mouse. From the remote desktop, the user can utilize software applications that reside in the server, without having to install and manage them locally.

The main benefit of desktop virtualization in IT strategy is clear such as software resources can be centrally managed to save maintenance and repair cost. Since the client machines interact with the virtual machine and not directly with the software applications, the client machines can be made simple and inexpensive. Without the complex operating system functions, the client machines are more efficient and reliable for ordinary users.

We briefly introduce the background and related work on desktop virtualization world in Section II and III, respectively. Considering to the challenges we states in Section II, we set up our design objectives in Section IV, and then describe our proposed architecture based on them in Section V. Speculative page uploading, which is the key feature of the proposed architecture, is described in Section VI. We also propose two algorithms to alleviate implementation difficulty and improve performance in Section VII and VIII. Finally, there are contributions and future work in Section IX and X, respectively.

II. Background

There are two common architectures of desktop virtualization, as shown in Fig. 1. The architecture shown on the left in Fig. 1 resembles the traditional terminal architecture wherein the client machine's main job is performing the I/O functions. Based on the inputs received from the client machine, the server runs software applications and sends the outputs back to the client machine for display or other forms of output (e.g., audio). This architecture reaps the benefits of traditional server-client architecture such as the ease of management and cost reduction. The drawback of this architecture is the large amount of data that needs to be exchanged between the client machine and the server across the communication link, which often is unreliable. This results in potentially slow and unpredictable interactions between the client machine and the

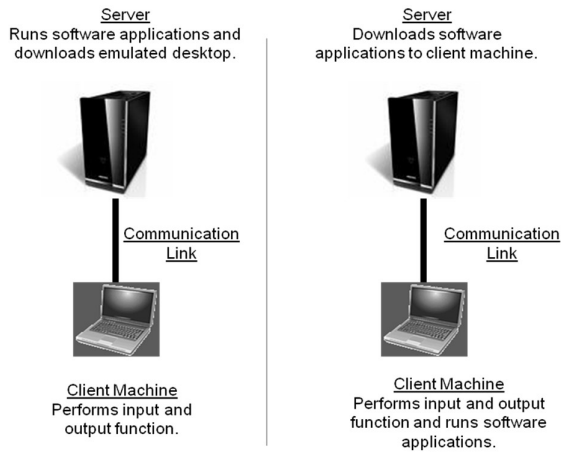

Fig. 1. Common architectures of virtual computing.

server. Today, most of the browser-based applications in the Internet use the architecture shown on the left in Fig. 1.

The architecture depicted on the right in Fig. 1 is known as the automated software distribution or application streaming. When a user chooses to run a software application in the server, the server uploads the selected software application over the communication link and the software application is run on the client machine using the local processor(s). This architecture improves interaction time for the users since the software application is run locally. A drawback for this architecture is that a significant portion (40% or more in current implementations) of the software application must be first uploaded, which delays the start-up process significantly. Depending on the performance of the communication link, there may be substantial delay before the software application is downloaded to the client machine. Another drawback is that since the software application is run on the client machine, it might be sensitive to the operating system's configuration and stability of the client machine, which leaves the responsibility of operating system maintenance to the users.

It is clear that virtual computing presents substantial opportunities for high performance computing and ease of IT management. Currently, however, there is no method of virtual computing that provides the necessary performance, reliability, and convenience that are expected from the users. The resulting architecture must have the look and feel of the computers that today's users are accustomed to. Without this, wide spread acceptance of virtual computing may remain elusive.

III. Related Work

The early days of virtual computing was based on mainframe computers [1]. Using a centralized mainframe computer, a number of remote terminals emulate the mainframe remotely. In this method, the remote terminals are connected directly to the mainframe using dedicated cables. The computing resource in the mainframe is simply shared among the remote terminals by time-division multiplexing.

As the Internet proliferated in 1990, network-based approaches to virtual computing emerged [2]–[11]. In this method, the remote terminals are not connected directly to the computing resource. By using the network connection available to the remote terminals, any computing resources in the Internet can be accessed and used.

For example, using network browsers, users can access computing resources located anywhere in the Internet. However, as anyone who has surfed the Internet can testify, the performance of network browser can be highly unpredictable. [4], [6].

In [2], [10], authors discuss a new problem of security arising from virtual computing. In [3], [8], [12], authors describe issues related to application streaming.

In [12], authors propose a novel virtual web service architecture which integrates web services without user awareness. The users can access the web services with the same experience they are used to without manually switching around the service providers.

In [13], authors take the advantages of virtualization and further integrate the Java Virtual Machine technology into the operating system. We believe it is the future of virtualization computing.

