

Compilation Of Virtualization Methods

Shivapooja Patil
dept. of Electronics and
Telecommunication
Vishwakarma Inst of Information Technology
Pune, Maharashtra, India, 411048

Venkatesh Tiwari
dept. of Electronics and
Telecommunication
Vishwakarma Inst of Information Technology
Pune, Maharashtra, India, 411048

Abstract: Currently, virtualization serves to distribute a physical computer's capabilities among multiple operating systems. The concept of Virtual Machines (VMs) originated in 1964 through an IBM project known as CP/CMS system. There exist various virtualization techniques supporting the execution of complete operating systems. These techniques are categorized based on their handling of modified and unmodified guest OSs. For modified guest OSs, we explore operating system-level virtualization and para-virtualization. On the other hand, techniques like binary translation and hardware-assisted virtualization enable the execution of unmodified guest OSs. Additionally, a summary of resource management facilities is presented to aid in the capacity planning and consolidation of server applications without delving into specific points or details.

Keywords— virtualization, CPU Protection, Capacity Management, Hypervisor

INTRODUCTION

Over the past decade, virtualization has become a widely utilized technology that allows the shared use of physical computer capabilities by distributing resources among various operating systems (OSs). Its origins date back to IBM's 1964 project, which pioneered the concept of Virtual Machines (VM) through the CP/CMS system and later evolved into the Virtual Machine Facility/370. This system enabled a single computing machine to function as multiple copies of itself, each governed by its own OS (the CMS component).

The groundwork laid by IBM, along with subsequent research efforts by Goldberg and Popek in the 1970s, provided a comprehensive understanding of the principles and advantages inherent in low-level virtualization technologies. Interest in virtualization waned until 1999, when VMware Inc. introduced the VMware Virtual Platform tailored for the x86-32 architecture.

Literature Review

The survey will explore the journey of virtualization, starting from its inception in the 1960s with IBM's groundbreaking projects and subsequent advancements made by researchers like Goldberg and Popek in the 1970s. It will delve into the different facets of virtualization, covering hardware, software, network, and storage virtualization, spotlighting their unique traits and applications.

Furthermore, the survey will focus on hypervisor technologies, distinguishing between Type 1 and Type 2 hypervisors and their roles in managing virtual environments. It will also highlight containerization tools such as Docker and Kubernetes, showcasing their significance in lightweight virtualization and modern software development.

Examining the impact of virtualization on performance and strategies for enhancing efficiency will be a key aspect. Security concerns specific to virtual setups and potential remedies will also be discussed.

Moreover, the survey will emphasize real-world applications across various industries like cloud computing, data centres, and edge computing. It will touch upon emerging trends like serverless computing and virtualization's role in handling AI/ML workloads, while also spotlighting recent advancements and future possibilities in virtualization technology.

A. OS-level virtualization:

his approach allows the host OS to run multiple isolated Containers, also known as Virtual Private Servers (VPS), jails, or virtualized servers, each sharing the same kernel as the host OS. Examples of this technique include Linux-V Server, Solaris Zones, and Open VZ. While it offers low overhead and wide implementation, it doesn't support multiple kernels.

On the other hand, "Para-virtualization" is a different technique involving the addition of specialized instructions (Hypercalls) that replace the instructions of the actual machine's instruction set architecture. Examples of Para-virtualization solutions include Denali, Xen, and Hyper-V. In the x86 architecture, the Virtual Machine Monitor (VMM) or Hypervisor operates just above the physical hardware (Ring 0), allowing guest OSs to function at higher levels. Although Para-virtualization supports multiple kernels, modifying the kernel of guest OSs to utilize Hypercalls fully is a drawback.

B. Hardware Virtualization: Binary translation is a technique used to emulate one computer's processor architecture on another. It allows running unmodified operating systems by translating one instruction set into another. An example is QEMU, a processor emulator supporting various architectures like x86, ARM, SPARC, and more. QEMU lets you run OS developed for these processors on different hardware. While

this emulation offers multi-platform flexibility, it also causes overhead due to translating the complete instruction set, impacting performance.

