Zap & VMMigration: Zap and Virtual Machine Process Migration

Todd Tannenbaum
CS 739: Distributed Systems
University of Wisconsin, Madison

Feb 3
Spring 2006

1 Overview

In class we discussed the paper "The Design and Implementation of Zap: A system for Migrating Computing Environments" (Zap) by Osman et all at Columbia, as well as "Live Mirgration of Virtual Machines" (VMMigration) by Clark et all at Cambridge and Copenhagen. We had guest lecturers for the class, Greg Q and Joe M. In addition to discussion of the two different approaches to process migration presented in these papers, we compared them to the approach taken in the Sprite system.

2 Problem Statement and Assumptions

Zap presents a system for process migration that strives to meet the following goals:

1. Migrate legacy applications, without modification or limitations on use of common operating system services.
2. Leverage existing commodity operating systems.
3. Avoid creating residual dependencies.
4. Overhead should be small for both normal execution and migration.

Zap assumes an environment consisting of machines connected by a network, and that commodity operating systems offer loadable kernel modules. Very little is explicitly mentioned about the anticipated workload other than legacy unmodified networked applications are expected.

Because Zap requires a process that could migrate to be initially created in a pod, Zap assumes that the user knows in advance which processes should be candidates for migration. This contrasts with Sprite, in which any process on the machine can be told to migrate.

Zap also assumes the presence of network file servers that store applications and user data, as well as dynamic DNS servers to support network address virtualization. Finally, the authors assume that each machine is independent of the others, each containing its own independent operating system. This is in stark contrast to Sprite.

In the VMMigration paper, the goal is to consider the design options for migration of active operating systems hosting live services. Therefore, much attention is paid to minimize the downtime during migration, to minimize resource contention between the service and the migration mechanism, and other factors that could degrade the quality of the service from the perspective of the client. The implementation does not address migration in a wide-area network. *Essentially, the assumed environment is a cluster in the data center charged with providing interactive network-based services. All machines are further assumed to be on a local switch, and all data accessed via locally attached storage.*

3 Design and Implementation

3.1 Architecture

Fundamental to Zap's design is the introduction of the pod abstraction. A pod provides a collection of processes with a host-independent virtualized view of the operating system. Pods are self-contained units that can be suspended to secondary storage, migrated to another machine, and transparently resumed. Pods have their own private, virtual namespace. This namespace provides consistent, virtual resource names in place of host-dependent resource names such as PIDs. Processes within the pod are able to

interact with each other as usual, but processes outside of the pod do not appear in the namespace and therefore cannot interact with processes inside a pod using IPC mechanisms such as shared memory and signals.

In the VMMigration, the mechanisms presented by the Xen virtual machine monitor are leveraged to migrate operating system instances across physical hosts. *In other words, Xen provides much of the transparency and checkpoint functionality that the Sprite and Zap developers had to develop themselves*. The design focuses on minimization of downtime (during which services are entirely unavailable), and on total migration time (during which the state on both machines is synchronized and hence may affect reliability). *Because the "heavy lifting" is performed by Xen itself, and because Xen needs to move so much state, the focus of this paper is really on performance*.

The core of the VMMigration design introduces a pre-copy approach, in which pages of memory are iteratively copied from the source to the destination host, all without ever stopping the execution of the virtual machine being migrated.

This design highlights an important distinction: migration time -vs- total migration time.

So which design allows for memory to be moved faster? In both Sprite and Zap, all migration process memory state is written to disk as a checkpoint and then pulled, while in VMMigration a pre-copy mechanism that is a combination of push and pull is used. After discussion, we felt that total migration time in a push system wold almost always be better.

But comparing VMMigration to Sprite/Zap is difficult because VMMigration is moving an entire machine's state instead of just a few processes. A fundamental question to ask when comparing these works is can approaches such as pre-copy overcome the fact that so much more state is being moved?

3.2 Migration

Zap performs migration by checkpointing the memory areas allocated by processes within the pod. *This sounds difficult to port and maintain*. For file state, Zap provides each pod with its own virtualized file system and corresponding private file system namespace while leveraging file systems such as NFS to store file data. This approach takes advantage of distributed file systems to reduce file state that would need to be moved during migration without requiring a global file system across all host machines. *But since Zap does not provide a mechanism to actually move file data with the checkpoint, it actually does require a file system across all host machines as best as we can tell. The file namespace virtualization just allows the local system to mount the global filesystem upon an arbitrary local mount point. However, at least any network file system may be used with Zap — in the case of Sprite, the migration mechanism required the use of Sprite's network file service.*

Sprite, by adding residual dependencies, had better support for access to local devices after migration. Although Zap supports virtualization of the device namespace to support devices such as pseudo terminals, neither Zap nor VMMigration worried much about access to physical devices such as CD-ROM drives after migration.

In VMMigration, migration is performed at the level of the virtual machine. Xen provides most of the mechanism to checkpoint, restart, and therefore migrate most state including process, kernel, and machine state. *Because Xen does much of the "heavy lifting" with respect to migration, the VMMigration authors focused primarily on performance, but also network and file state migration.*

In class, we talked about the nuances of migration performance. We discussed the idea of "migration downtime" -vs- "total migration time". Migration downtime is the amount of time a service is not executing while in the process of being migrated. Total migration time is the amount of time it takes to complete the migration process from source to destination machines and ideally remove any residual dependencies from the source machine. In Zap, the migration downtime and the total migration time are equal. In VMMigration, due the pre-copy approach, the migration downtime is minimized and is much smaller than the total migration time.