While virtual computing is touted as the solution to manage extreme complexity in today's computing, there is no clear approach reported to date that address the performance, convenience and cost involved in virtual computing.

IV. Objectives of Our Solution

The main aim of this paper is to develop a novel method of desktop virtualization that improves the performance, reliability, and convenience, and at the same time, addresses the challenges stemming from diverse hardware and software. In this paper, we focus on the following three objectives for the desktop virtualization method to be developed.

Objective 1
: Software applications and general operating system components should reside centrally at the server in the proposed architecture, which enables skilled IT professionals to manage them efficiently.

Objective 2
: Personal data and files are store in the client machines, since some users would feel more comfortable storing their personal data and files locally in their own machines rather than in remote servers controlled by somebody they don't know. Privacy and security are top concerns of most users.

Objective 3
: In the proposed architecture, software applications are run in the client machines, where computational resources are dedicated for them without incurring long transmission delay.

V. Proposed Architecture

Based on the above-stated objectives, Fig. 2 reflects a high-level depiction of the proposed method. The proposed architecture employs a server wherein software applications and operating systems reside. The client machines store personal

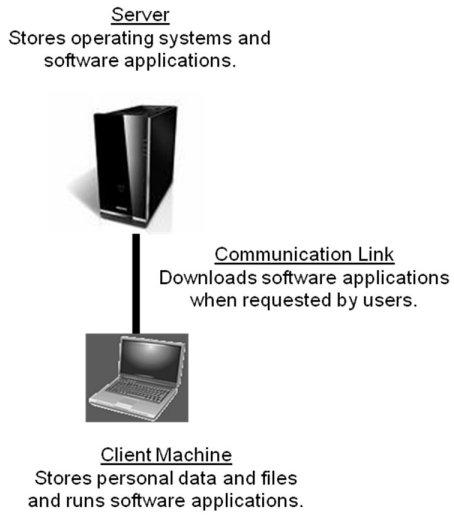

Fig. 2. High-level depiction of the proposed architecture

data and files, and have limited device system functions such as boot loader, file management, and I/O.

When a software application is running on any computer (real or virtual), the operating system assigns a memory structure with which the processor(s) interacts. This memory structure is known as the virtual memory (VM). From the VM, the processor fetches the instructions and data, responses to the inputs, writes intermediate results, invokes output routines etc. All of processing done by the processor is based on the VM. For Windows XP, the VM is defined over the address range of 00000000–FFFFFFFF, of which 00000000–7FFFFFFF is the user space and 80000000–FFFFFFFF is the kernel space. There are a total of 4GB VM addresses defined for each Windows XP process.

The VM is organized in terms of page. For example, for Windows XP, each page corresponds to 4KB of data. For each application, some pages of the VM are specific to individual users, some are populated by the operating system, and some are populated during the run time. Once an ISA-compatible machine has the same VM image and limited system level compatibility (e.g., providing device drivers in case of the application software performing low-level access), it can execute the same application software and get expected results regardless of who creates the VM space. In other words, the VM space is an instance which represents the major information about running a process; the proposed virtualization becomes the matter of how VM space is created, transferred, and accessed.

In the proposed architecture, the VM space should be created by the server since memory management is handled by the operating system, which should be managed by IT personnel centrally based on Objective 1. Similar to the VM space in a standalone computer, some of the pages are mapped to components that belong to the operating systems and the application software originally stored in the server while some other pages are mapped to user's personal data and files, which are physically stored in the client machine to satisfy Objective 2.

In order to achieve Objective 3, the context in the VM space, or at least the pages required immediately to continue execution, should be made available at the client's machine. In other words, there might be more pages of data transferred from the server to the client machine over the Internet compared to conventional application streaming architectures.

Furthermore, the client's machine would request the pages from the VM space in various orders during the runtime. If the server uploads every page physically unavailable at the client's machine, which is similar to the on-demand paging we apply on current memory management, the performance would be intolerably low since the transmission latency over the Internet is thousands of times slower than that of the local hard drive bus. Therefore, we need to investigate a speculative page uploading algorithm to reduce the chance of on-demand transmission and thus improve the runtime performance.

VI. Speculative Page Uploading

A. Observations

A typical VM space in 32-bit Windows XP (i.e., 4GB space consisted of 4KB pages) can hold 1M pages. Theoretically there will be $2^{20}!$ possible page access sequences within the VM space, which is difficult, if not impossible, to manage. However, three properties make it possible to manage the probability model of page access sequences. First, most of the pages are never accessed, which significantly reduces the number of possible page access sequences. Second, some pages are considered essential that have to be accessed anyway. Third, some pages tend to be followed by or follow particular ones; others might never or unlikely to be accessed before or after particular ones.