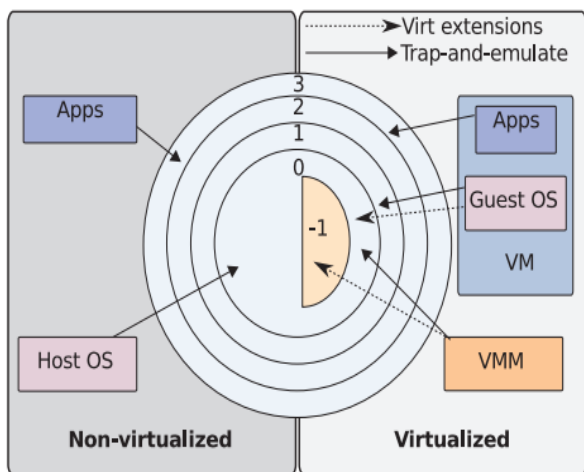


Fig: 1 CPU Protection Mode

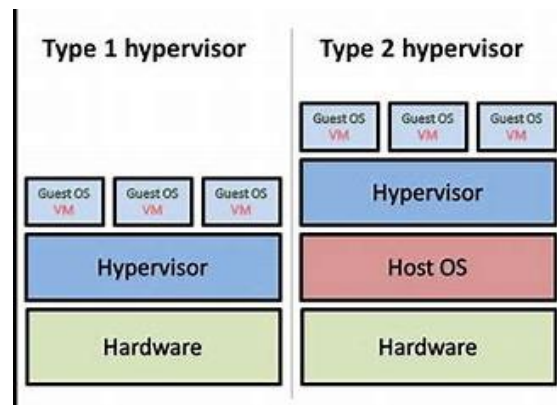
Binary translation, often used in virtualization, combines translation with direct execution. It's used to create copies (VMs) of physical machines by translating only a small set of processor instructions, like those needing privileged execution. Examples include VMware solutions and QEMU Accelerator. This method performs better than full emulation, with lower overhead, but it's limited in supporting unmodified guest OSs to those compatible with the host CPU.

Hardware-assisted virtualization, on the other hand, leverages processor extensions introduced by companies like AMD and Intel. This hardware enhancement creates a higher privileged mode (Ring -1) in the processor architecture, enabling unmodified guest OSs to run in Ring 0 while the VMM or Hypervisor operates in Ring -1. It improves virtualization performance by offloading tasks from software to hardware. Virtualization technologies like KVM, VirtualBox, Xen, Hyper-V, and VMware products use this hardware support for efficient virtualization.

D. Hypervisors: A Hypervisor, also known as a Virtual Machine Monitor (VMM), is a software layer that virtualizes all the resources of a physical machine, allowing the execution of multiple virtual machines (VMs). There are two main types of Hypervisors: Type I and Type II.

Type I Hypervisors, also called native Hypervisors, operate directly above the hardware in the Ring of highest privilege. They control all VMs and support Classic system VMs. Examples include VMware ESX, Hyper-V, and Xen.

On the other hand, Type II Hypervisors, known as hosted Hypervisors, run within an operating system, sharing the Ring of the Host OS. They support Hosted VMs and examples include VMware Server and VirtualBox.



Type I Hypervisors have dedicated resource management components for memory, CPU, and I/O, including a VM scheduler for CPU resource management. In contrast, Type II Hypervisors rely on the host OS's process scheduler, treating each running VM as another OS process.

E. Capacity Management and Optimization: Choosing between different virtualization solutions ultimately depends on how well they meet infrastructure needs compared to their provided features, like portability, migration, snapshots, and administration capabilities.

Platforms like Xen and VMware ESX offer similar CPU resource management features, such as SMP support, proportional resource allocation through shares, processor affinity, and max CPU limits. VMware ESX adds features like resource pools and minCPU limits. Dynamic memory management, supported by Xen, VMware ESX, and KVM, is currently facilitated through memory ballooning.

These solutions enable workload consolidation on physical nodes using specific VM schedulers for CPU resource management. Initially, administrators define resource assignments during VM creation based on infrastructure policies. However, achieving optimal performance requires deep knowledge of running VMs, their needs, and workload types. Dynamic changes in resource allocation pose challenges in dynamic environments for administrators.

Clustering VM-based resource providers is possible due to features offered by virtualization platforms. Yet, each application exhibits unique behaviours and requires varying hardware and software resources. Managing these dynamic features within a cluster becomes complex, making manual management impractical for administrators.

Although virtualization management tools assist in setting physical resource proportions, proper configuration demands expertise and adherence to suggested guidelines to match each VM's required resources effectively, including memory, CPU, disk, and more.

CONCLUSION

Different virtualization solutions offer varying CPU management features, like weighted shares and limits. Planning for each VM in large setups is tricky, requiring expertise to configure for different workloads. The VMM manages hardware without needing workload specifics, letting upper layers create app-aware resource managers. We're building a cloud management layer to abstract virtualization for high-level app needs on physical machines.

