3. HETEROGENEITY AWARE VIRTUAL MACHINE PLACEMENT AND CONSOLIDATION

3.1 OUTLINE

Cloud systems are typically heterogeneous with generations of servers of different configurations owing to factors such as specific workload processing requirement, technology changes, price, hardware component failures etc. From a cloud service provider’s perspective, it is crucial to process workloads, provision and schedule VMs in a manner that exploits the heterogeneity of resources to improve energy efficiency of the datacenter. The proposed energy efficient VM allocation and consolidation approach considers server’s performance and power consumption aspect to reduce overall datacenter energy consumption and within acceptable performance.

Reducing datacenter energy consumption could be achieved by switching servers to off state immediately when not processing workloads (using less number of servers). But, this could impact application’s performance if subsequent workload processing requires additional servers to be on, as switching a server from off to on involves transition time. Thus, the proposed approach to reduce datacenter energy consumption considers a breakeven wait time before a server can be switched off to account for this power-performance trade-off.

The proposed approach a) increases the usage of servers with high energy efficiency from amongst available heterogeneous servers in the DC, b) reduces the number of active or on state servers using server consolidation, and c) uses dynamic power management server state transitions to transition the server between on or off states.

The contributions of this Chapter are the following:

- Metrics to classify servers in a heterogeneous cloud environment based on server’s power consumption and performance characteristics
- Approach for server and VM placement
• heuristics for the problem of energy and performance efficient dynamic VM consolidation
• An extensive simulation-based evaluation and performance analysis of the proposed algorithms.

3.2 SOLUTION APPROACH

3.2.1 System model and problem description

Table. 3.1: Notations: System model

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RK</td>
<td>Total number of Racks</td>
</tr>
<tr>
<td>N</td>
<td>Total number of servers</td>
</tr>
<tr>
<td>RS</td>
<td>Number of servers per rack</td>
</tr>
<tr>
<td>M</td>
<td>Number of Virtual machines (VMs)</td>
</tr>
<tr>
<td>ST</td>
<td>Total number of Server type configurations</td>
</tr>
<tr>
<td>server&lt;sup&gt;i&lt;/sup&gt;</td>
<td>i&lt;sup&gt;th&lt;/sup&gt; Server ; 0 &gt; i ≥ N</td>
</tr>
<tr>
<td>vm&lt;sup&gt;j&lt;/sup&gt;</td>
<td>j&lt;sup&gt;th&lt;/sup&gt; VM ; 0 &gt; j ≥ M</td>
</tr>
<tr>
<td>serverType&lt;sup&gt;k&lt;/sup&gt;</td>
<td>k&lt;sup&gt;th&lt;/sup&gt; ServerType; 0 &gt; k ≥ ST</td>
</tr>
<tr>
<td>rack&lt;sup&gt;j&lt;/sup&gt;</td>
<td>j&lt;sup&gt;th&lt;/sup&gt; Rack; 0 &gt; j ≥ RK</td>
</tr>
<tr>
<td>serverTypeMap&lt;sup&gt;(i, k)&lt;/sup&gt;</td>
<td>Bivalent variable that mentions that server&lt;sup&gt;i&lt;/sup&gt; is of serverType&lt;sup&gt;k&lt;/sup&gt; map;</td>
</tr>
<tr>
<td>rackMap&lt;sup&gt;(i, j)&lt;/sup&gt;</td>
<td>Bivalent variable that mentions that server&lt;sup&gt;i&lt;/sup&gt; is placed in rack&lt;sup&gt;j&lt;/sup&gt; map;</td>
</tr>
</tbody>
</table>

This Chapter considers IaaS cloud environment comprising of datacenter with N physical servers. Notations used in the system model are captured in Table 3.1. Configuration of each of these physical servers is from amongst a set of ST server types. Each server type represents a server’s configuration and is characterized by CPU performance defined in million instructions per second (MIPS), RAM capacity, network bandwidth and common NAS storage to enable live migration of virtual machines (VMs).
Servers are placed geographically in one of the RN racks in the datacenter. Each rack can accommodate RS number of Servers. The system has no knowledge of application workloads and time for which these applications are hosted i.e., the system is application-agnostic.