To explore and utilize these properties which reduce uncertainty of memory access model, we have to profile the memory usage behavior of the application software.

B. Profiling

Fortunately, Oracle provides *truss*, which is a powerful tracing facility, integrated in Solaris and OpenSolaris. We can get page access sequences from a program starts, with runtime user interactions, till it ends, by tracking and recording the page fault addresses (since Solaris is a pure on-demand paging operating system.) If we keep track of enough of these page access sequences, we can characterize and establish the probabilistic model of memory usage per application software and user.

C. Algorithm

Fig. 3 illustrates the traditional on-demand paging protocol applied to network based desktop virtualization. As we can see, the server only sends the page requested by the client each time, which leaves vast idle periods due to the round-trip delay of the network.

In order to utilize the idle periods, we propose an algorithm illustrated in Fig. 4. Once the client machine tries to access

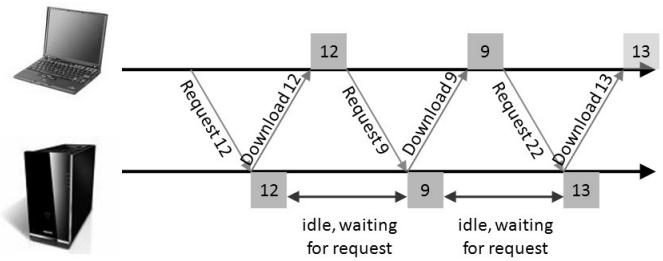

Fig. 3. On-demand page download scheme.

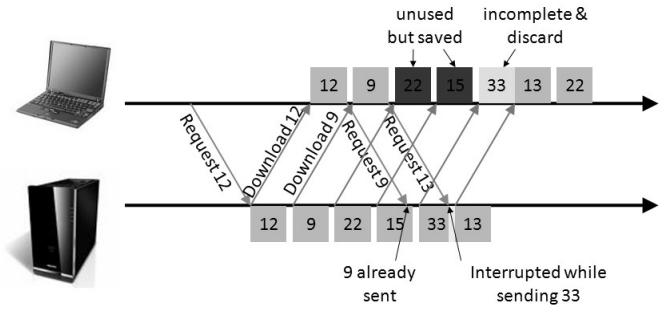

Fig. 4. Speculative page download scheme.

a page that is currently unavailable locally, it sends a request for the page to the server. The server does not only reply with the page, but also uploads a series of other pages, which are considered the most likely ones to be accessed thereafter according to the probability model. If the prediction is correct, the client machine can continue the execution without the relay involved in requesting the next page. If the prediction is incorrect, the client machine sends the request for the next page, just as it does using the conventional on-demand paging, and stores the incoming pages for potential future use.

D. Probability Model Management

We can get a list of potential page access sequences with their occurrence probabilities by using *truss* in Oracle Solaris or OpenSolaris to trace and count the page faults of sufficient runtime samples. In the proposed architecture, the probability model is managed by a special look-up table (LUT). Each entry of the LUT represents a particular page access sequence. Once the server gets a client machine's request, including the current required page number and the page usage history, it looks through the entries, finds the one whose prefix matches the client's request with highest probability, and then sends the requested page and the following ones in that entry, i.e., the most probable page series that the client might need at the time. In other words, we expect the page access behavior to be Markovian.

Managing and searching from the LUT, however, is not going to be easy due to the huge number of entries, even if the number of potential page access sequences has been significantly reduced from the worst case. Therefore, we need further algorithms to reduce the LUT's size without significantly degrading the speculation accuracy.

VII. LUT Reduction

Almost everything in the world can be modeled in a Markov chain. However, the model can be difficult to implement if too much information is carried in states.

If the server sends a series of pages according to the full page access history from the beginning of the application software in the proposed architecture, the Markov model would be too complex to handle. Therefore, we try to create a simplified Markov model, which carries shorter page access history in its states while providing approximately equal prediction accuracy.

Here is our observation of a hypothetic memory access model. Assume we have N pages of data and M possible page access sequences $\{Seq_0, Seq_1, \ldots, Seq_{M-1}\}$, where $M \ll N!$. We also randomly assign P_m, where $m \in \{0, 1, \ldots, M-1\}$, to be the probability of each Seq_m's occurrence. Each sequence represents an outcome, which is the following page access behavior in this case.

