CMPSCI 677 Operating Systems **Spring 2014**

Lecture 6: February 10

Lecturer: Prashant Shenoy *Scribe: Aditya Sundarrajan*

6.1 Virtualization continued

6.1.1 Case Study: Planet Lab

Planet Lab is a distributed set of machines (contributed by many different universities) that are used for research. It is used mainly for networking/distributed systems research. If a student doing research in, say, networking needs a distributed set of machines at 'n' different locations, they can get them from Planet Lab. Planet Lab makes this possible via virtualization. They have many physical machines physically sitting at several different locations. They create a VM at 'n' of those locations and give the student access to them. Where as most others have used hardware level virtualization, Planet Lab uses OS level virtualization. They assign a slice of the machine to a user by giving them a container (like the Solaris container or BSD jails that were described earlier). Therefore, the user does not get a VM running another copy of the OS, but they get a container with the allocated amount of resources (including memory, CPU etc.). This enables users to run experiments without impacting other users. This could have been implemented using hardware virtualization as well, however, OS level virtualization makes it more efficient as another copy of the kernel does not need to be there for container.

6.2 Code and Process Migration

As part of distributed scheduling, code/processes may need to be migrated from one machine to the other. In some cases, entire VMs may need to be migrated. Both these cases will be discussed.

6.2.1 Motivation

- Performance and flexibility are two important reasons why code/process migration is done. By moving code/processes, the load on a system can be redistributed for better performance. The program can be executed on any machine, which offers the flexibility of not being tied to just one machine.
- Process Migration (aka *strong mobility*): In this case, a process that is running is moved to another machine. This is more complicated than code migration, as every process has state (like an address space and resources) that needs to be transferred, and then it needs to be resumed from where it was suspended.
- Code Migration (aka *weak mobility*): In this case, code is shipped from one machine to another. Code migration can only be done before the process begins, once the code starts executing, it is essentially process migration. Code migration is somewhat simpler as only the code needs to be moved and started on the receiving machine. This is much more common than process migration. Here are some examples: 1) Filling a form on a browser which then sends a query that is executed in the server., 2) A

Java applet is another example of of code migration, where the client downloads the applet code from sever and executes it on the local machine.

- Flexibility: Many operating systems today let you plug in any device without having the driver software pre-installed. If the driver software is not present, the OS will offer to download it from a repository or from the internet. This is another example of code migration. Hence code migration offers more flexibility as you don't need to have every piece of software/driver pre-installed and can be downloaded on demand.

6.2.2 Migration Models

- Processes have an address space that consists of a code segment, a stack segment, and a heap segment. Apart from these, the processes could also have associated resources that it uses. If a process is migrated, then all its associated code and resources need to be transferred.
- Migrating code is simpler than migrating an entire precess. During weak mobility, the binary code or the script is transferred to the new system and is executed from the initial state.
- Migration can either be sender-initiated or receiver-initiated. There is a difference between whether the sender pushed the code (sender-initiated) or whether the receiver requested it (receiver-initiated). E.g. a query that is sent to the server is sender-initiated code migration. A Java applet is an example of receiver-initiated code migration in that the client can request it from the server which it then downloads to its own system.

6.2.3 Executing Migrated Entity

- Code migration : In code migration the entire code is transferred to the new system and is executed from the initial state.
- Process migration : In process migration, the existing process including the code and execution segments and the resources it is using, is moved to the new system from where it continues executing. This can be implemented int two ways.
  - Remote cloning : In this technique, the process is cloned to the new system. Once this is completed, the cloned process continues executing and the former process in the old system is stopped.
  - In the second technique, cloning is not involved. The actual process is moved to a new machine where it registers with the new system and it continues executing.

6.2.4 Models of Migration

As discussed previously, mobility is of two types, strong and weak mobility, and each of them can be sender and receiver initiated mobility, thus combinations of these lead to 8 different migration models.

6.2.5 Resource Migration

Migrating a process from one machine to another requires stopping the process at the point of execution on one machine and then starting from the same point in other machine. When process migration takes place then code segment, resource segment and execution segment needs to be migrated to the new machine. This can be a tedious task as many constraints are involved. For instance, if a process on the original system is accessing a file and the process is migrated, an error would be caused when the same file is accessed in the new system as that file doesn't exist. Before we look at how to handle these resources, we discuss different types of process to resource bindings.

