CMPSCI 677 Operating Systems **Spring 2023**

Lecture 10: March 08

Lecturer: Prashant Shenoy Scribe: Diptyaroop Maji (2023), Samriddhi Raj (2022), Xingda Chen (2019)

10.1 Proportional Share Scheduling

This mechanism is widely used in virtualization to allocate resources to individual virtual machines (Type 1 and Type 2 Hypervisors) and containers. It is used to decide how much CPU weightage to allocate to each container and network bandwidth to each container. In share-based scheduling, a weight is allocated to each container and CPU time is divided in proportion to this weight. So if two containers have 1 and 2 as weights they will receive 1/3 and 2/3 of the CPU time. However, if the container is idle and is not using the resource, its share is redistributed across other containers/processes that have something to run in proportion to their weights (fair share scheduling).

Lottery-based scheduling: Fair share scheduling in a randomized way. We assign each container some number of lottery tickets. The CPU scheduler decides which process/ container gets the next time slot via a lottery – it's going to take all the tickets that have been assigned to all the containers and pick one winning ticket randomly. The container that holds the winning ticket gets to run next next time slice. The number of lottery tickets a container has determines the chance of winning the lottery. So, by controlling the number of lottery tickets, we can decide how much share of a resource a container will get.

Hard limits: Hard limit means there is a hard allocation. Even if the container is not using the resources, it is not redistributed across other containers. Those cycles are simply wasted.

10.1.1 Weighted Fair Queuing (WFQ)

Each container is assigned a weight w_i and it receives $w_i / \sum_j w_j$ fraction of CPU time. The scheduler keeps a counter for each container, s_i, which tracks how much CPU time each counter has received so far. The scheduler picks the container with minimum count so far and allocates a quantum time unit q. The counter value for the selected process is updated, $s_i = s_i + \frac{q}{w_i}$.

If one container is blocked on I/O and not using CPU, its counter value does not increase. However another container keeps on running and its counter value is incrementing. When the first container finishes I/O, it will have a low counter value as compared to second counter. Thus it will be scheduled for a long time while the second container is starved. This does not follow fair based scheduling. To avoid this the counter value is updated using this formula: $s_{min} = \min(s_1, s_2, \ldots)$ and $s_i = \max(s_{min}, s_i + \frac{q}{w_i})$.

Question: How does the scheduler know whether the container has used CPU cycles?

Answer: Let's only consider processes first. Each process gets a weight, and the share-based scheduler schedules these processes. These processes get CPU share in proportion to their weights. The CPU scheduler knows which processes are blocked on I/O and which processes are ready to run. If a process is blocked on I/O then it is not running and basically giving up its share. If some process is runnable, that means it is active. It will get to run and its counter will be incremented. It's the same concept for the container. There are processes in the container, so if any process from that container runs that container's counter is incremented based on the cycles used by that container. If there's nothing running, then the counter doesn't

increment and something else gets to run.

Question: Is the number of processes static or dynamic?

Answer: The number of processes/containers are assumed to be dynamic. It will change over time. It means that the relative share of the resource you get will change over time. Say a container is given a weight of one. If there's nothing else in the system, it means that the container gets a hundred percent of the resource. If another container gets created and it also gets a weight of one, now the resource is shared 1:1. If a third container comes now again the allocated fractions will change.

Question: Why is $s_i = \max(s_{min}, s_i + \frac{q}{w_i})$?

Answer: This says that if a process P becomes inactive and then comes back and becomes active, its counter has to be at least the min counter of the system. This is because its counter was the same for a while and so it's likely to be the least, as everyone else's counter has advanced. Now, if I simply pick the min=1, P will start hogging the resources. So, whenever a process becomes active its counter has to be at least the minimum counter in the system, and then from that point on, its counter is incremented. If a new process arrives and there are existing processes running its counter can't be initialized to 0. Its counter has to be initialized to the min counter of any active task in the system. That way it is at least starting as the minimum task in the system and not 0.

Question: Is the value of s_{min} going to always increase?

Answer: Yes. It's basically tracking the process/container that has received the least service (least amount of cycles in the system). So, it will increase monotonically.

10.2 Docker and Linux Containers

Docker uses abstraction of Linux containers and additional tools for easy management.

