ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 8, Issue 3, March 2020

VM Performance Optimization by CPU Affinity using NUMA API

Farhana Kausar[1], Vamsi Anamalamudi[2]

Professor, Department of Computer Engineering, Atria Institute of Technology, Bengaluru, Karnataka, India[1]

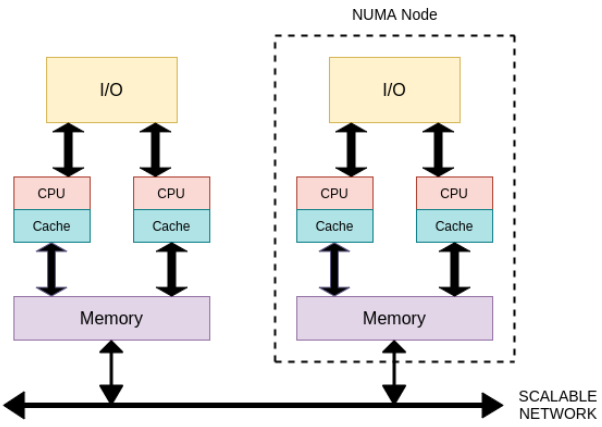
B.E. Student, Department of Computer Engineering, Atria Institute of Technology, Bengaluru, Karnataka, India[2]

ABSTRACT: Hundreds of thousands of servers are set up in the data centres to satisfy the cloud-services. The major problem faced during building highly scalable servers is the VM performance. NUMA node balancing is one of them. Some hypervisors don't deal with NUMA node issues. In order to improve NUMA architecture, we also need some software optimization. Processor affinity and data placement are techniques used for software optimization. In this paper, I'll give an overview of NUMA, issues of NUMA and various optimization techniques for NUMA systems in order to improve the performance of VM's.

KEYWORDS:NUMA, CPU affinity, VM optimization.

I. INTRODUCTION

In the past, Symmetric Multi-processing or Uniform Memory Architecture (UMA) were used. In which processors shared the access to all memory available over a bus. Due to limited bus bandwidth there raised challenges like latency and collisions among multiple CPUs. So Non-Uniform Memory Access came into picture where the requirement of speed between CPU and memory is satisfied.In NUMA architecture, system memory is divided into zones [nodes] as shown below, which are allocated to particular CPUs or sockets. Access to the local memory will be faster than memory connected to remote CPUs on the system. In practice what that means is, as the number of CPUs and memory are growing so large that it no longer makes sense for all memory in your system to be accessible with the same levels of latency from different CPUs.



NUMA ARCHITECTURE

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

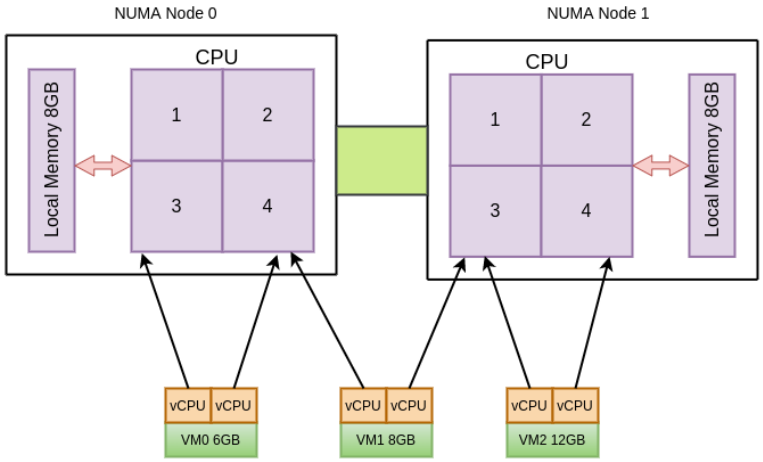
(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 8, Issue 3, March 2020

WHY IS NUMA NEEDED?

- For increase in clock speed.
- To overcome scalability issues.



- To overcome common-channel contention.

The above diagram, VM0 will be assigned with 6GB and 2 cores which are available. VM1 should never get assigned because it is requesting cores from different NUMA nodes. VM3 is requesting for 12GB but NUMA node has 8GB availability, so even VM3 is not assigned.