Intuitively, we can always pick up the right outcome based on the information provided by the entire known page access history. However, how well we can do if we can only remember several pages recently accessed depends on the mutual dependency between the full history and the recent partial one, which can be measured by mutual information.

The original equation of mutual information is given by:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p_1(x) p_2(y)} \right) \qquad (1)$$

where X and Y are random variables represent the full and partial page access histories, respectively. In our case, however, each y is the postfix of the corresponding x, such that (1) becomes the following equation:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p_1(x) \log \left(\frac{1}{p_2(y)} \right) \qquad (2)$$

The mutual information between different orders of y and full page access history x in our 32-page experiments, which have different numbers of possible sequences, is represented in Fig. 5. As we can see, the mutual information gets "saturated" in certain order of y, i.e., we cannot get more information and thus more accurate prediction by remembering longer sequences. Also, the more possible sequences we have, which means the less even in our probability distribution, the higher saturation order we get.

The idea is similar to lossy data compression techniques. We first analyze the full model using mutual information approach, and then select a proper order of page access history to keep track of. Therefore, we can have approximately the same prediction quality with a simpler Markov model, which can be implemented in a smaller LUT, and thus reduce the cost in keeping and addressing records.

A. Hierarchical LUT

After we simplify the Markov model and reduce the size of the LUT that represents the page access model, we may

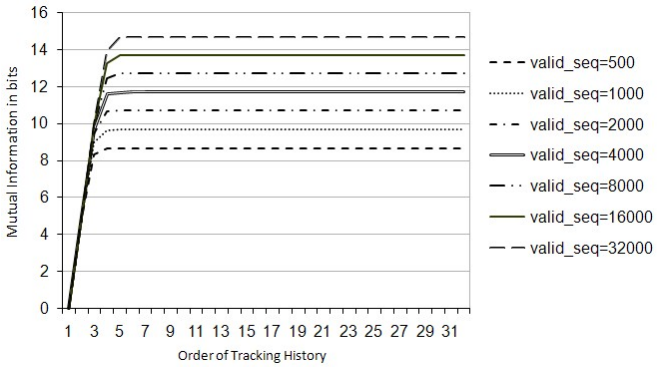

Fig. 5. Mutual information of different number of random sequences and different orders of tracking history.

find that preserving all of the information in a single LUT is still not feasible within the given memory technology. We now consider separating one LUT into multiple LUTs in hierarchy, i.e., storing the LUT in multiple levels of storage elements which are different in speed and capacity. In this paper, we simplify the problem by separating one LUT into only two levels. One is the fast portion, which may be implemented by fast and expensive technology (e.g., semiconductor memory), and the other is the slow portion, which may be implemented in bulk storage devices (e.g., hard drives).

Let T_{av} be the average transaction time.

$$T_{av} = P_{fast} \cdot T_{fast} + (1 - P_{fast}) \cdot T_{slow} \tag{3}$$

where P_{fast} is the probability of accessing the fast portion and T_{fast} (T_{slow}) is the transaction time for the fast (slow) portion.

Intuitively, the optimal algorithm would be first sorting the entire entries in the LUT by the occurrence probabilities from high to low. Then, the entries with high occurrence probabilities are put in the fast portion, and the other entries are put in the slow portion. We can, therefore, minimize T_{av} by maximizing P_{fast}. However, we have to sort the entire LUT by occurrence probabilities in the beginning, and again whenever the probabilities change, which is an unsustainable overhead for the LUT, in this approach. In order to reduce the complexity, we propose an alternative algorithm to partition the LUT.

B. Our Markov Model Properties

Fig. 6 shows an example of the Markov chain in the proposed algorithm. Since we have to keep track of the page accesses from the beginning, our Markov chain is similar to a tree structure. The Markov chain starts from an imaginary root state, and then goes to multiple first-level states, which represent the earliest page we can trace, with a probability distribution. Each first-level state also has multiple paths going to second level states, which represent the second page we can trace given the first page accessed before, with a probability distribution, and so on. Corresponding to each leaf node, there is an output value, i.e., the pages most likely to be accessed following the page access history. If we have long enough page access history, the state transitions only occur between the leaf nodes according to the latest incoming information.

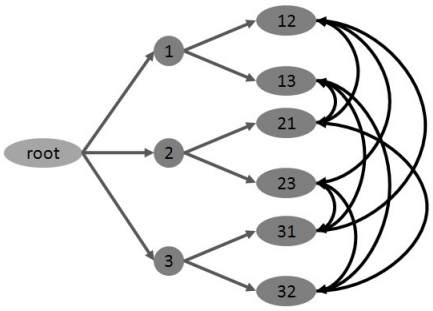

Fig. 6. A simplified example of our Markov chain structure.

