virtdc 0.1.0

[Virtual Datacenter]

[Source code - https://github.com/dcsolvere/virtdc]

[wiki - http://www.utdallas.edu/~dxa132330/virtdc.html]

Contents

[1] INTRODUCTION

1.1. Purpose

Assume that the workloads of the VMs are known when submitted, statically place and schedule and dynamically scale and migrate the VMs to make best resource utilization while never letting users feel short of resources.

[2] VIRTDC ARCHITECTURE AND FRAMEWORK

2.1 VIRTDC – High Level Architecture Diagram

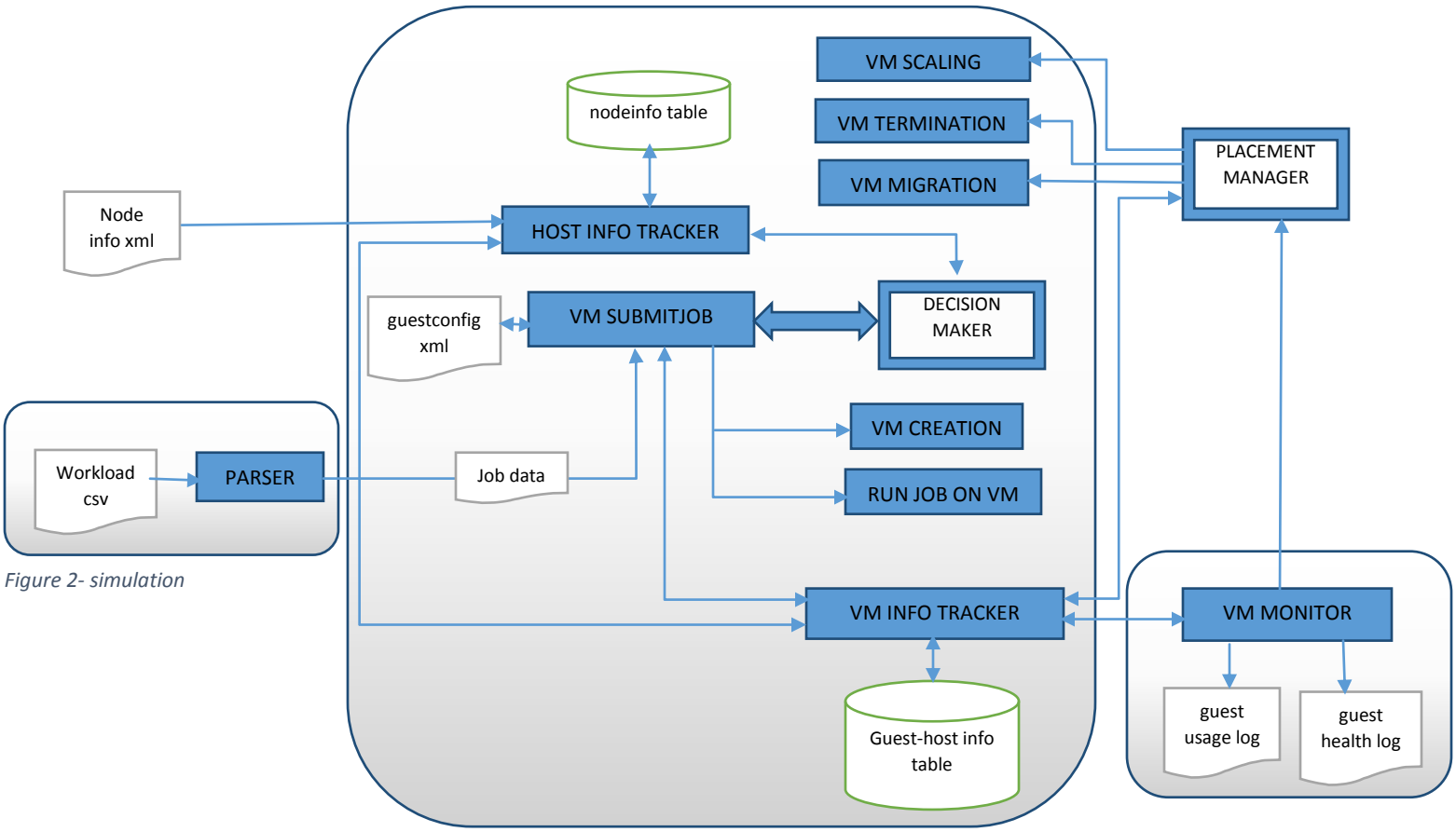

Figure 2- simulation

Figure 1 - framework

2.2 VIRTDC –Framework [libvirt, python, shell script, c]

Virtdc is an API for virtual machine placement and scaling which provides an environment to create, manage and monitor virtual machines effectively. Virtdc – framework provides API's to create virtual machines, maintain information about guest and host, terminate virtual machine and to handle static virtual machine placement. Internally it uses libvirt management API to accomplish this.