1. Portable Containers - With LXC, we would have to use namespaces and cgroup commands to construct a container on a machine. With Docker, all of this information can be saved in the container image and this image can be downloaded and run on a different machine.
2. Application Centric - Docker can be used for designing applications quickly. Software can be distributed through containers.
3. Automatic Builds.
4. Component Reuse - Helps to create efficient images of containers by only including libraries/files not present in the underlying OS. Achieved using UnionFS.

Question: Does Docker deal with packages dependencies for the applications running in the container?
Answer: Packages needed for an application are included in the Docker image. Specific version libraries that are required for application need to be included in the image if they are not present in native OS. OS Does not need to worry about these libraries or issue of incompatible versions.

Question: Docker images are also available for Windows/MAC. How is it possible to run Linux Containers on Windows/Mac?
Answer: Docker provides a hidden VM of linux and containers run on top of it. These are small barebone linux kernels that run on non-Linux platform. Docker does not use true OS virtualization for running Linux containers on other platforms, as it would have to translate Linux calls to other platform calls (Windows/Mac Calls).

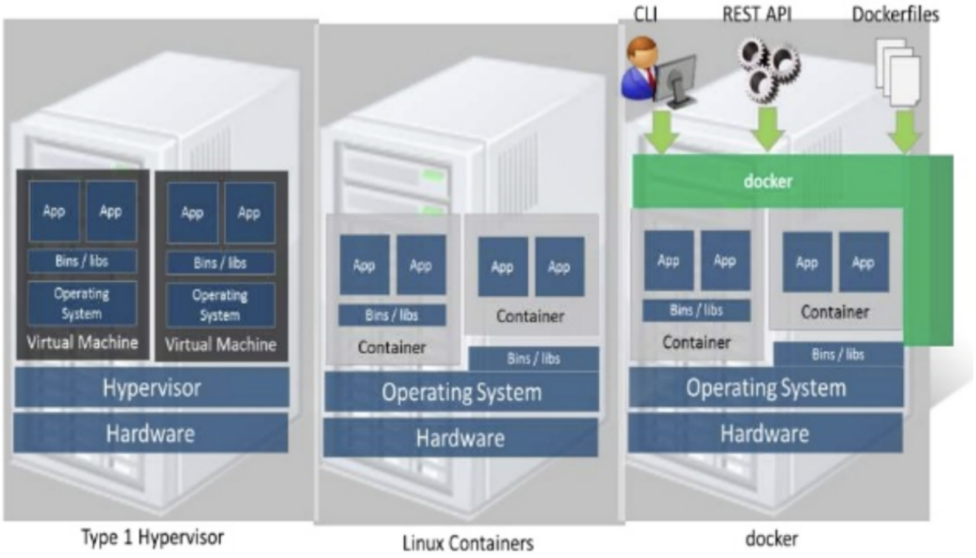

Figure 10.1: Visualization of type 1 hypervisors vs Linux containers vs docker.

Docker is basically a layer around the Linux container (refer Fig. 10.1) that provides tools to build these containers very easily and distribute them. For example, we can distribute them through git repositories. We can just download a container from a git repository and run it on any machine. Docker images will run anywhere Docker runtime is present. All we need to do is download a Docker image and just say run. We don't have to create a container or configure it (e.g., allocate resources). Docker has done all of it.

Docker is a Linux-based OS virtualization technology. However, it can run on any platform (e.g., Mac OS). This is because there is a hidden version of Linux that Docker is starting which is not visible to us.

Question: Is Docker a type 2 hypervisor?

Answer: Docker is a container management runtime. So, it is not technically a type 2 hypervisor. It is using a type 2 hypervisor to run Linux in the background, and then it's running on that Linux.

Question: Is Docker performance the same in all kinds of systems?

Answer: If we are running Docker in any virtualized environment, there will be some overhead which is not going to be present if we are running it in a native environment. For example, if Docker is running on native Linux, the overhead will be less than if it is running on a Linux VM that's running on a Mac.

Question: If you have multiple containers running, do each of them run on a separate VM?

Answer: All of them run on a single Linux instance. if the machine does not have native Linux it will create a virtual machine with Native Linux and all the containers will run on that machine.

Question: If you create a virtual Linux can you go into the shell of the Linux?

Answer: The docker prompt that was visible during the demo was a shell on that Linux machine. It's just that it's inside a container.

Question: To create a python image, you cannot enter the shell; but if you do Ubuntu, you can enter.