We can view the system's NUMA topology by use numactl with -H as option:

$ numactl -H

```
available: 2 nodes (0-1)
node 0 cpus: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46
48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78
node 0 size: 288730 MB
node 0 free: 264447 MB
node 1 cpus: 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47
49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79
node 1 size: 290271 MB
node 1 free: 258263 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
```

The above topology tells that the system has 2 NUMA nodes [0,1] can the list of CPUs in their respective nodes and the distance between the nodes.

II. RELATED WORK

When a VM tries to access a memory, which is not part of the same NUMA node it raises a performance issue [Because access time of foreign memory is more]. If VMs are randomly distributed across nodes it might cause a performance issue. There are various methods to improve the NUMA performance. One of them is Manual static

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 8, Issue 3, March 2020

NUMA bindings, which can improve the performance slightly. On quad-socket servers the performance can be improved by using numad and numactl tuning.
Numad is a Daemon which monitors the numa resources dynamically.
Numactl is a command line tool for running specific NUMA policy on processes.

numactl --cpubind=0 --membind=0,1 myprog

Disadvantage of numactl is it applies the policy on the whole program. But more policy control is required. This can be achieved by libnuma. This is a library that can be added to your programs and control the numa policies.

III. PROPOSED ALGORITHM

NUMA API allows us to query NUMA status, change policies, binding to CPUs, etc. of the system. To check if NUMA policy is enabled in the system we need to use the numa_available( ) function, which returns a negative value if NUMA policy is not supported in the system. numa_max_node( ) function returns the max NUMA node found in the system.

1. Check if hyperthreading is enabled in the system [It is important to take care of assigning at least one physical CPU and virtual CPUs]. That means each CPU is viewed as 2 CPUs .Total CPUs will be double of the actual CPUs.
2. Get the CPU list from each node.
3. Get the VMs running on system.
4. Perform CPU pinning of the guest VM with the available CPU based on the VM requirement.
5. Update the CPU list.

IV.PSEUDO CODE

Step 1: To check if Hyperthreading is enabled. We can use lscpu command.If threads per core is 2 then its hyperthreading is enabled.

Step 2: Python snippet to find siblings in the system ie. CPUs list:

```
siblings = []
for cpu in range(no_of_cpus):
    cpucmd = ""
    cpucmd = "cat /sys/devices/system/cpu/cpu"
    cpucmd += str(cpu)
    cpucmd += "/topology/thread_siblings_list"
    sibling_pair = os.popen(cpucmd).read()
    sibling_pair=sibling_pair.rstrip()
    sibs= []
```

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 8, Issue 3, March 2020

```
        sibs = sibling_pair.split(',')
        #print(sibs)
        siblings.append(sibs)
        #print(siblings)
        siblings.append(sibs)
    siblings_list = [list(x) for x in set(tuple(x) for x in siblings)]
    print("Siblings: "+str(siblings_list))
```

OUTPUT: Siblings: [['3', '43'], ['37', '77'], ['14', '54'], ['10', '50'], ['6', '46'], ['11', '51'], ['7', '47'], ['13', '53'], ['25', '65'], ['9', '49'], ['24', '64'], ['8', '48'], ['30', '70'], ['26', '66'], ['32', '72'], ['2', '42'], ['12', '52'], ['39', '79'], ['15', '55'], ['29', '69'], ['35', '75'], ['28', '68'], ['23', '63'], ['34', '74'], ['16', '56'], ['31', '71'], ['22', '62'], ['38', '78'], ['21', '61'], ['17', '57'], ['5', '45'], ['20', '60'], ['4', '44'], ['36', '76'], ['1', '41'], ['27', '67'], ['18', '58'], ['33', '73'], ['0', '40'], ['19', '59']]

Where 3 is physical and 43 is virtual CPU.

Step 3: Python snippet to get VMs running on system using use virsh command.

```
 vm_names = []
    vm = os.popen("virsh list --name").read()
    vm=vm.rstrip()
    vm_names = vm.split('\n')

    print("VM NAMES:"+str(vm_names))
```

OUTPUT:VM NAMES:['centos7', 'ubuntu18.04']

Step 4: Python snippet to do CPU pinning of 2 VMs of 1 core each.

```
command_cpu = "virsh vcpupin"
    command_emulator = "virsh emulatorpin"
    details_vm = dict()
    apply_node = 0
    next_cpu_set = 0
    count = 0
    for vm in vm_names:
        apply_cpu = siblings_list[count][0]
        emulator_cpu = siblings_list[count][0]
        cmd_cpu = command_cpu +" "+ vm +" "+str(apply_node)+" "+str(apply_cpu)+" --live --config"
        cmd_emulator = command_emulator +" "+ vm +" "+str(emulator_cpu)+" --live --config"
```