One host acts as the master node to create the virtual machines on any slave nodes. Guests can also be placed in master node.

Host Information:

Virtdc API maintains a table to store the host information. In the initial setup of virtdc API this table is updated from the nodeinfo.xml. This xml will be collected from the user. Host Info Tracker module will update the table based on the nodeinfo.xml. Whenever there is a new resource/node update, host info tracker module has to be executed. This module will tweak the table.

host	ip_adddress	max_cpu	max_memory	max_io	avail_cpu	avail_memory	avail_io
node1	192.168.1.11	8	32689796	1.07E+09	8	32689796	1.07E+09
node2	192.168.1.12	8	32689796	1.07E+09	8	32689796	1.07E+09
node3	192.168.1.13	8	32689796	1.07E+09	8	32689796	1.07E+09
node4	192.168.1.14	8	32689796	1.07E+09	8	32689796	1.07E+09

Figure 3

Guest Information:

Virtdc API maintains a table to store the guest information along with the host. This dictionary is accessible from all API's in virtdc but can be updated only by the master node. By default it provides disk/io size of 4GB for each guest.

host	vmid	current_cpu	max_cpu	current_memory	max_memory	io
node1	vm1	1	3	4194304	5242880	4194304
	vm2	2	3	4194304	5242880	4194304
	vm3	1	3	4194304	5242880	4194304
node2	vm4	2	5	4194304	5242880	4194304
	vm5	2	6	4194304	5242880	4194304
	vm6	3	4	4194304	5242880	4194304

Figure 4

2.3 Update Host Information

Data centers can add new resources to the existing system. To achieve that 'Host Info Tracker' module provides an API to update the host information dynamically to the host info table. This module retrieves the information from nodeinfo.xml. So whenever there is an update in the nodeinfo xml, this module has to be executed in the master node to update the host info table.

Nodeinfo xml as follows,

```
<node_info>
        <nodes>
            <node>
            <hostname>node1</hostname>
            <ipv4address>192.168.1.11</ipv4address>
            <max_capacity>
                <cpu_core>8</cpu_core>
                <memoryunit="KiB">32689796</memory>
                <iounit="KiB">1073741824</io>
            </max_capacity>
        <available_capacity>
                <cpu_core>8</cpu_core>
                <memoryunit="KiB">32689796</memory>
                <iounit="KiB">1073741824</io>
            </available_capacity>
        </node>
        <node>
    …
    …

</node_info>
```

Figure 5

2.4 Update Guest Information

Guest info table will be updated whenever a new guest is created on any host. Guest information will be added along with the information about the host on which the guest is to be created. VM_Info_Updater will be responsible for add/update of guest information.

[3] GUEST CREATION

Guests can be created from XML configuration files. Guest configuration can be copied from existing XML from previously created guests or use the dumpxml option. To create a guest with virsh from an XML file:

virsh create configuration_file.xml

Creating a virtual machine XML dump(configuration file)

The following libvirt API gives the configuration XML from the existing guest,

virsh dumpxml [domain-name]

virsh dumpxml base_guest> guestconfig.xml

guestconfig.xml will contain the configuration for the base guest. Virtdc uses this xml as the base guest to create new guest on host by tweaking the guestconfig xml based on the requirement.

Guestconfig.xml looks similar to the following,

```
<domain
type='kvm'id='2'>
            <name>vm_name</name>
            <uuid>vm_uuid</uuid>
            <memoryunit='KiB'>max_memory</memory>
            <currentMemoryunit='KiB'>current_memory</currentMemory>
            <vcpuplacement='static'current='current_cpu'>max_cpu</vcpu>
            <resource>
            <partition>/machine</partition>
            </resource>
            …
            …
</domain>
```

Figure 6

vm_name, vm_uuid, max_memory, current_memory variables will be replaced based on the new guest requirement.

vm_submitjob() creates guest from the configuration xml and updates the information about the guest in virtdc API's

```
vm_submitjob(vmid,cpu,memory,io)
```

This API takes the vmid, cpu, memory, io as the parameters to create the guest machine.

