High Level Design Document
Bin-packing VM Consolidation Algorithm

Atchutuni Bhavana 13MCMT01
Surineni Sampath Kumar 13MCMT49
Terli Venkatesh 13MCMT55

Contents

1 Detailed Design

1.1 Parser Module

This module is invoked by the GUI module to read the data from the input file given by the user and pass it to the PM-modifier.The format of the input file is specified below.

- VM_ID—Integer
- VM_Capacity—Integer

1.1.1 Interface Data Structure

NONE

1.1.2 Internal Data Structure

NONE

1.1.3 Interface Functions

- int parse(filename)

Description : This function opens the given file.Each row in the input file should be in this format [VM_ID, Capacity]. Parser reads each row values and calls addVM(VM_ID,capacity) function in the PM-modifier module.

Input Parameters : The path of the file given by the user from the GUI module, complete path should be specified.

Output Parameters :VM_ID,capacity.

Return values : Respective error numbers will be sent for empty file and wrong file format.

Pseudocode :

parse(filename)

1: open the input file which was given by GUI.
2: **if** $fp == NULL$ **then**
3: return the error you entered nonexisting filename.
4: **end if**
5: check whether file is empty or not.
6: **if** no data in file **then**
7: return the error empty file.
8: **else**
9: Read the each line and send that value to addVM(VM_ID, capacity) in PM-modifier module
10: this step will repeat untill we we read all the lines.
11: **end if**

1.2 User Interface

User interface is useful for taking input from user and giving results to the user. First this module asks user to load an input file and passes it to parser module. After initialization of the PMs and VMs by the PM modifier module it displays a screen with buttons specifying the PMs and labels specifying their respective VMs.
It also has buttons to specify functionalities of the the system such as addVM,deleteVM and consolidate.

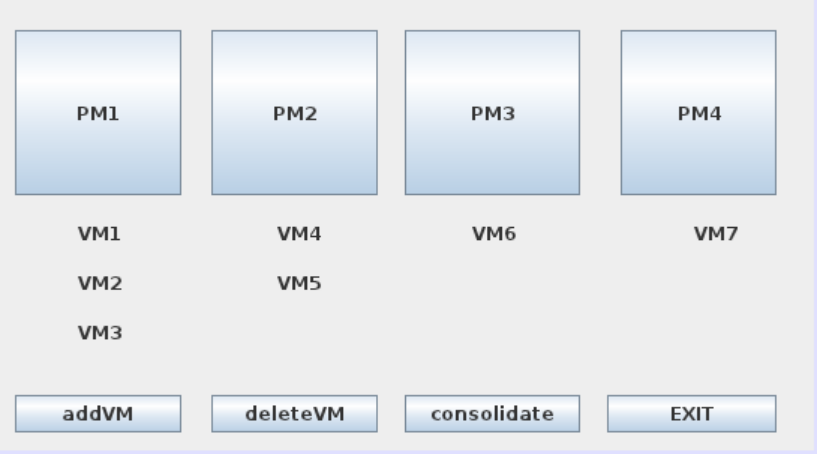

Figure 1: User interface

1.2.1 Internal Functions

callParser()

Initially user interface has a load button.On clicking, it opens a file chooser which helps the user to choose an input file.
The selected input filename is passed to the parser module.

Input parameters	: Click event
Output parameters	: Filename
Returns	: UI is updated on success. If the file chosen is not in the specified format an error message is returned.

Pseudo code :

```
1: on clicking load
2: open file chooser
3: if file choosen then
4:    call parser(filename) in parser module
5:    if error is returned then
6:       display the error message
7:    end if
8:    call updateUI()
9: else
10:   no change
11: end if
```

