CSC548, Fall 2006
Final Project Report

Name: Arun Babu Nagarajan
Project: Enhanced Proactive Fault tolerance system for HPC with Xen virtualization
Website: http://www4.ncsu.edu/~abnagara/csc548_project

Problem Statement:

To improve the currently existing fault tolerance system by cutting down the cost to migrate a system image by pre-deploying portions of the system image.

Problem Description:

The currently implemented FT system necessitates the health monitoring system to figure out node failure before 13 – 40 seconds for live migration and 10-13 seconds for stop/copy migration.(based on the application) so that the VM can be safely migrated to the destination. The way the migration happens is that during the initial iteration, the pages are sent over to the target. The next iteration sends the pages, which have been dirtied since the previous send, and so on. It has been observed with the NAS parallel benchmarks that a large chunk of the pages (in fact more than 90%) are sent during the initial iteration and the other pages are sent repeatedly during the following iterations, (depending on the working set at the time the migration command was initiated).

We would like to exploit this behavior by sending some part of the VM image earlier than required to the spare node so that we could significantly cut down on the transfer cost.

Motivation:

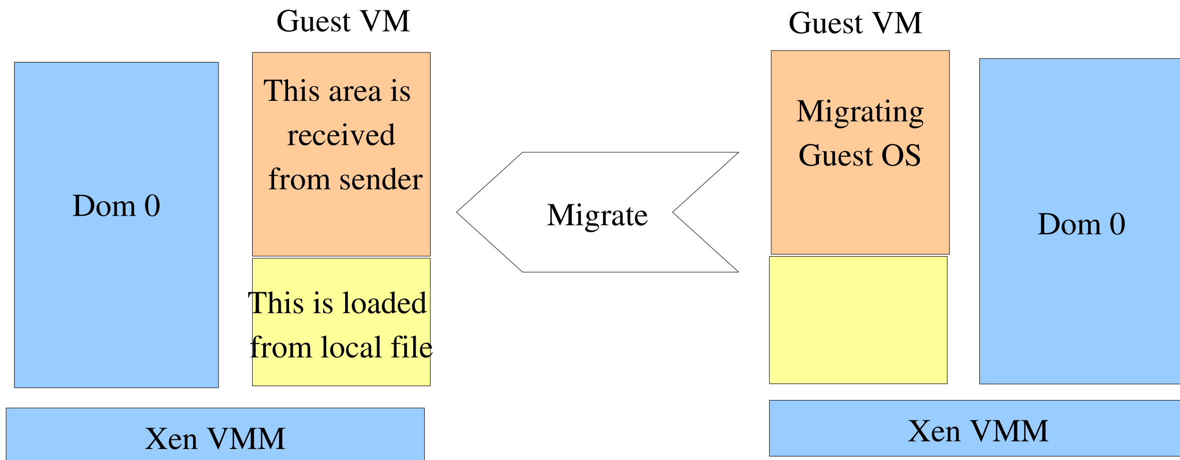
A study to identify the memory usage associated with the NPB codes is shown in the table.

	Class B		Class C	
	RSS (MB)	Vsize (MB)	RSS (MB)	Vsize(MB)
BT	107.90	119.41	406.01	423.33
CG	110.45	125.85	282.09	306.23
EP	2.26	10.61	2.28	10.61
LU	59.14	50.30	183.06	193.60
SP	95.55	111.88	351.39	379.35

Here RSS is the Resident Set Size which gives the memory usage of the process + shared pages. Vsize represents the total address space of the process. Also it is observed that, with the current configuration, the memory occupied before any process is running on the guest VM is around 250-300 MB. So there is scope for pre-deployment.

Project Description:

The outline of the work is to identify the pages that can be pre-deployed and make them available at a destination. So when a guest VM needs to be migrated not all the pages of the VM needs to be sent to the destination. The pages identified earlier as pre-deployable can be loaded from the local machine. The constraints are discussed in the following sections.



In order to implement this, we need some off-line tools which help in analysis phase for pre-deployment. Initially we need to identify the candidate pages to be made available at the destination. After identifying the candidate pages, we need to extract them from the live OS image and write them to a file so that the file can be made available at the destination. Utilities which accomplish these functionality are described in sections A and B.

The migration infrastructure needs modifications on both sending and receiving ends. The sender needs to take care of sending only the pages which are to be sent. The receiver will have to receive the pages that are sent by the sender and also will have to load the missing pieces from the pre-deploy file available locally. This is discussed in section C.