Virtdc uses the base image and clones new guest from the base using the configuration file. It uses virsh tool to create guest,

```
virsh --connect qemu+ssh://"+host+"/system create new_guestconfig.xml
```

Host Identification for placement:

Host will be identified using VM decision maker. Virtdc provides an API to verify whether the guest can be created in any host.

```
is_space_available_for_vm(cpu,mem,io)
```

This API checks the availability of cpu, memory and io from the guest and host information table [node_dict].

[4] GUEST MIGRATION

Guest can be migrated from one host to another. Placement manager will make the decision to migrate the guest. The following diagram illustrates the guest migration. Client host [figure 7] is the master node where the libvirt and virtdc API's are running.

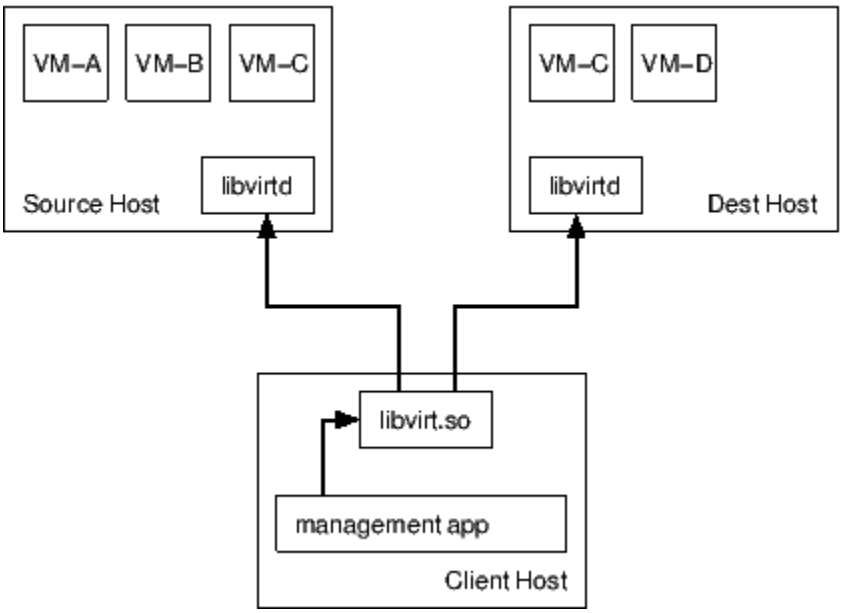

Figure 7 – ref: libvirt portal

All disk images of the guest will be stored in the network shared folder. Libvirt API migrates the currently running CPU processes and memory operations but the disk image of the guest will not be migrated. The guest images will be stored in the network file system (NFS). Even after the migration the guest will be accessing the image from source host.

Guest can be migrated under following scenarios—

1. Load balancing
2. Consolidation
3. Host removal

Migration can be achieved through libvirt API,

```
ssh -q -o StrictHostKeyChecking=no root@source_host"virsh migrate vmid
qemu+ssh://dest_host/system"
```

virtdc API for migration:

The following API in virtdc uses the vmid, source_host, dest_host to migrate the guest from the source host to the destination host. This uses the libvirt API to achieve the migration (as mentioned above).

```
initiateLiveMigration(vmid,source_host,dest_host)
```

[5] GUEST SCALING

Virtdc provides API for cpu scaling and memory scaling. Scaling is not an user request process. Scaling decision will be retrieved from the placement manager and the client SLA configuration.

CPU scaling:

Virtdc API for cpu scaling is as follows,

```
initiateVMCPUScaleUp(hostname, vmid, cpu)
```

virtdc API internally uses libvirt to accomplish the cpu scaling.

Memory Scaling:

Virtdc API for memory scaling,

```
initiateMemScaleUpOrDown(hostName,vmID,memorySize)
```

libvirt API for memory scaling,

```
ssh -q -o StrictHostKeyChecking=no root@'+hostName+' virsh setmem '+vmID+' '+memorySize
```

Source code for virtdc:

https://github.com/dineshappavoo/VMPlacementAndScaling

[6] GOOGLE DATA SIMULATION

Google workload is retrieved from Google cloud storage and parsed using csv parser. Each task is considered as a single guest and a job may have multiple tasks/multiple guests.