Figure 2: Initial GUI

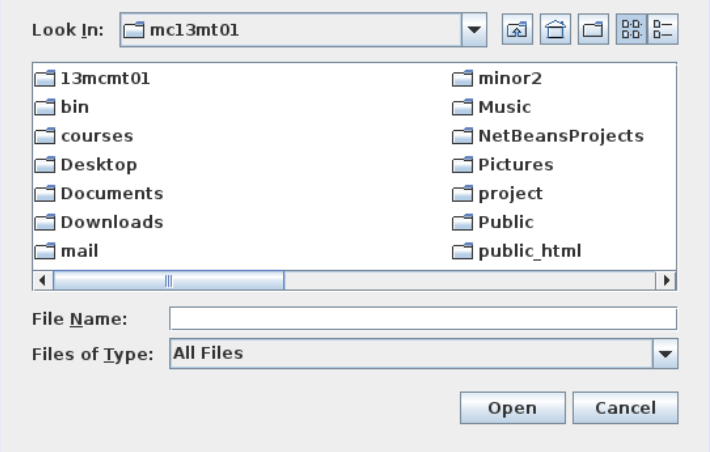

Figure 3: File chooser

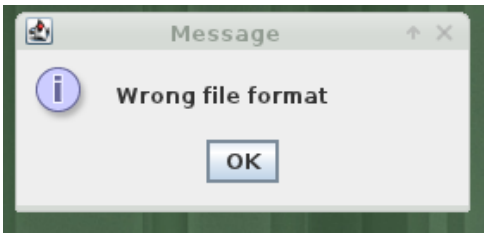

Figure 4: Error message

callAddVM()

On click addVM button,a dialog asking the user to enter VM_ID and capacity opens. Once the user specifies VM_ID and its capacity it calls the addVM() in PM modifier module and then calls updateUI().

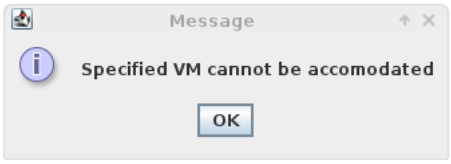

Figure 5: Error message

Input parameters	: Click event
Output parameters	: VM_ID, capacity
Returns	: UI is updated on success. Error message is returned if the specified VM cannot be accommodated by any of the PMs.
Error condition	: Entering negative numbers,alphanumerics,special characters, numbers greater than physical machine capacity can be avoided by data validation.

Pseudo code :

```
1: on clicking addVM
2: open dialog asking for VM_ID and capacity
3: validate input
4: if valid then
5:     call addVM(VM_ID,capacity) in PM modifier
6:     call updateUI()
7: else
8:     show error message
9: end if
```

callDeleteVM()

On clicking the deleteVM button in user interface, it opens a dialog which prompts the user to choose a VM for deletion. This calls the deleteVM() in PM modifier module. Later calls updateUI().

Input parameters	: Click event
Output parameter	: VM_ID
Returns	: UI is updated on success.
Error conditions	: Choosing an incorrect VM_ID. But this case is avoided by listing all the VMs present in the system.

```
1: on clicking deleteVM
2: open dialog displaying list of VMs, asking user to choose VM for deletion
3: if choosen then
4:     call deleteVM(VM_ID)
5:     call updateUI()
6: else
7:     do nothing
8: end if
```

Figure 6: VM Deletion

callConsolidate()

On clicking consolidate button in UI it opens a dialog for confirmation from the user
If the user chooses not to consolidate the state of the system does not change.
If the user chooses to consolidate , then conolidate() funtion in PM modifier is called.
The user interface is updated using the updateUI() function.

Input parameters	: Click event
Output parameters	: None
Returns	: UI is updated and the color of the physical machines that are switched off changes.
Error condition	: None

Pseudo code :

```
on clicking consolidate
open dialog for confirmation
if confirmed then
    call consolidate()
    call updateUI()
else
    do nothing
end if
```

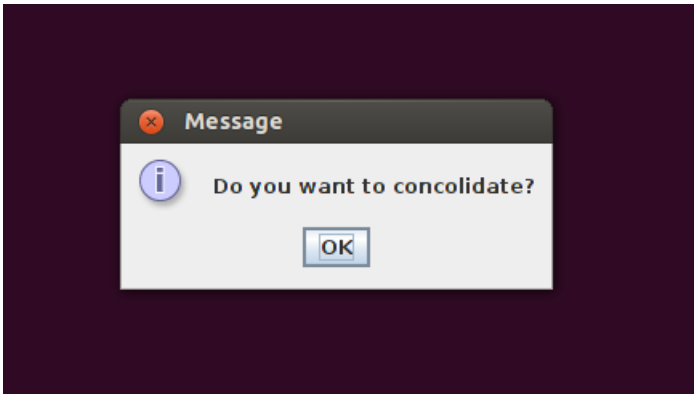

Figure 7: Consolidation

updateUI()

This function is called after every change performed on the system to reflect changes to the user.It calls the status() function in PM modifier module.