- Binding by identifier : This is a hard binding (accessing local files on A), such kind of resources need to be moved to system B since the process will use them as is i.e use the same file name to access the open file in System B. Other examples include specific web pages that will need to accessed.
- Binding by value : Suppose the process was running in JVM 2.0 environment in system A, then when it migrates to system B, it will expect JVM 2.0 environment to be present to continue execution. Thus when process migrates to B, a reference to the JVM 2.0 present on B has to be provided to the process.
- Binding by type :This is the weakest form of binding. Suppose the process was accessing hardware devices such as a printer, then when it moves to machine B, this particular process can use the device that is attached to machine B.

The above bindings show that some resources allow more flexibility when a process is moved than other resources. Now we need to discuss if we can move the resources to different machines i.e. resource to machine binding.

- Unattached resources : These are generally data files that have been opened by the process for read / write operation. These resources can be moved easily and quickly to another machine but they need be referenced properly in the new system so that process can access the path of the files correctly.
- Fastened resources : These are generally local databases on machine A that are used by the process. Moving these resources to new machine B is possible but can be expensive. Thus, it is preferable to create global references for such resources that can be accessed over network.
- Fixed Resources : These are resources like printer or other devices which cant be moved. For these generally global referencing technique is used. For example, open socket connections can be accessed through tunneling.

Different combinations of the bindings produce different type of complexity and each has to be handled in a different way. For example, if we have a resource bound by the identifier like a url and it is unattached, then we would need to move that resource to the new machine. If however, the resource is fixed, then we would need to provide remote access to the old machine over a network. If no access is provided to the resource then the process will break.

6.2.5.1 Network Socket Migration

If a process has a socket open on the original machine A and that process is moved to machine B, then a tunnel needs to created that will be used forward all the packets received on the original socket in machine A from the source to machine B. Though complicated, this is the only way the connection remains active since the process is not aware of the migration. If the process is aware, then it could communicate the new port and IP information with the source.

6.2.6 Migration in Heterogeneous Systems

When a process is moved in a heterogeneous environment, i.e. between machines using different architectures (Intel / ARM), this leads to a real challenge since the binaries, assembly code instructions are all different for different architectures. The code will need to be recompiled according to new architecture and care has to be taken about little Endian and big Endian formats etc. One way to achieve migration on heterogeneous system is binary translation of code on the fly, i.e. each instruction received is converted to new machine-readable instruction. However, this is not the efficient way to do migration. These issue can be resolved using virtual machines or for instance programs running on JVM since it is platform independent. Other instances include migrating scripts such as python programs. As long as a python interpreter is present in the new machine, the code can be executed.

6.2.7 Migration in Today's Systems

- Web : In the web, code migration is very prevalent. Many HTML pages have javascript code embedded in it that is downloaded and executed in the browser.
- Batch schedulers : Batch jobs such as large scale simulations or other computations, that are sent to compute clusters are examples of weak mobility.
- Virtual machine migration : In this case, the entire OS and all the associated processes are moved to a new machine. This is discussed in the following section.
- Malware : This is another example of code mobility where malicious/harmful code spreads across systems affecting its operation.

6.2.8 Virtual Machine Migration

VMs are migrated when the server it is on cannot handle more capacity. One way to do this is to shut down the VM and copy the virtual disk from one machine to another where it can be restarted. There will be a downtime in this method of migration since the VM has to be suspended, moved and then resumed. This downtime can be averted using live migration. For live migration we can assume that the disk is shared between the two machines. This is the case in a LAN environment. However, the memory state is moved using iterative copying. In this technique, while the VM is running on machine A the memory state of the processes running in the VM is copied to the new machine B, and once the copy is completed, the VM in machine B can start executing after the VM in machine A is terminated. Thus user will not see any delay while switching. During live migration of process, the RAM pages are copied from old machine A to new machine B while process is still being executed in A. During the duration of copy, the process in machine A would have made some changes to the RAM pages in machine A. Thus to overcome this, iterative copying of the new pages is done until only a fraction of pages remain. The idea is that in each iteration, the number of pages transferred decreases since each consecutive iteration is shorter than the previous one. When this fraction is small enough, all the new pages are copied to the new machine, the original machine is paused and the new machine resumes from where the old machine paused.

Now suppose, the process in VM of machine A uses network connection to access data through an open port, then in this scenario, the switch can be informed to reroute the data to a new IP address of machine B. This won't create a problem because the process inside the VM was assigned a virtual IP address and it can remain same when moved to machine B, only the VM needs to connect to the physical address of new machine which is hidden from the process.

6.2.9 Case study : Viruses and Malware

Viruses and malware can be propagated using code mobility. It can be sender-initiated (proactive viruses go and affect other machines) or receiver-initiated (user clicks on malicious link and the virus code gets downloaded onto host machine).