ACKNOWLEDGMENTS

We're deeply grateful for the unwavering support and guidance provided by the faculty and resources at Vishwakarma Institute of Information Technology, Pune, India. A special acknowledgment goes to our department of Electronics and Telecommunication for their invaluable assistance throughout the research and writing process. Their encouragement and resources were pivotal in making this work possible.

REFERENCES

1. Creasy, R. J. (1981). "The origin of the VM/370 time-sharing system." *BM Journal of Research and Development*, 25(5), 483–490.
2. Goldberg, R. P., & Popek, G. J. (1973, 1974). "Architecture of virtual machines" and "Formal requirements for virtualizable third generation architectures." *ACM Workshop on Virtual Computer Systems*.
3. VMWare Inc. (1999). "Introducing VMware virtual platform." Technical white paper.
4. Lawton, K. P. (1999). "Running multiple operating systems concurrently on an ia32 pc using virtualization techniques." Unpublished.
5. Barham, P., et al. (2003). "Xen and the art of virtualization." *Proceedings of SOSP '03*.
6. Uhlig, R., et al. (2005). "Intel virtualization technology." *Computer*, 38(5), 48–56.
7. Smith, J. E., & Nair, R. (2005). "The architecture of virtual machines." *Computer*, 38(5), 32–38.
8. Whitaker, A., Shaw, M., & Gribble, S. D. (2002). "Denali: Lightweight virtual machines for distributed and networked applications." Technical report.

Part II: Skill Competency Exam Operating System

1. Implementation of job scheduling algorithm

a. FCFS :

Code :

// Venkatesh Tiwari Roll No: 312050

// Shivapooja Patil Roll No: 312062

package fcfs;

import java.util.Scanner;

public class fcfs

{

```
public static void main (String args[])
{
Scanner sc = new Scanner (System.in);

System.out.println("Enter number of processes:");

int n = sc.nextInt();

int pid [] = new int[n];

int ar [] = new int[n];

int bt [] = new int[n];

int ct [] = new int[n];

int ta [] = new int[n];

int wt [] = new int[n];

int temp;

float avgwt=0;

float avgta=0;

for(int i=0;i<n;i++){

System.out.println("Enter process"+ (i+1)+"arrival time:");

ar[i]=sc.nextInt();

System.out.println("Enter process"+ (i+1)+"burst time:");

bt[i]=sc.nextInt();

pid[i]=i+1;

for(int k=0;k<n;k++){

for(int j=0;j< n-(k+1);k++){

if(ar[j]>ar[j+1]){

temp=ar[j];

ar[j]=ar[j+1];

ar[j+1]=temp;

temp =bt[j];

bt[j]= bt[j+1];

bt[j+1]= temp;
```

```

temp= pid[i];

pid[j]=pid[j+1];

pid[j+1]=temp;

}

}

}

for(int m=0;m<n;m++){

if(m==0){

ct[m]= ar[m]+bt[m];

}

else{

if(ar[m]>ct[m-1]){

ct[m]= ar[m]+bt[m];

}

else{

ct[m]= ct[m-1]+bt[m];

}

}

}

ta[i]=ct[i]-ar[i];

wt[i]=ta[i]-bt[i];

avgwt= avgwt+ wt[i];

avgta= avgta+ ta[i];

}

System.out.println("\npid arrival burst complete turn waiting ");

for(int i=0;i<n;i++){

System.out.println(pid[i]+"\\t"+ar[i]+"\\t"+bt[i]+"\\t"+ct[i]+"\\t"+ta[i]+"\\t"+

wt[i]);

}

sc.close();

```

```
System.out.println("\naverage waiting time"+(avgwt/n));
```

```
System.out.println("average turnaround time"+(avgta/n));
```

```
}
```

```
}
```

Output

```

<terminated> fcfs [Java Application] C:\Users\Lenovo\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\
Enter number of processes:
3
Enter process1arrival time:
1 4 6
Enter process1burst time:
Enter process2arrival time:
Enter process2burst time:
2 3 5
Enter process3arrival time:
Enter process3burst time:

pid arrival burst complete turn waiting
0      0      0      0      0      0
2      6      2      8      2      0
3      3      5     13     10     5

average waiting time1.6666666
average turnaround time4.0