As previously mentioned, the VMMigration approach must move a lot more memory state than either Zap or Sprite. The authors explored several memory migration techniques such as

Push The source keeps running while pages are pushed to the destination. Pages modified during this process must be pushed again to ensure consistency.

Stop-and-copy The source is stopped, pages are copied to the destination, and then execution resumes at the destination. *This is the simple approach used by Zap. Complexity is minimized at the cost of increased migration downtime. Also similar to Sprite, which wrote the entire memory image to disk at the source and then read it at the destination.*

Pull Execution is started at the destination, and if a page is accessed that has not yet been copied, a fault occurs and then the page is pulled from the source. *Decreases migration downtime at the cost of total migration time.*

The authors developed a technique they called pre-copy, which consists of a bounded iterative push phase followed by a relatively small stop-n-copy. The pre-copy technique works best when memory pages can be copied to the destination host faster than they are dirtied by the migrating virtual machine. If this is not happening, the VMMigration authors propose a "process stunning" technique that limits each process to 40 write faults before being moved to the wait queue. *We found process stunning to be a very interesting idea - essentially, it gives a tuneable knob to allow you to negatively impact the performance of the running service in order to improve migration time.*

For network resources, the VMMigration authors observed that in a cluster environment, the interfaces of the source and destination typically exist on a single switched LAN. Therefore, their solution was to generate an unsolicited ARP reply from the migrated host to advertise the IP address of the new location. *This solution limits deployment to data-center type environments, and would not be suitable to a departmental or organizational-wide deployment. On the other hand, it does remove the need for a residual dependency on the originating machine.*

For storage, VMMigration relies upon network-attached storage *(as does Sprite and Zap).*

3.3 Transparency

A fundamental transparency issue is with both Zap and VMMigration, you need to know in advance which processes could be candidates for migration, since these processes must be started in either a pod or virtual machine respectively. This is not the case with Sprite.

We discussed the internal -vs- external transparency. Internal is the viewpoint from the processes being migrated; external is the viewpoint of processes not being moved. For example, Sprite has better external transparency than Zap because you can run /bin/ps on the source machine and get a consistent view before and after migration. We felt that good external transparency usually comes at the cost of increased residual dependencies.

As for network operation transparency, Sprite keeps network operations very transparent at the cost of a residual dependency for the entire process lifetime. Zap has a residual dependency only for a finite X number of minutes, but has a major transparency issue in that BOTH the client and server side of the connection need to be in pods in order for their Virtual Network Address Translation (VNAT) mechanism to work. VMMigration places no such restrictions on the client side, but instead requires the server side to migrate only between machines that are on the same subnet (so ARP broadcasts work) as previously mentioned above.

3.4 Implementation

In VMMigration, the implementation is integrated with the Xen virtual machine monitor. Xen provides one special management virtual machine used for the administration and control of the machine. *This design lends itself well to managed migration approaches.*

Zap virtualization is largely implemented by providing a mechanism for intercepting system calls to translate between pod and operating system namespaces. Zap migration mechanism is based upon CRAK, a Linux kernel module written earlier by one of the Zap authors. Linux netfilter is used for network virtualization.

We felt that the Zap implementation was more complex and fragile than the VMMigration implementation, largely because Xen shielded VMMigration with a clean level of indirection from having to deal with kernel state. It seems to us that CRAK would be very sensitive to kernel changes, similar to how the Sprite migration mechanism constantly failed in regression testing as the kernel changed. Furthermore, the Zap authors have never made Zap publically available in either source or binary form (perhaps because it is embarrassingly fragile?), and constantly referred to their implementation as "proof-of-concept". This makes us speculate that an actual production-quality Zap implementation would be very complex and difficult to get working correctly.

4 Evaluation

The Zap authors provide benchmarks of several simple synthetic applications as well as Apache web server and VNC virtual X11 server. *Similar to the benchmarking done by the Sprite authors; not necessarily indicative of a real-world work load.* Since VMMs have been proposed for migration, Zap authors also performed experiments with VMWare Workstation for Linux. Measurements were made of execution times inside a pod -vs- inside a VMWare virtual machine, as well as times for VMWare to checkpoint/restart -vs- Zap to migrate.

The Zap authors state that the performance overhead for a process running inside a Zap pod is much smaller than running in a VMware virtual machine. *However, we would have loved to see performance numbers -vs- Xen in addition to VMWare. We suspect that the performance overhead of a visor such as Xen is much lower than VMWare, and it would have made performance comparisons between Zap and VMMigration easier.* As for mi-

gration, the Zap authors found that it was generally bound by the pod image size, and was faster than a VMware checkpoint/restart in all instances because Zap saves less state. *But unlike the VMMigration and Sprite authors, the Zap authors did not consider techniques to overlap migration with continued program execution.*

5 Policy and Usage

Sprite performed some trivial, yet apparently effective, policies to decide where to migrate a process and the Sprite authors did some work to investigate policy. For instance, Sprite would select a candidate resource based upon how long it has been unused. Zap and VMMigration are purely mechanism and do not address the questions of which, where, or when to migrate.

6 Conclusions

The class felt the fundamental question that arises from these papers is do you really need an entire virtual machine to migrate just a process or two? We felt the answer was no; however, using a virtual machine has advantages beyond migration such as better sandboxing/security isolation and running other operating systems on the same hardware.

This discussion lead into talking about direct migration (like VMMigration) -vs- migration via checkpoint and restart (like Sprite and ZAP). While direct migration may have a performance advantage, it only helps with issues of availability. We felt that checkpoint/restart was advantageous because it provides enables additional functionality "for free", such as fault tolerance, migration to storage until a processor is available (queuing), and multiple "snapshots" through time (useful for incremental debugging of long running services and other applications).