Users submit requests for provisioning of M VMs. Configuration of each of these VMs is from amongst a set of VT VM types. Each VM type is characterized by processing performance defined in MIPS, RAM capacity, storage capacity and network bandwidth. Since, it is an IaaS environment, the cloud service provider would have no knowledge of application and workloads, number of VMs required and the VMs required duration. The fact that the VMs are managed by independent users implies that the resulting workload created due to combining multiple VMs on a single physical node is mixed. The mixed workload is formed by various types of user specific applications which could utilize the resources simultaneously [12]. Each instruction executed by the applications running in the VMs is assumed to consume 1 cycle per second, which implies that 1 MIPS is equal to 1 MHz in processor clock speed. Typically, the users establish SLAs with the cloud service provider to formalize the Quality of Service (QoS) delivered. The cloud service provider pays a penalty to the users in cases of SLA violations.
The proposed system architecture (Figure. 3.1) comprises of the following functional modules:

- **Monitor module**: This module continuously monitors and tracks server and VM utilizations, servers’ current temperature and run-time power state transitions using host and rack managers. This function is initiated at regular epoch intervals.

- **Provisioner module**: Provisioner module assigns VMs to the best energy efficient server in the datacenter based on power, performance, thermal and resource availability characteristics. This is a superset module that contains the logic to find the best energy efficient server (uses Heterogeneity controller module) that satisfies the VM resource requirement and the server’s temperature is within temperature threshold criteria. Additionally, this logic is used by Optimizer module during server consolidation process as-well. The function of this
module is discussed in detail in Section 3.2.7. Provisioner module is initiated at the start of the simulation run.

Table 3.2: Server type configurations [73]

<table>
<thead>
<tr>
<th>Server Type K</th>
<th>Server Configuration</th>
<th>CPU Frequency (MHz)</th>
<th>Cores #</th>
<th>Memory (GB)</th>
<th>Peak Power (Watts)</th>
<th>Performance to peak Power ratio (Mhz/Watts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HP ProLiant ML110 G3 (Pentium D930)</td>
<td>3000</td>
<td>2</td>
<td>4</td>
<td>169</td>
<td>35.50</td>
</tr>
<tr>
<td>2</td>
<td>HP ProLiant ML110 G4 (Xeon 3040)</td>
<td>1860</td>
<td>2</td>
<td>4</td>
<td>117</td>
<td>31.79</td>
</tr>
<tr>
<td>3</td>
<td>HP ProLiant ML110 G5 (Xeon 3075)</td>
<td>2660</td>
<td>2</td>
<td>4</td>
<td>135</td>
<td>39.40</td>
</tr>
<tr>
<td>4</td>
<td>IBM server (Xeon X3470)</td>
<td>2933</td>
<td>4</td>
<td>8</td>
<td>113</td>
<td>103.82</td>
</tr>
<tr>
<td>5</td>
<td>IBM server (Xeon X3480)</td>
<td>3067</td>
<td>4</td>
<td>8</td>
<td>113</td>
<td>108.57</td>
</tr>
<tr>
<td>6</td>
<td>IBM server (Xeon X5670)</td>
<td>2933</td>
<td>6</td>
<td>12</td>
<td>247</td>
<td>71.24</td>
</tr>
<tr>
<td>7</td>
<td>IBM server (Xeon X5675)</td>
<td>3067</td>
<td>6</td>
<td>16</td>
<td>222</td>
<td>82.89</td>
</tr>
</tbody>
</table>

- **Heterogeneity controller module:** This module categorizes servers in terms of their performance and power characteristics and is obtained from SPEC data (Table 3.2). Servers of a given type are modeled by their processing capacity, thermal as well as their power consumption. Also, servers have unique state transition power overhead and transition time latency characteristics. Server’s energy efficiency is driven by both performance and power consumption. This categorization using StaticPPMMMax metric (Equation 3.4) and similar metrics StaticPPMMIn, DynamicPPM etc., helps in selecting the best energy efficient server with some host allocation policies being used in provisioner and optimizer modules.

- **Optimizer module:** Goal of this module is to improve utilization of servers and reduce the number of switched on servers in the datacenter, thereby improving energy efficiency of the datacenter. It also performs VM re-sizing due to load change and accounts for this re-
sizing into server resource capacities. The function of this module is discussed in Sections 3.2.5, 3.2.6 and 3.2.7. This function is initiated at regular epoch intervals to:

- consolidate VMs by identifying the servers which are over-loaded or under-loaded
- identify VMs which need to be migrated to another server so as to alleviate over load or over temperature scenarios, and enable switching off servers in case of under-load scenarios
- identify servers which can accommodate these identified VMs for re-assignment and
- migrate these identified VMs to the identified servers

Network bandwidth and server’s primary memory details are considered to arrive at the migration time required by the VM migration process to migrate a VM from one server to another. Other network communication aspects have not been accounted for in this work.