```

b) SJF

Source Code:

```

class Process
{
    int pid; int bt; int
art;
    public Process(int pid, int bt, int art)
    {
        this.pid = pid; this.bt = bt;
        this.art = art;
    }
}

public class Main
{
    static void findWaitingTime(Process proc[], int n, int wt[]) {
        int rt[] = new int[n]; for (int i = 0; i < n; i++) rt[i] =
proc[i].bt;

        int complete = 0, t = 0, minm = Integer.MAX_VALUE; int shortest = 0, finish_time; boolean check
= false;

        while (complete != n) {

            for (int j = 0; j < n; j++)
            {
                if ((proc[j].art <= t) &&
(rt[j] < minm) && rt[j] > 0) { minm = rt[j]; shortest = j;
check = true;
                }
            }
            if (check == false) { t++; continue;
            }
            rt[shortest]--;
            minm = rt[shortest]; if (minm == 0) minm =
Integer.MAX_VALUE;
            if (rt[shortest] == 0) {
                complete++; check = false;
            }
        }
    }
}

```

```

finish_time = t + 1;
wt[shortest] = finish_time - proc[shortest].bt -
proc[shortest].art;

if (wt[shortest] < 0) wt[shortest] = 0;
} t++; }
}

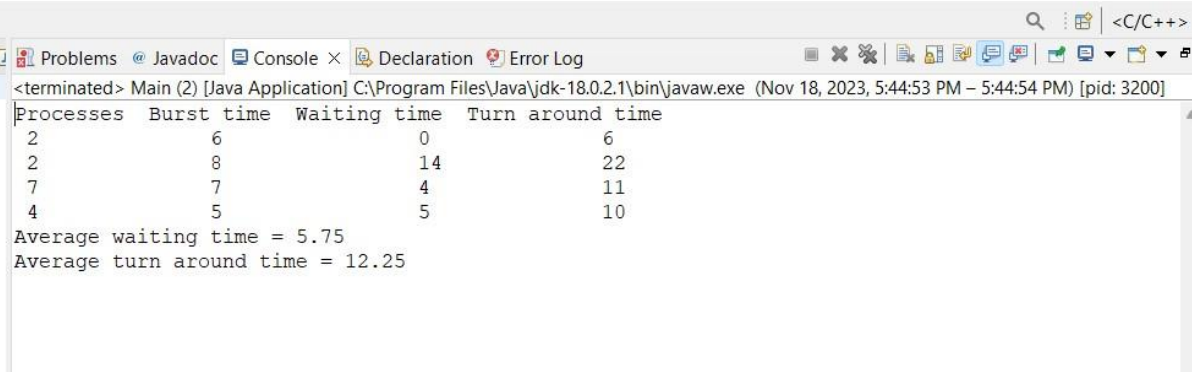
static void findTurnAroundTime(Process proc[], int n, int wt[], int tat[])
{
for (int i = 0; i < n; i++) tat[i] = proc[i].bt + wt[i];
}
static void findavgTime(Process proc[], int n)
{
int wt[] = new int[n], tat[] = new int[n]; int total_wt = 0, total_tat = 0; findWaitingTime(proc,
n, wt); findTurnAroundTime(proc, n, wt, tat);
System.out.println("Processes " +
" Burst time " +
" Waiting time " + " Turn around time"); for (int i = 0; i < n;
i++) { total_wt = total_wt + wt[i]; total_tat = total_tat + tat[i];
System.out.println(" " + proc[i].pid + "\t\t"
+ proc[i].bt + "\t\t" + wt[i]
+ "\t\t" + tat[i]);
}

System.out.println("Average waiting time = " +
(float)total_wt / (float)n);
System.out.println("Average turn around time = " +
(float)total_tat / (float)n);
}
public static void main(String[] args)
{
Process proc[] = { new Process(2, 6, 1), new Process(2, 8, 5), new Process(7, 7,
3), new Process(4, 5, 9)};

findavgTime(proc, proc.length);
}
}

```

Output:



```

<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-18.0.2.1\bin\javaw.exe (Nov 18, 2023, 5:44:53 PM - 5:44:54 PM) [pid: 3200]
Processes Burst time Waiting time Turn around time
2          6          0          6
2          8          14         22
7          7          4          11
4          5          5          10

Average waiting time = 5.75
Average turn around time = 12.25

```

2) Implementation of Page replacement algorithm (Anyone) LRU

Source Code:

```

import java.util.ArrayList;

import java.util.Scanner;

public class LRU {