C. LUT Separation Algorithm

Once we have the tree structure and the probability distributions, we can store the full Markov model in fast and slow LUTs accordingly. The proposed algorithm is described below:

1) Assume all states are stored in the fast LUT.
2) Analyze the probability distribution of the outgoing paths from the root node.
3) Remove the subtrees which are rooted from the outgoing paths with relatively low probabilities, to the slow LUT.
4) Redo steps 2 and 3 for each next level node, which is remain in the fast LUT, until reaching the leaf nodes.

D. Comparison

Comparing to the optimal approach, the proposed algorithm has some pros and cons:

Pros:
1) Assume we have total N pages and $(N - M)$ pages are monitored in our model. Instead of sorting P_{N-M}^{N} entries in the worst case, the proposed approach only needs to sort $(N - m + 1)$ entries C_{m-1}^{N} times for the m_{th} level, which significantly reduce the computational complexity.
2) Each sorting can be done independently, i.e., we can significantly speed up the process by utilizing parallel computing technologies.
3) We can limit the portion that needs to be updated in case of minor probability changes instead of processing the whole LUT again.

Cons:
1) The proposed algorithm is not optimal; we cannot guarantee that all entries in the slow portion have lower probabilities than any entry in the fast portion.
2) We have to develop a fast analyzing algorithm to keep the computational complexity low.
3) We cannot know the coverage rate and the entry number of each portion from any preset coefficient before we run the first pass.

E. Analyzing the Probability Distribution

We introduce three standardized models which can characterize the unknown probability distributions with some degree of accuracy in the proposed algorithm's step 2 and help us to determine whether its element has relatively low probability or not.

Model A

The first model is illustrated in Fig. 7(a). As we can see, of the possible n points, some points have the same nonzero probabilities while the others have zero probabilities. The probability distribution function is given by

$$P_1(x) = \frac{1}{\hat{n}} \text{ for } x = 1, 2, \dots \hat{n} \tag{4}$$

where $\hat{n}$ is the number of points with nonzero probabilities. In Model A, the entropy provided by the same $\hat{n}$ is maximized.

$$H(P_1(x)) = -\sum_{x=1}^{\hat{n}} \left\{ \frac{1}{\hat{n}} \cdot \ln\left(\frac{1}{\hat{n}}\right) \right\} = \ln(\hat{n}) \tag{5}$$

Model B

In this model, the probability of each element is exponential decreasing within a limited range. The distribution function is given by

$$P_2(x) = ar^{x-1} \text{ for } x = 1, 2, \dots n \tag{6}$$

where

$$a = \left(\frac{1-r}{1-r^n}\right) \quad 0 \le r < 1 \tag{7}$$

Parameter r represents the degree of concentration for all Model B's. Model B with $n = 7$ and $r = 0.5$ is illustrated in Fig. 7(b).

The entropy of model B is

$$\begin{aligned} H(P_2(x)) &= -\sum_{x=1}^{n} \left\{ ar^{x-1} \cdot \ln\left(ar^{x-1}\right) \right\} \\ &= \ln(1-r^n) - \ln(1-r) \\ &\quad - \frac{(n-1)r^{n-1} - nr^n + r}{(1-r)(1-r^n)} \cdot \ln(r) \end{aligned} \tag{8}$$

Model C

In this model, the probability of each element is linearly decreasing within a range. The distribution function of Model C is given by

$$P_3(x) = \begin{cases} a - k \cdot (x-1) \text{ for } x = \{1, 2, \dots, \tilde{n}\} \\ 0 \qquad \text{otherwise} \end{cases} \tag{9}$$

where

$$\begin{aligned} a &= \frac{1}{\tilde{n}} + \frac{k \cdot (\tilde{n}-1)}{2} \\ \tilde{n} &= \min\left\{ \left\lfloor \frac{1 + \sqrt{1 + 4 \cdot k^{-1}}}{2} \right\rfloor, n \right\} \end{aligned}$$

There are two types of Model C which are illustrated in Fig. 7(c) and (d), where $\tilde{n} = n = 7$, $k = 0.01$, and $n = 7$, $\tilde{n} = 4$, $k = 0.1$, respectively. For the probability distribution shown in Fig. 7(c), all elements have nonzero probabilities; in Fig. 7(d), the probabilities decrease so sharply that the last three elements have zero probability.