Answer: That's not the case. All containers, regardless of what is packaged in them (say, python package),

run on a Linux instance. So, we will get a shell, which is a shell for the container. The application is sandboxed in the container.

Question: In the language image, we cannot enter the shell.

Answer: That's not the case. Whatever the language runtime is, it is still running on that virtualized Linux instance. Even though we showed a web server package in the demo, we still haa a shell that is on the native instance. If we do "top", we will see our process(es) running. The Linux instance may have other running processes, but we don't see everything as we did a namespace and hence we can see only the processes that are running inside our container.

Question: Are namespace and containers the same?

Answer: No. A namespace is a way by which we can limit what a process can see. A container is a namespace + C-group (which allocates some resources to those processes) + some other things. A container has a namespace associated with it, but a namespace itself doe not make a container.

PlanetLab: Virtualized architecture used for research by students in different locations.

10.3 Code, Process and VM Migration

The motivation behind developing techniques for code, process and VM migration is that migrating these components of a system helps improve performance and flexibility. For example, in distributed scheduling, we submit a job on one machine. However, if that machine is overloaded, we would want to migrate either the code (of the submitted job) or the process itself to some other machine and improve performance. From a flexibility standpoint, migration helps in the sense that we can configure distributed systems dynamically. For example, clients can download software on demand from some repository and don't need the software to be preinstalled (refer Fig. 10.2).

There are two types of migration models:

- **Process Migration (aka strong mobility):** This includes the migration of all the components of a process, i.e., code segments, resource segments and execution segments. An active process (an already executing program) on a machine is suspended, its resources like memory contents and register contents are migrated over to the new machine, and then the process execution is restarted. It involves significant amount of data transfer over the network.
- **Code Migration (aka weak mobility):** In this model only the code is migrated and the process is restarted from the initial state on the destination machine. The network transfer overhead is low since only the code is transferred. Many scenarios map to code migration. For example, filling out a web form and hitting submit is a form of code migration because the form becomes a small piece of code that goes to the server, which then processes it and executes. Another example is doing a web search. Web search can be thought of as a query for some words, and we want all the pages for that query. So, that's a "program" we made. The query actually moves to some other system (in this case, a search engine) and is executed there. Simple examples of code migration are taking a real Java program or a binary, moving it and executing it in another machine. Docker is also considered to be an example of code migration. Additionally, anything that is "download and execute" is also an example of code migration — for example, downloading device drivers on demand for new hardware.

  We can use code migration to get better parallelism (replicating same code across machines).

Question: When you do code migration, are you doing process migration?

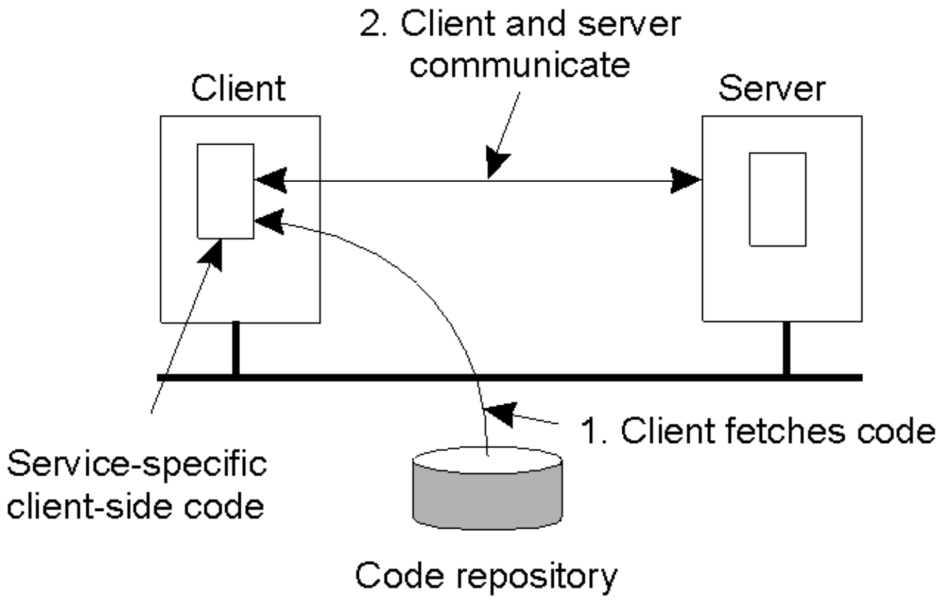

Figure 10.2: Motivation for migration – adds flexibility.