```

Published by :
<http://www.ijert.org>

```
public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the capacity : ");

int capacity = scanner.nextInt();

System.out.print("Enter the number of page references: ");

int numReferences = scanner.nextInt();

int[] arr = new int[numReferences];

System.out.println("Enter the page reference sequence: ");

for (int i = 0; i < numReferences; i++) {

arr[i] = scanner.nextInt();

}

ArrayList<Integer> s = new ArrayList<>(capacity);

int page_faults = 0;

for (int i : arr) {

if (!s.contains(i)) {

if (s.size() == capacity) {

s.remove(0);

s.add(capacity - 1, i);

} else {

s.add(i);

}

page_faults++;

} else {

s.remove((Object) i);

s.add(i);

}

}

System.out.println("Number of page faults: " + page_faults);

}

}
```


Output:

```

<terminated> LRU [Java Application] C:\Users\Lenovo\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\java.exe
Enter the capacity : 3
Enter the number of page references: 15
Enter the page reference sequence:
1 2 3 5 6 3 1 3 2 5 4 6 8 6 7
Number of page faults: 12

```

3) Bankers algorithm for deadlock detection and avoidance Source Code:

```

package bankers;

import java.util.Scanner;

public class bankers {

    static int[][] arrmax;

    static int[][] alloc;

    static int[][] need;

    static int[] avail;

    static int n, r;

    public static void input() {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the no of Processes\t");

        n = sc.nextInt();

        System.out.print("Enter the no of resource instances\t");

        r = sc.nextInt();

        arrmax = new int[n][r];

        alloc = new int[n][r];

        need = new int[n][r];

        avail = new int[r];

        System.out.println("Enter the Max Matrix");

```

Published by :
<http://www.ijert.org>

```

for (int i = 0; i < n; i++) {

for (int j = 0; j < r; j++) {

armax[i][j] = sc.nextInt();

}

}

System.out.println("Enter the Allocation Matrix");

for (int i = 0; i < n; i++) {

for (int j = 0; j < r; j++) {

alloc[i][j] = sc.nextInt();

}

}

System.out.println("Enter the available Resources");

for (int j = 0; j < r; j++) {

avail[j] = sc.nextInt();

}

}

public static void show() {

System.out.println("Process\t Allocation\t Max\t Available\t");

for (int i = 0; i < n; i++) {

System.out.print("\nP" + (i + 1) + "\t ");

for (int j = 0; j < r; j++) {

System.out.print(alloc[i][j] + " ");

}

System.out.print("\t");

for (int j = 0; j < r; j++) {

System.out.print(armax[i][j] + " ");

}

System.out.print("\t ");

if (i == 0) {

for (int j = 0; j < r; j++) {

```

```

System.out.print(avail[j] + " ");

}

}

}

}

public static void cal() {

int[] finish = new int[n];

int[][] need = new int[n][r];

int[] dead = new int[n];

int[] safe = new int[n];

int i, j;

for (i = 0; i < n; i++) {

finish[i] = 0;

}

// find need matrix

for (i = 0; i < n; i++) {

for (j = 0; j < r; j++) {

need[i][j] = arrmax[i][j] - alloc[i][j];

}

}

int flag = 1;

while (flag != 0) {

flag = 0;

for (i = 0; i < n; i++) {

int c = 0;

for (j = 0; j < r; j++) {

if (finish[i] == 0 && need[i][j] <= avail[j]) {

c++;

if (c == r) {

for (int k = 0; k < r; k++) {

```

```
avail[k] += alloc[i][j];

finish[i] = 1;

flag = 1;

}

if (finish[i] == 1) {

i = n;

}

}

}

}

}

}

}

}

}

j = 0;

flag = 0;

for (i = 0; i < n; i++) {

if (finish[i] == 0) {

dead[j] = i;

j++;

flag = 1;

}

}

if (flag == 1) {

System.out.println("\n\nSystem is in Deadlock and the Deadlockprocess are");

for (i = 0; i < n; i++) {

System.out.print("P" + dead[i] + "\t");

}

} else {

System.out.println("\nNo Deadlock Occur");

}

}
```

Published by :
<http://www.ijert.org>

```
public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println(" Deadlock Detection Algorithm ");

input();

show();

cal();

}}
```

Output:

```
<terminated> bankers [Java Application] C:\Users\Lenovo\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre
Deadlock Detection Algorithm
Enter the no of Processes      4
Enter the no of resource instances      3
Enter the Max Matrix
 1 2 3 9 6 6 4 4 5 5 4 4
Enter the Allocation Matrix
1 2 2 7 2 3 4 5 5 6 6 4
Enter the available Resources
1 0 1
Process  Allocation      Max      Available
P1      1 2 2      1 2 3      1 0 1
P2      7 2 3      9 6 6
P3      4 5 5      4 4 5
P4      6 6 4      5 4 4
No Deadlock Occur
```