The entropy of Model C is

$$\begin{aligned} &H(P_3(x)) \\ &= -\sum_{x=1}^{\tilde{n}} (a - k \cdot (x-1)) \cdot \ln\left(a - k \cdot (x-1)\right) \end{aligned} \tag{10}$$

Unfortunately, there is no way to further simplify the entropy of Model C as in the previous two cases.

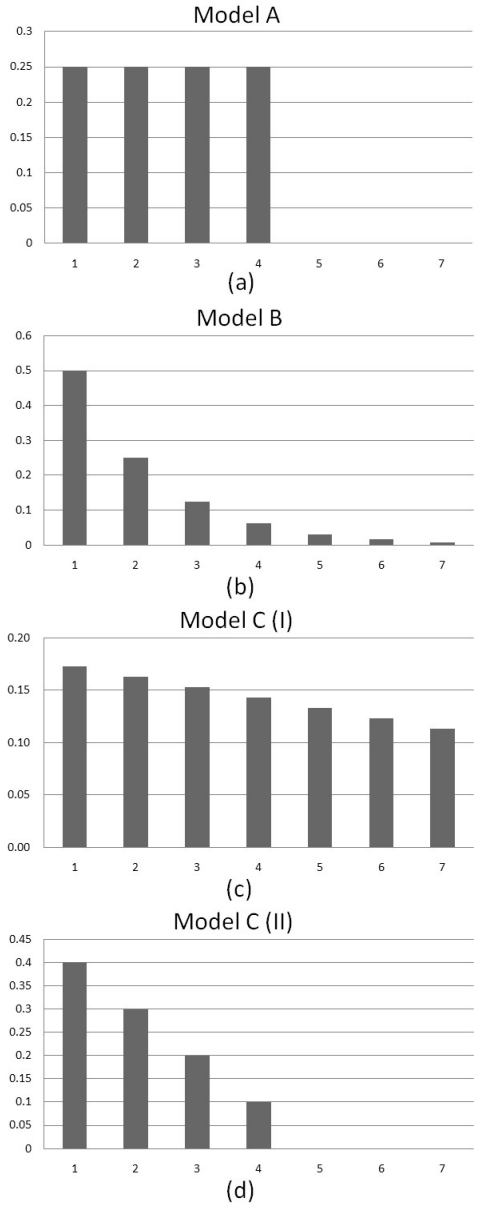

Fig. 7. Illustrations of the three standardized models.

As we can see, the entropy of Model B is the function of parameters r and n, the entropy of Model C is the function of parameters k and n, while the entropy of Model A is only the function of $\hat{n}$. We can model every non-increasing probability distribution function by the three standardized models with equal entropy based on this observation. Therefore, we can establish an approximate relationship between the coverage rate and the number of selected elements.

The threshold point k for an arbitrary non-increasing probability function is defined by the following equation:

$$k = \underset{i \in \mathrm{N}}{\operatorname{argmin}} \left\{ \sum_{x=1}^{i} P(x) \geq \alpha \right\} \tag{11}$$

where α is the coverage rate between zero and one.

We can solve k_2 for Model B by

$$\begin{aligned} \sum_{x=1}^{k_2} P_2(x) = a \sum_{x=1}^{k_2} r^{x-1} = \frac{1-r^{k_2}}{1-r^n} \geq \alpha \\ \Rightarrow k_2 = \lceil \log_r \{1-\alpha(1-r^n)\} \rceil \end{aligned} \tag{12}$$

We can calculate k_1 for Model A as well:

$$\sum_{x=1}^{k_1} P_1(x) = \sum_{x=1}^{k_1} \frac{1}{\hat{n}} = \frac{k_1}{\hat{n}} \geq \alpha \Rightarrow k_1 = \lceil \hat{n}\alpha \rceil \tag{13}$$

The threshold point k_3 for Model C is solved by geometry similarity of right triangle and trapezoid:

$$k_3 = \begin{cases} \left\lceil \frac{\frac{2}{n} + nk - \sqrt{\frac{2}{n} + nk^2 - 8\alpha \cdot k}}{2 \cdot k} \right\rceil & \text{for } 0 \leq k < \frac{2}{n^2} \\ \left\lceil (1 - \sqrt{1-\alpha}) \cdot \sqrt{2/k} \right\rceil & \text{for } k \geq \frac{2}{n^2} \end{cases} \tag{14}$$

Then we expect for an arbitrary non-increasing probability distribution function $f(x)$, the threshold point $k_{f(x)}$ would be close to k_1, k_2, and k_3, which come from the standardized models with the same entropy and total points as of $f(x)$. We verify it by two test sets; each includes 99 randomly generated non-increasing probability distributions, whose probabilities are assigned by uniform and Gaussian distributions.