Answer: Code migration simply means moving files, not processes. For example, Python code is a file. If we want to execute it remotely, we will copy that program somewhere else and run it. The process is created when we start executing that code on a remote machine.

Question: Why is a search query an example of code migration?
Answer: Keywords typed in a search bar basically become part of a query and query is a program. On pressing submit, the query is sent to another machine and is executed there. This illustrates migration of code from client machine to the server machine.

Question: In process migration, if you suspended an active process and migrated it, how do you take care of its state?
Answer: The specific state of process like its memory contents can simply be written onto a disk and the process can be resumed elsewhere. Debuggers perform a similar operation.

Question: Does the migration have to be only between client and server or can it be between a cluster of servers?
Answer: Migration is not limited to just client and server. Migration is independent of the source and destination of the code or process.

Migration models: Migration can be sender-initiated or receiver-initiated. An example of receiver-initiated migration is a browser downloading a Java applet or Flash application from the server. In sender-initiated, the sender process has the code and sends it somewhere else. An example of sender-initiated migration is a web search and database query. Fig. 10.3 shows a flowchart of the migration models.

A process can be migrated or cloned. In the case of migration, the complete process is moved to a different machine. In cloning, a copy of the process is created on a different machine, and we kill the process in the sending machine and start executing the process in the receiving machine. We can also allow both copies to execute. Cloning is a convenient way of replicating the process. An example of cloning is forking a process.

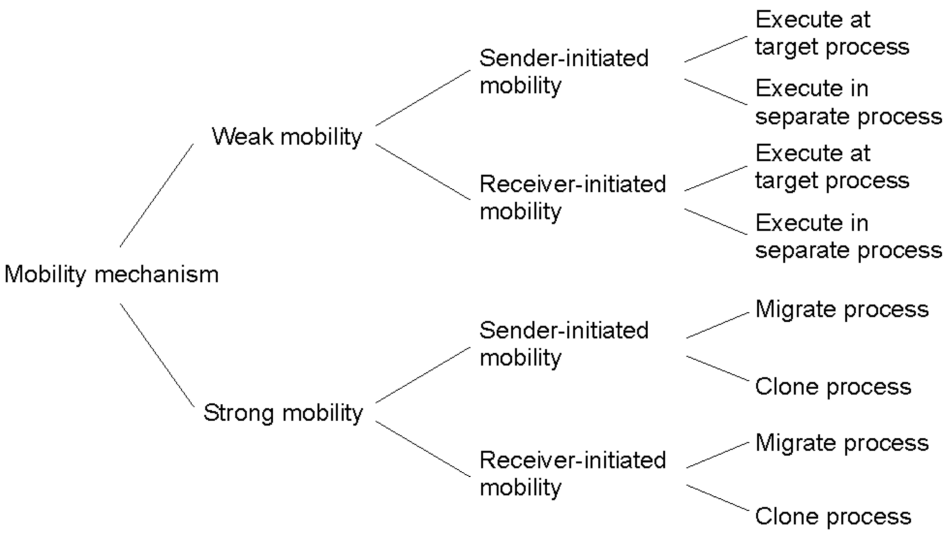

Figure 10.3: Migration models.

Question: What does it mean to "execute at target process" or "execute in separate process"?

Answer: Let's take an example to understand this. Say, we have a browser that contacts a server. The server sends the browser a small piece of code (say, JavaScript). After migration, if the Javascript executes in the same process of the browser, then it is "execute at target process". In contrast, if the browser process created a second process and Javascript was being executed in that second process, it is "execute in separate process".

Question: Whose decision is it to decide whether to do it in the target process or a separate process? Is it the receiver or the sender?

Answer: Typically it's neither. It is the developer's decision (application-level decision).

What happens to the resources that the process was accessing? Let's say a process migrated to a remote machine was accessing a file on the local machine. The process needs to read/write to the file, but it is still in the previous machine. If the file is not present on the new machine, the process will throw an error. So, we need to deal with all kinds of resources that the process was accessing when we deal with code or process migration. How we are going to handle resources depends on what type of resource it is. The process must continue to have access to the resources even after migration to avoid any errors.