Input parameters	: None
Output parameters	: None
Returns	: Updated UI
Error conditions	: None

Pseudo code :

1: call status()
2: accordingly modify PMs and VMs

onPM()

This function is used to switch on a PM manually by the user.When user clicks a PM which is in off state,changes its state to ON by calling the switchOnPM() function in PM modifier.Then calls updateUI(). If user chooses a PM that is already in on state ,a message is prompted to user if he/she wishes to turn off respective PM

Input parameters	: Click event
Output parameters	: PM_ID
Returns	: UI is updated on success.The color of the PM that is turned on changes.On failure returns an error message.

Pseudo code :

```
1: on clicking PM
2: if PM in OFF state then
3:    OPEN DIALOG FOR CONFIRMATION
4:    if confirmed then
5:       call switchOnPM() in PM modifier
6:    else
7:       no change
8:    end if
9: end if
```

offPM()

This function is used to switch off a PM manually by the user.When user clicks a PM which is turned on,changes its state to OFF by calling switchOffPM() function in PM modifier.

Input parameters	: Click event
Output parameters	: PM_ID
Returns	: UI is updated on success.The color of the PM that is turned off changes.On failure returns an error message.
Error conditions	: If the VMs in selected PM cannot be accommodated in another PMs.

Pseudo code :

```
1: on clicking PM
2: if PM is in ON state then
3:    open dialog for confirmation
4:    if confirmed then
5:       call switchOffPM()
6:       if no error then
7:          call updateUI()
8:       else
9:          show error message
10:      end if
11:   else
12:      no change
13:   end if
14: end if
```

1.3 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

1.3.1 Interface Data Structures

1. PMstruct

PMstruct

Different fields in PMstruct data structure are

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

We maintain a linked list of this Data Structure to store the information about PM's. This the data structure returned to status() function which is called by User Interface

1.3.2 Internal Data Structures

1. VMstruct

VMstruct

Different fields in VMstruct data structure are

1. VM_ID - String
2. cap - integer

This is the structure used by PM modifier to create a VM. We maintain array of VM's for each PM.

1.3.3 Interface Functions

void addVM(VM_ID, 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 :

```
1: void addVM(VM_ID, cap)
2: for each PMstruct in PMlist do
3:    if res_cap ≤ cap then
4:       create an ID for this VM
5:       add VM to this PM
6:    end if
7: 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 :

```
1: void deleteVM(VM_ID)
2: for each PMstruct in PMlist do
3:    for each VMstruct in VMarray do
4:       if VM_ID matches then
5:          delete this VM
6:       end if
7:    end for
8: 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 :

```
1: void switchOffPM(PM_ID)
2: for each PMstruct in PMlist do
3:    if PM_ID matches then
4:       change onState to OFF
5:    end if
6: 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 :

```
1: void switchOnPM(PM_ID)
2: for each PMstruct in PMlist do
3:    if PM_ID matches then
4:       change onState to ON
5:    end if
6: 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 :

```
1: void consolidate()
2: quicksort(PMlist,lo,hi) {sorts PMs in decreasing order of their residual
   capacity}
3: for i from 0 to PMlist.lenght-1 do
4:    quicksort(PMlist[i].VMarray, lo, hi) {sorts VMs in decreasing order
      of their capacity}
```

```
    for each VMstruct in VMarray do
      for i from PMlist.lenght-1 to 0 do
        if PMlist[i].PMstruct.res_cap ≥ 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(PMlist,lo,hi)
if lo < hi then
    p = pivot(PMlist,lo,hi)
    left, right = partition(PMlist, p, lo, hi)
    quicksort(PMlist, lo, left)
    quicksort(PMlist, right, hi)
end if
```

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

PMstruct status()
Description : This function returns the number of PM's, thier On/Off status and VM's in them.
Input parameters : NONE.
Output parameters : Head of the list of type PMstruct.
Return Values : No possible error condition.
Pseudocode :

```
PMstruct status( )
```

2: return head of list *PMstruct*