Fig. 8 shows the simulation results of the two test sets. The differences of the 90% threshold points among the three standardized models and the real distributions generated by uniform distribution are shown in Fig. 8(a). As we can see, k_2's are closest to the actual threshold points while k_1's are also very accurate. The simulation result of the Gaussian counterpart is shown in Fig. 8(b). As we can see, k_1 and k_2 are good references for the actual threshold point.

The errors in coverage rate of the uniform and Gaussian test sets are shown in Fig. 9(a) and (b), respectively. As we can see, the coverage rate's errors are acceptable for each standardized model in both test sets, i.e., the idea that using these standardized models with the same entropy to estimate the threshold point of an unknown probability distribution, even before they are sorted, is sustainable.

Since the conversion from the entropy to the threshold point based on Model A has the lowest computational complexity among the three standardized models and also provides pretty good accuracy, we believe that Model A has higher potential to be applied to the proposed algorithm.

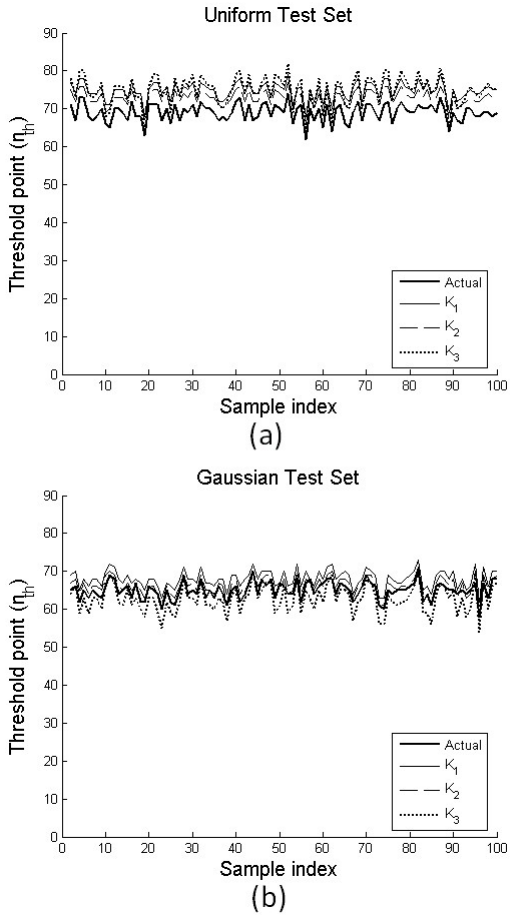

Fig. 8. Differences between estimated and real threshold points.

In the real world, we expect the LUT's separation to be efficient, i.e., the fast LUT saves much in capacity without losing too much information. If we found the estimated threshold point is close to $0.9n$, i.e., this probability model is very uniform, we can conclude that applying separation for this probability model is inefficient thus we keep all the data in the fast portion.

Since we cannot get exact partitioning until completing the first pass, further adjusting is required. Since our Markov model is a tree-like structure, it naturally provides multiple granularities in probability adjustment. For example, if we observe that the coverage rate of the fast portion is far less than the expected value after the first pass, we can relax the criteria for the root levels. On the other hand, if we observe that the coverage rate of the fast portion exceeds the capacity a little bit, we can tighten the criteria for the leaf levels. By controlling the criteria for each level, we can achieve successive adjusting of the coverage rate to full utilize the fast portion capacity.

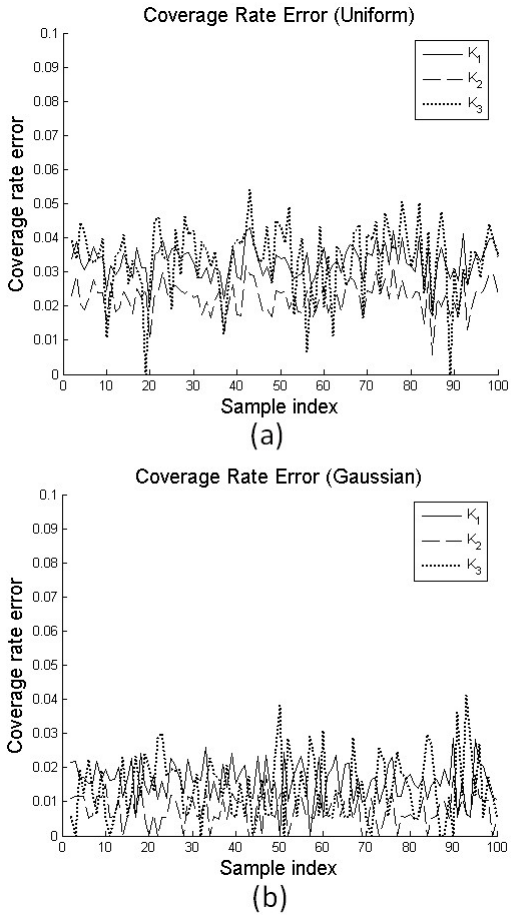

Fig. 9. Coverage rate errors generated by estimated threshold points.