We classify the resources along two dimensions. To decide whether to migrate a resource attached to a process or not, first, we look at the nature of binding of resource to a process. There are three types of resources to process bindings:

- **Identifier:** Hard binding, one that you cannot substitute. Least flexibility. An example is URL for a website.
- **Value:** Slightly weaker binding. We can substitute it with another similar resource. Libraries used in Java are a good example. If the Java process was using JVM in the previous machine, as long as the new machine has a JVM of the same (or compatible) version, the Java process will work fine.
- **Type:** Weakest binding. We can substitute it with another resource which need not be exactly similar.

Maximum flexibility. An example is a local device like a printer — different printer models will work just fine as long as there is a printer ready to accept print jobs.

Second, it is also necessary to look at the cost of moving resources. This can also be classified into three categories:

- **Fixed:** Can't be moved. E.g., an ethernet card or an IP address.
- **Fastened:** High cost of moving. E.g., databases. Databases can be moved, but if their size is large, moving them is expensive (in terms of time, or, network bandwidth cost, etc.)
- **Unattached:** Very low cost of moving. E.g., files.

Different combinations of resource-to-machine binding and process-to-resource binding are tabulated below:

	Unattached	**Fastened**	**Fixed**
By Identifier	MV (or GR)	GR (or MV)	GR
By Value	CP (or MV, GR)	GR (or CP)	GR
By Type	RB (or GR, CP)	RB (or GR, CP)	RB (or GR)

where GR means establishing global system-wide references (resource can't be moved, so giving it remote access), MV means moving the resources, CP means copying the resource and RB means rebinding process to locally available resource.

Migration in heterogeneous systems: Here, the assumption is that the sender and receiver machines are heterogeneous (different hardware/OS etc.). If the OSes are different (say, Windows and Linux), we cannot just copy the memory of a Windows process to the Linux machine and expect it to run, due to different binary formats in the different OSes. If we have different hardware then the instruction sets will be different (say, Intel vs ARM). Thus, process migration is much harder due to these constraints. Code migration is easier. For example, as long as Python is installed on both machines, a Python program written on a Linux machine will run fine on a Mac.

Only if we can abstract out the differences in OS/hardware, we can migrate processes. For example, Java. Java runs on JVM which abstracts out the OS/hardware differences. So, a Java process running on Windows can still run on some other platform.

Question: When you get a Docker image is that process migration or is that code migration?

Answer: If we take a Docker image and start it, that's code migration because we got the code for the program. It's just packaged in a container. On the other hand, if we have a running container and we move that running container to another machine, that's process migration because the container actually has active processes running.

Question: Would you want to do checkpoint and restart instead of migrating a process?
Answer: Checkpoint and restart is a standard way to implement process-migration and not an alternative to process-migration.

Question: Would you want to do checkpoint and restart instead of migrating a process?
Answer: Checkpoint and restart is a standard way to implement process-migration and not an alternative to process-migration.

Question: If JVM is a value resource, why can't we bind JVM of the new machine to newly migrated process ?
Answer: If exact same version of JVM as required by the process is available on the new machine, then we

can bind it to the process. However a different JVM version can be incompatible and cause the program to crash.

10.4 Virtual Machine Migration

If a process is communicating with some other process, then process migration is much harder as the IP addresses of the sender and receiver machines are different and we cannot move IP addresses across physical machines. Virtual machine migration (VMM) gives us a way we can actually deal with even network connections of a process. Even the network connections will move when we migrate a VM. The IP address will also move and it will not change.

VMs can be migrated from one machine to another, irrespective of architectural differences. A VM consists of its OS and some applications running on this OS. So, in VM migration, the OS and these applications are migrated with negligible downtime. As the processes inside a VM will also move, VM migration also involves process migration. VM migration is usually done live; that is, it keeps executing during migration. Applications continue to run, nothing has gone down, and then after a while, the VM disappears from one machine and shows up on another machine with the applications still continuing to run.

There are two methods for VM migration, but in this lecture, we only talk about Pre-copy migration.

