High Level Design Document
Bin-packing VM Consolidation Algorithm

Surineni Sampath Kumar 13MCMT49

Contents

1 Detailed Design

1.1 PM Modifier Module

This module will be called by Parser module and User Interface module for

- Adding a Virtual Machine(VM),
- Deleting a VM,
- Switching off a PM,
- Switching on a PM and
- Consolidation

1.1.1 Interface Data Structures

1. PMstruct

PMstruct

Different fields in PMstruct data structure are

1. PM_ID - final String
2. res_cap - integer
3. VM_list - array of type class VMstruct
4. onSate - integer

This is the data structure returned to status() function which is called by User Interface

1.1.2 Internal Data Structures

1. VMstruct

VMstruct

Different fields in VMstruct data structure are

1. VM_ID - final String
2. cap - integer

This is the structure used by PM modifier to create a VM.

1.1.3 Interface Functions

void addVM(cap)
Description : This function checks the PM's if there is enough capacity available and if available adds the VM to it. If there is no enough capacity it returns an error.
Input parameters : The cap of VM which is to be added. The ID for VM is automatically generated by the function.
Output parameters : NONE.
Return Values : If sufficient capacity to add a VM is not available it returns **No enough capacity** error message
Pseudocode :

```
void addVM(cap)
for each PMstruct in PMarray do
    if $res\_cap \leq 1$ cap then
        create an ID for this VM
        add VM to this PM
    end if
end for
```

void deleteVM(VM_ID)
Description : The purpose of this function is to delete the VM which is passed as an input parameter to while calling this function.
Input parameters : The VM_ID of VM which has to be deleted.
Output parameters : NONE.
Return Values : None, because all the error conditions that may arise are handled by data validation in user interface.
Pseudocode :

```
void deleteVM(VM_ID)
for each PMstruct in PMarray do
    for each VMstruct in VMarray do
        if VM_ID matches then
            delete this VM
        end if
    end for
end for
```

void switchOffPM(PM_ID)
Description : This function switches off the specified PM.
Input parameters : PM ID of the PM which has to be switched off.
Output parameters : NONE.

Return Values : Returns error if the VM's in the current PM can't be consolidated in to other PM's.
Pseudocode :

```
void switchOffPM($PM\_ID$)
for each $PMstruct$ in $PMarray$ do
  if $PM\_ID$ matches then
    change onState to OFF
  end if
end for
```

void switchOnPM(PM_ID)
Description : This function switches on the specified PM.
Input parameters : PM ID of the PM which has to be switched on.
Output parameters : NONE.
Return Values : No possible error condition.
Pseudocode :

```
void switchOnPM($PM\_ID$)
for each $PMstruct$ in $PMarray$ do
  if $PM\_ID$ matches then
    change onState to ON
  end if
end for
```

void consolidate()
Description : The function runs the consolidation algorithm to consolidate VM's in PM's and swithces off the PM's if any of the PM's become empty after consolidation.
Input parameters : NONE.
Output parameters : NONE.
Return Values : No possible error conditions
Pseudocode :

```
void consolidate()
quicksort($PMarray,lo,hi$) {sorts PMs in decreasing order of their residual capacity}
for $i$ from $0$ to $PMarray.lenght-1$ do
  quicksort($PMarray[i].VMarray, lo, hi$) {sorts VMs in decreasing order of their capacity}
  for each $VMstruct$ in $VMarray$ do
    for $i$ from $PMarray.lenght-1$ to $0$ do
      if $PMarray[i].PMstruct.res\_cap \geq VMarray.VMstruct.cap$ then
        move VMstruct into this PMstruct's VMarray
      end if
    end for
  end for
end for
```

Method used to sort PM's and VM's.

```
void quicksort(PMarray,lo,hi)
if $lo < hi$ then
  p = pivot(PMarray,lo,hi)
  left, right = partition(PMarray, p, lo, hi)
  quicksort(PMarray, lo, left)
  quicksort(PMarray, right, hi)
end if
```

```
int partition(PMarray, left, right, pivotIndex)
pivotValue = PMarray[pivotIndex]
swap PMarray[pivotIndex] and PMarray[right]
storeIndex = left
for i from left to right - 1 do
  if PMarray[i] $\leq 1$ pivotValue then
    swap PMarray[i] and PMarray[storeIndex]
    storeIndex = storeIndex + 1
  end if
  swap PMarray[storeIndex] and PMarray[right]
end for
return storeIndex
```