6.3 Server Design Issues

6.3.1 Server Design

There are two types of server design choices:

- Iterative : It is a single threaded server which processes the requests in a queue. While processing the current request it adds incoming requests in the wait queue, and once the processing is done, it takes in the next request in the queue. Essentially, requests are not executed in parallel at anytime on the server.
- Concurrent or multi-threaded server : In this case when a new request comes in, the server spawns a new thread to service the request. Thus all processes are executed in parallel in this scenario. This is the thread-per-request model There is also one more flavor to the multi-threaded server. Unlike the previous case where new thread is spawned every time a new request comes in, there is a pool of pre-spawned threads in the server which are ready to serve the requests. A dispatcher or scheduler thread handles this pool of pre-spawned threads.

6.3.2 How to find servers?

The client can determine the server it need to communicates using one of the two techniques.

- Hard code the port number of the server in the client. If the server moves then the client will need to be recompiled.
- In the second technique, a directory service is used to register a service with a port number and IP address. The client then connects with the directory service to determine the location of the server with a particular service, and then sends it the request. In this case, the client only needs to know the location of the directory service.

6.3.3 Stateful or Stateless Servers

Stateful servers are those, which keep a state of their connected clients. For example, push servers are stateful servers since the server need to know the list of clients it needs to send messages to.
Stateless servers on the other hand don't keep any information about the connected clients. In this case, the client will keep its own session information. Pull servers are examples of stateless servers where the clients send requests to the servers as and when required.
Soft state servers maintain the state of the client for a limited period of time on the server and when the session timeouts it discards the information.

6.3.4 Server Clusters

In a cluster, the servers are segregated according to their functionality into tiers and each tier is replicated so as to enhance serviceability. For example, the database tier of the system can be replicated and distributed on several different machines, thus when a request comes in, the dispatcher needs to identify which machine should the request be directed to. These dispatchers are typically called load balancers. It keeps track of load on each server in the cluster and is also responsible for maintaining session details of clients, so that it can send the request to the correct server upon concurrent requests by the same client. Maintaining consistency among replicated data is important to ensure correct operation. Once the dispatcher gets the request from the client, it uses the following techniques to transfer the request to the correct sever.

- TCP Splicing The client connects to the dispatcher through TCP connection but there is no direct connection between the client and server. The dispatcher sets up a separate connection between the server and itself and then splices the client connection and server connection together so that data transmission is possible between server and client.
- TCP Handoff The dispatcher hands off the request from the client directly to the server at TCP Level or using HTTP redirect.

6.3.5 Server architecture

The different types of server architectures are:

- Pure sequential : In this architecture, a single-threaded process is used to serve requests one at a time and no concurrency is achieved.
- Concurrent : This could be of two kinds.
  - Multi-threaded server : In this design, a new thread is spawned each time a request is received. This is the thread-per-request mode. A pre-spawned pool of threads could also be used to serve incoming requests.
  - Multi-process server : Similar to the multi-threaded server design, a process-per-request model or a pre-spawned pool pf processes could be used to serve incoming requests.
- Event-based server : In this technique, asynchronous non-blocking calls are used in a single-threaded server. For example when asynchronous I/O is used, file reads which are usually blocking calls are no longer blocked, and are instead handed over to the OS. In the meantime, the next request in the queue could be processed. When the I/O operation is complete, an interrupt is received and the original request resumes execution. This way, we can use have a single thread of execution but still achieve concurrency since non-blocking calls are used.

6.3.5.1 Which architecture is most efficient?

In the uni-core environment, Event-based implementation > thread-based implementation > process-based implementation > pure sequential based implementation.

Event-based implementation though sequential offers concurrency because asynchronous non-blocking calls are used. Hence when a request being executed issues a non-blocking I/O call, it is passed onto the OS, and the next request in the queue is executed until the response for the I/O is received via an interrupt. Moreover the event-based implementation is a sequence of function calls rather than context-switches (among threads or processes) that are present in both the thread-based or process-based implementations making it an efficient design. However, it is more difficult to develop event-based programs when compared to multi-threaded or multi-process programs. Threads are more light-weight to create than processes. Hence thread-based implementation is more efficient than process-based implementation. In general, concurrent implementation is more efficient than pure sequential implementation since more requests could be served at once in concurrent servers. In sequential servers, only one request can be served at a time.

However, in the multi-core environment, event-based servers will not achieve true parallelism since it has only one thread of execution. In this scenario, a thread-based implementation would perform better as it could run on multiple cores simultaneously.