VIII. Contributions

We developed a novel architecture for virtual computing in this paper. The proposed architecture combines the merits of the known methods: (1) By storing the software applications and the general operating system components at the server, it reduces the complexity arising from the incompatibility and inconsistency in the hardware and software. (2) By storing the personal data and files at the client machines, it allows the users to maintain the control of own data and files, and ensure the security and privacy. (3) By running the software applications on the client machines locally, optimal performance results. (4) By prioritizing the download sequence of the VM pages based on users' usage profile, the start-up delay is minimized and the probability of running out of pages during run time is minimized. (5) By introducing the LUT reduction and separation algorithms, the difficulties on implementation can be significantly alleviated.

IX. Future Works

In this paper, we address the performance of network-based virtual computing using the probability of code block usage. While the proposed approach improves the start-up delay and the performance, we have not addressed the issue of controlling the code block sequence, when the user behaviors change in real time generating an unacceptable rate of misses. In the future, we will investigate the use of information associated with the misses to adapt the code block sequence in real time.

Even with the methods of reducing the complexity in the LUTs, the entries in the LUTs remain substantial. We will investigate ways to reduce the complexity further. For example, in addition to the two-level hierarchy we described, we will extend our study to multiple levels of hierarchy.

In addressing the performance, the granularity of code block size is an important issue. We will investigate this issue in the context of performance control.

Copyright and privacy protections are significant issues in today's digital information distribution. Imposing access control on certain code blocks can be effectively used to prevent unauthorized accesses of some code blocks. We will investigate this issue as part of the code block download control.

There are numerous application opportunities that can benefit from virtual computing. We will investigate one or more applications to adopt virtual computing. This may involve some level of physical prototyping.

References

[1] L. P. Deutch and B. W. Lampson, SDS 930 Time-sharing System Preliminary Reference Manual, Doc. 30.10.10, Project Genie, Univ. Cal. at Berkeley, April 1965.

[2] Michael Price, *The Paradox of Security in Virtual Environments*, Computer Magazine, Vol. 41, Issue 11, Nov. 2008, pp. 22-28.

[3] Joeng Kim; Baratto, R.A.; Nieh, J., *An Application Streaming Service for Mobile Handheld Devices*, 2006. SCC '06. IEEE International Conference on Services Computing, Sept. 2006, pp. 323-326.

[4] Philip Winslow et al., *Desktop Virtualization Comes Of Age*, Credit Suisse, 2007-11-26.

[5] Rosenblum, M.; Garfinkel, T., *Virtual Machine Monitors: Current Technology and Future Trends*, Computer Magazine, Volume 38, Issue 5, May 2005, pp. 39-47.

[6] Uhlig, R. et al., *Intel Virtualization Technology*, Computer Magazine, Volume 38, Issue 5, May 2005, pp. 48-56.

[7] Vaughan-Nichols, S.J., *New Approach to Virtualization Is a Lightweight*, Computer Magazine, Volume 39, Issue 11, November 2006, pp. 12-14.

[8] *VMware ThinApp Agentless Application Virtualization Overview.*

[9] EMA Report: *AppStream: Transforming On-Premise Software for SaaS Delivery - without Reengineering*

[10] Chen, P.M. and Noble, B.D., *When Virtual Is Better Than Real*, Proceedings 8th Workshop Hot Topics in Operating Systems, IEEE CS Press, 20-22 May 2001, pp. 133-138.

[11] Sunwook Kim et al., *On-demand Software Streaming System for Embedded System*, WiCOM 2006 International Conference on Wireless Communications, Networking and Mobile Computing, 22-24 Sept. 2006, pp. 1-4.

[12] Ana Fernandez Vilas et al., *Providing Web Services over DVB-H: Mobile Web Services*, IEEE Transactions on Consumer Electronics, Vol. 53, No, 2, May 2007, pp. 644-652.

[13] Godmar Back and Wilson C. Hsieh, *The KaffeOS Java Runtime System*, ACM Transactions on Programming Languages and Systems (TOPLAS), Volume 27, Issue 4, July 2005, pp. 583-630.