Figure 8 – Overview

Workflow diagram for Simulation:

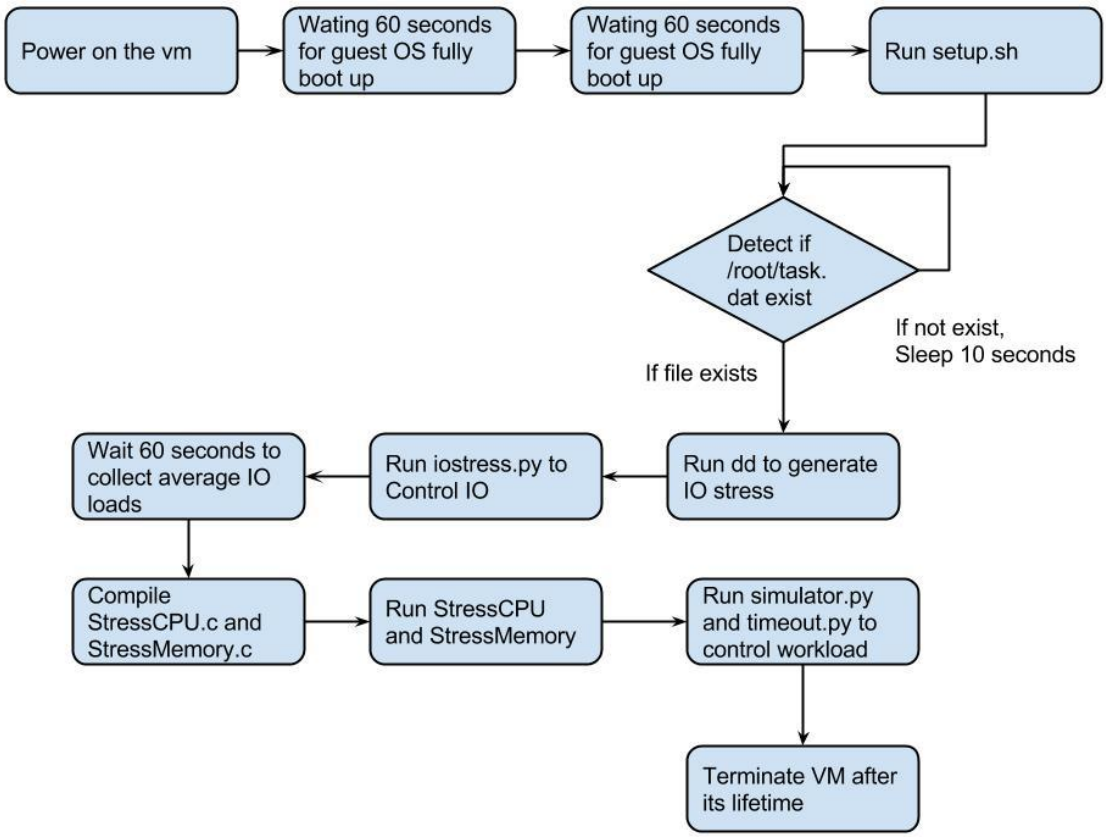

Figure 9 – Simulation workflow

The Simulation module initiates the following processes to simulate the workload parsed from Google data-

1. StressCPU : Keeps the CPU busy by calculating prime numbers on a continuous basis.
2. StressMemory : Keeps the memory busy by performing writes on it.
3. IOStress : Keeps the IO busy by continuously writing/reading disk.

All the above processes once started will be continuously monitored to check if it is overshooting the workload. Based on this data, the execution of the process will be controlled and made to simulate the given workload.

[8] REFERENCES

[1] On Theory of VM Placement: Anomalies in Existing Methodologies and Their Mitigation Using a Novel Vector Based Approach Appendix I: Process Details

[2] On Resource Management for Cloud Users: A Generalized Kelly Mechanism Approach

[3] SLA-aware virtual resource management for cloud infrastructures

[4] VMware Distributed Resource Management: Design, Implementation, and Lessons Learned

[5] Q-Clouds: Managing Performance Interference Effects for QoS-Aware Clouds