A. Compare utility:

The aim is to find the set of pages which can be predeployed to the destination before the actual migration. The constraints are

1. The pages should take the same position in different virtual machine images(a same VM kernel version is assumed)
2. These pages are to be rarely written

For handling the first constraint the 'prelink' option on the virtual machine needs to be turned off. This helps in eliminating the random placement of pages. The second constraint helps in improving the

overall performance of the system. Since we are identifying the pages which are to be present in the destination machine, preferably the pages should be such that they are not dirtied by the virtual machine. But this is not a strong constraint in the sense that even if the pages are dirtied, the system needs to get the latest page from the live VM.

This functionality is added as a new tool to the set of currently existing tools. For coding this tool the functionality of the save/restore tools and structure of the save file was to be analyzed. The save file structure/ flow is presented in Appendix A.

The compare utility is embedded in the existing xm <..> set of tools and it can be invoked with
xm compare <img1> <img2>

B. Create Pre-deploy file

After identifying the set of pages to be predeployed those pages are to be written to a file so that it can be made available at the destination. Currently the save infrastructure is modified to generate this file from a live VM image. (Should be improved to allow accepting the set of pages to be written to the file)

C. Modifications to migration infrastructure

The way migration works currently is that the migration command is initiated from the privileged domain to migrate a guest domain. The privileged domain activates shadow paging and starts sending the pages to the destination iteratively. For live migration, the pages that were dirtied after being sent can be identified by the help of shadow paging technique. Now the dirtied pages are resent and finally the VM is stopped and the pages are copied to the other side.

We need to modify the above described infrastructure such that the sending and receiving sides take care of the new features.

Sending side:

The sending side executes the logic for saving the pages to disk/ sending them to the destination. While sending the pages, currently the system sends all the pages. The functionality is explained briefly here. Processing happens in a batch of pages. The batch size is 1024 pages.

i) A batch of page is read from the guest VM (with write protection) and their page type is also obtained from the hypervisor.
ii) These page types reflect the machine frame number and they are canonicalized to page frame numbers.
iii) Write/send the above page type array
iv) After this, the pages in a batch are walked through and they are sent directly or send after canonicalization(if it is a page table page as identified from the page type array)

Suppose we have the list of pages which are not to be sent, at step (ii) we can mark them by adding a new page type for depicting this. So during step (iv) we can make sure that if we encounter a page of above type we simply skip that page from sending. Here we need to be careful in that, we need to double check if the page we are skipping is not a page table page. (A page can be identified as a page table page from the array obtained from step(ii)).

Receiving side:

Like the sending side, the processing happens in a similar way on the receiving side. The receiving side reads the pages in batches.

i) Read the page type array sent by step ii of sending side
ii) Receive the pages one by one from the sender
iii) Walk throthe page type array and receive the pages of the batch one by one and in case it is a page table page, canonicalize it

So the modification is done is step iii. When we encounter a page type as the new page type (which we have introduced in sender side) instead of receiving the page from the sender, it is received/read from the local file.

Handling dirtying of the pre-deploy pages:

The assumption in above discussion is that the pages which are pre-deployed are not modified in the guest OS. But it might happen that it can be modified. In order to handle this case, shadowing on the pages which are marked pre-deployable needs to be done. The sender will have to maintain the shadowing facility and if it detects that the page has been changed, then it will have to send this dirtied page to the sender.

Issues to be addressed:

i) One issue which was not addressed is handling the free pages. At any given point of time, the Os will have a set of free pages, which are not used by any processes. So there is no need to transfer them to the destination while migration. But the problem here is that we will have to break some abstraction and investigate about the free list of the guest OS.
ii) As discussed above the handling of dirtying of the pre-deployed pages need to be considered.
iii) Identifying a good set of pages for pre-deployment. The compare tool is in place and it needs to be run over different images and data needs to be collected. But there is no guarantee yet that the pages identified are not written to by the guest often. Some mechanism needs to be added so that we can identify such pages.

Results:

Image compare:

Images of Ms are obtained by using the xm save tool and compared using the compare tool described above. This is done for different VM sizes.

VM size(MB)	# of pages	# matches	% matches
256	67584	2208	3.27
512	133120	58918	44.26
1024	264192	188564	71.37

We see that increase in OS image leads to increase in # of matches. (A bulk of them must be due to the free pages). Yet to obtain a good set of pages to be pre-deployed.

Migration with pre-deploy

Currently for proof of concept, a range of page numbers are assumed to be pre-deployed and the pre-deployment file is created from the save tool discussed earlier. This file is made available at a destination. On initiating a migration, the sender does not send these pages and the destination loads these pages from the file. The infrastructure changes discussed above are in place except for the handling dirtying of these pages.

Appendix A:

The structure of the save file is depicted below: