

High Level Design Document  
Bin-packing VM Consolidation Algorithm

Surineni Sampath Kumar 13MCMT49

# Contents

<b>1</b>	<b>Detailed Design</b>	<b>3</b>
1.1	PM Modifier Module . . . . .	3
1.1.1	Interface Data Structures . . . . .	3
1.1.2	Internal Data Structures . . . . .	3
1.1.3	Interface Functions . . . . .	4

# 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, VM\_ID)**

**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(cap)
2: for each PMstruct in PMarray do
3:   if  $res\_cap \leq 1\ cap$  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 PMarray 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 PMarray 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 PMarray 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 switches 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(PMarray, lo, hi) {sorts PMs in decreasing order of their residual
   capacity}
3: for i from 0 to PMarray.lenght-1 do
4:   quicksort(PMarray[i].VMarray, lo, hi) {sorts VMs in decreasing order
   of their capacity}
5:   for each VMstruct in VMarray do
6:     for i from PMarray.lenght-1 to 0 do
7:       if PMarray[i].PMstruct.res_cap  $\geq$  VMarray.VMstruct.cap then
8:         move VMstruct into this PMstruct's VMarray
```

```

9:         end if
10:    end for
11: end for
12: end for

```

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

```

1: void quicksort(PMarray, lo, hi)
2: if lo < hi then
3:     p = pivot(PMarray, lo, hi)
4:     left, right = partition(PMarray, p, lo, hi)
5:     quicksort(PMarray, lo, left)
6:     quicksort(PMarray, right, hi)
7: end if

1: int partition(PMarray, left, right, pivotIndex)
2: pivotValue = PMarray[pivotIndex]
3: swap PMarray[pivotIndex] and PMarray[right]
4: storeIndex = left
5: for i from left to right - 1 do
6:     if PMarray[i] ≤ pivotValue then
7:         swap PMarray[i] and PMarray[storeIndex]
8:         storeIndex = storeIndex + 1
9:     end if
10: swap PMarray[storeIndex] and PMarray[right]
11: end for
12: return storeIndex

```