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 8, Issue 3, March 2020

```
next_cpu_set+=1
#print(cmd_cpu)
#print(cmd_emulator)
count+=1
#CPU PINNING
if os.system(cmd_cpu) == 0:
   print("CPU PINNING "+vm+" with "+apply_cpu+" cpu SUCCESSFUL!")
else:
   print("CPU PINNING "+vm+" with "+apply_cpu+" cpu FAILED!")
#EMULATOR PINNING
if os.system(cmd_emulator) == 0:
   print("EMULATOR PINNING "+vm+" with "+emulator_cpu+" cpu SUCCESSFUL!")
else:
   print("EMULATOR PINNING "+vm+" with "+emulator_cpu+" cpu FAILED!")
```

OUTPUT:

CPU PINNING centos7 with 0 cpu SUCCESSFUL!

EMULATOR PINNING centos7 with 0 cpu SUCCESSFUL!

CPU PINNING ubuntu18.04 with 2 cpu SUCCESSFUL!

EMULATOR PINNING ubuntu18.04 with 2 cpu SUCCESSFUL!

Step 5: Update the CPU allocated above.

We can view the affinity of a task or process by using get_affinity().We can also set affinity to the tasks by using set_affinity() function as follows:

```
pid = 5837
cpus_set = [2,3,6]
print("initial cpus: "+str(get_affinity(vm_pid)))
set affinity(vm_pid,cpus_set)
print("after pinning: "+str(get affinity(vm_pid)))
```

OUTPUT:

initial cpus: [1,4,5]

After pinning: [2,3,6]

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

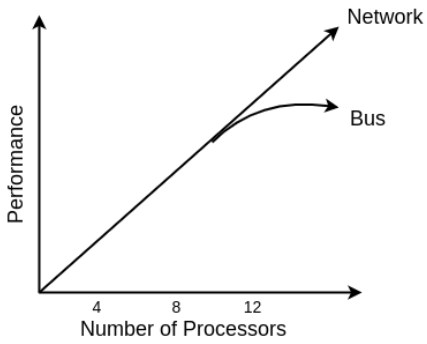
(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 8, Issue 3, March 2020

V. RESULTS AND DISCUSSIONS

Scalability is the main advantage of NUMA architecture. In the below plot we can see NUMA based systems have scalability issues. Initially the architecture scales linearly up to 8-12 processes and then stagnants due to bus contention problems. However, there is no concept of a shared bus in NUMA so it's scalable.



If we use hyperthreading to deploy two vCPUs per physical core then the first step is to enable NUMA affinity, if it is not enabled the conferencing node VM will end up creating multiple NUMA nodes which results in performance loss or degradation. It is not a practical method to use NUMA affinity in all data centre use cases as it is constrained to run a given VM on a certain CPU socket, but is very useful for dedicated capacity.

VI.CONCLUSION

A virtual machine can be assigned to a specific processor using CPU affinity. Restricting the assignment of virtual machines to a specific available processor in a multiprocessor system is possible in this method. Linux NUMA tuning had a positive impact on the performance value which was up to 4.2%. Based on different workload and hardware specific tuning was considered to be a best option. As it is not possible to have a “one size fit all” optimal configuration. On systems with newer CPUs a slight performance gain, or no effect for any benchmarks was observed when manual static NUMA bindings with numactl was implemented. The option of assigning a NUMA node affinity to a VM overrides the hypervisors' dynamic assignment of VMs to NUMA nodes. Therefore, incorporating such a configuration in future study would be interesting.

REFERENCES

[1]Colin Smith, “How to optimize VM memory and processor performance”, TechRepublic, October 26, 2010.
[2]Christopher Hollowell, Costin Caramarcu, William Strecker-Kellogg, Tony Wong, Alexandr Zaytsev,” The Effect of NUMA Tunings on CPU Performance”.
[3]Smira, “API Documentation”, github.io, 2010.
[4]Andi Kleen “A NUMA API for Linux”, SUSE Labs, Aug 2004.
[5]Andreas Kleen, A NUMA API for LINUX*, April 2005