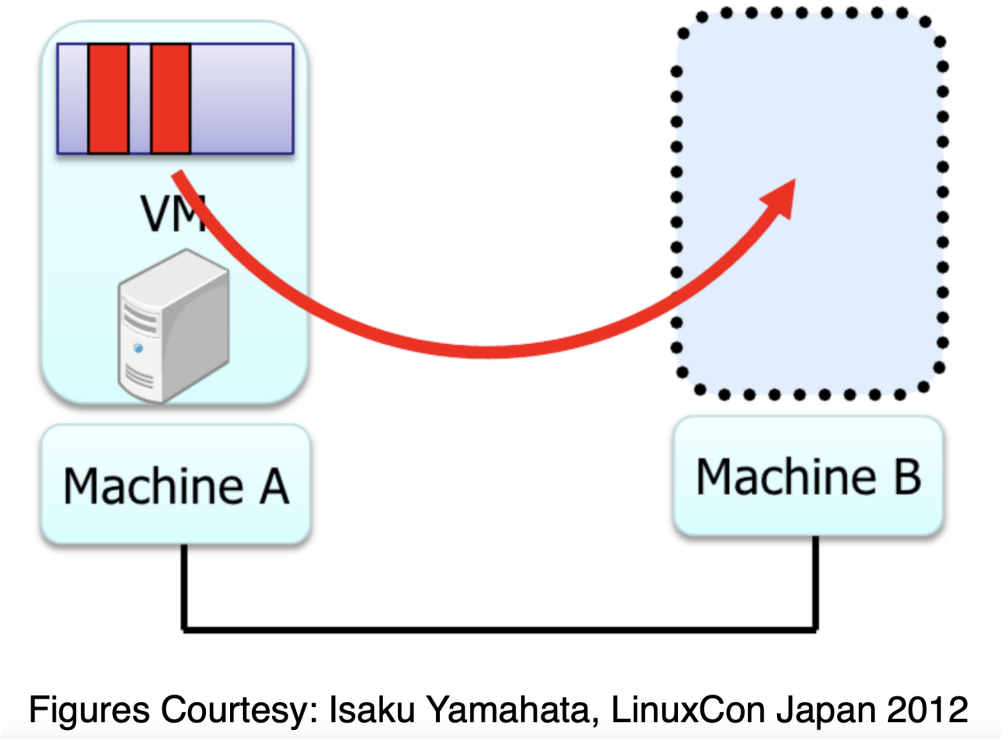

Figure 10.4: Pre-copy VMM.

Pre-copy Migration: Fig. 10.4 shows pre-copy VMM (A and B are two physical machines). The process of pre-copy migration involves the following steps:

1. Enable dirty page tracking. This is required to keep a track of pages which have been written to.
2. Copy all memory pages to destination.
3. Copy memory pages which were changed during the previous copy.
4. Repeat step 2 until the number of memory pages is small.
5. Stop VM, copy rest of memory pages at destination and start VM at the destination.
6. Send ARP packet to switch

Question: When a page changes are you going to send the entire page or are you only going to send what part of the page that changed?

Answer: In this case we'll only track dirty pages and send the entire page. If we want to send only the diff, then we have to keep the old copy and the new copy and do a difference, and that means even more memory allocation and added complexities.

Question: Till when do we repeat Step 4 above?

Answer: We assume that we send all the pages the first time. Before sending again, only a fraction of the pages change. Since it is a subset, it will take less time than the previous round, as the network bandwidth is fixed. This is true for all rounds; that is, each round takes less time than its previous round (assuming no adversarial application). Eventually, after n iterations, we will have a small amount of memory that we need to move. At that point, we go to step 5, that is, stop the VM and copy the remaining pages to the destination physical machine. Since we stopped the VM, no new pages were dirtied during this round.

Note that although it says "live" migration, there is a small time during when the VM is stopped.

Question: What is the threshold where the stop time is small?

Answer: That's configurable. We want the pause time to be as small as possible so that it's almost not noticeable. So, it will depend on factors like the network bandwidth between the two machines.

Question: What happens if there's a network failure?

Answer: If anything fails, VM migration will stop, and it will not continue.

Handling "IP migration": The host machines have different IPs. This is why process migration doesn't work. However, VM migration works because now there are two different IPs — VM IP and host IP — and we can actually keep the VM IP the same while migrating the VM. The applications running in the VM are coupled with the VM IP, and hence these connections do not break even after VM migration. For everything to work after migration, there is one last step — an ARP packet is sent to the ethernet switch to update the ARP-MAC mapping (although IP remains the same, MAC address has changed since physical hosts are different).

Question: Is the IP address is the VM public or private?

Answer: It doesn't matter. The applications are actually tied to the VM IP address and not the host IP address. The only thing is VM IP should be different from the host IP (source/destination host). The application connections will not break as VM IP remains the same.