- **Server DPM power state transitioner module**: Main goal of this module is to transition servers between power states depending on the following scenarios: a) downgrade scenario deals with selection of an active or on server (which) and the time (when) it can be transitioned to off state. Downgrade scenario is initiated at regular intervals; b) upgrade scenario deals with selection of server (which) and the time (when) it can be transitioned from off state to on state. Upgrade scenario is initiated when VM placement approach cannot identify a target server in on state for the VM.

Server power states transitioning operations like server switching off or switching on is managed by each server’s host manager. The function of this module is discussed in Sections 3.2.2 and 3.2.8. In this work, frequency scaling or DVFS controls has not been considered to control energy consumption. Also, server fan power consumption impacts have not been considered into the power model.
• **Thermal coverage controller module:** Main goal of this module is to:
  - dynamically compute server’s current temperature $\text{Temperature}^i_{\text{current}}$ and
  - dynamically compute server's coverage flag $(\text{SCoverage}^i)$

Server temperature $\text{Temperature}^i_{\text{current}}$ and $\text{SCoverage}^i$ is managed by each server's Host Manager (HM). Detailed discussion of the functions of this module is presented in Section 3.2.3. This module is initiated at regular epoch intervals.

In this Chapter, the focus is on research challenges with respect to energy-efficient VM scheduling, dynamic VM consolidation, dynamic server provisioning using server power state thereby reducing datacenter energy consumption. List of problems addressed in this chapter are as follows:

• Typical datacenter consists of multitude of server models or configurations catering to different user or application and performance needs. So, a good solution design should account for this heterogeneity aspect to meet user the needs but at reduced energy cost. Server heterogeneity influences both performance and power consumption. Some servers exhibit good performance characteristics at higher power consumption characteristics. Some servers comparatively have better performance, and consume less power and so on. An approach to rank servers based on energy efficiency with respect to heterogeneity is proposed in this work thereby accounting for both performance requirements and optimal datacenter energy consumption.

• With virtualization, a server can be configured to host multitude of VMs, each of these VMs, which from application or users perspective behaves like a physical server. Operationally, virtualization technology helps scale up and down resource capacities to virtual machines and also helps consolidate servers by selective migration of VMs between physical servers. This consolidation approach helps reduce number of physical servers by either switching off some of the un-used physical servers, thus reduce datacenter overall power consumption. An approach has been outlined that specifies when VMs should be selected for migration,
identify which VM to migrate, and to which target server should the VM be migrated.

- Server processors have various levels of power state operations that control server power consumption. In many of the current approaches, sleep state transition overheads (transition time and power consumption overhead) are not considered in virtualized server cluster environment, and even if considered are limited to single server. A combined dynamic server provisioning and VM consolidation approach has been proposed to optimize energy consumption by the system and avoid violations of the QoS requirements.

### 3.2.2 Server power and state transition model

Power consumption by servers can be described by a relationship between the power consumption and CPU utilization [18]. Server Power consumption is modeled from SPECpower benchmark’s [73] real data values using linear interpolation. Seven server configurations have been selected from the SPECpower benchmarks, and these server configurations are referred to as server types. The reason to have chosen these server types is to simulate effect of VM consolidation with server types with varied range of performance and power characteristics i.e., servers of with both similar and dissimilar power and performance characteristics which is a reality in a datacenter with heterogeneous server configurations. The configuration and power consumption characteristics [73] of the selected servers are shown in Tables 3.2, 3.4 and Figure 3.2. The power and time model related notations is tracked in Table 3.3. Table 3.4 captures the server state transition latency times in transitioning from one power state to another in servers of different server types. The model comprises of the following server power states viz., on, off and SETUP states, and the following power state transition time latency to transition from one power state to another viz., off to on, and on to off (Time_{off→on}^{k} and Time_{on→off}^{k}) SETUP latency times. At a high level, power consumption at different states and transition times between states follows the comparison [34] shown in Equation (3.1).
\[ \text{Power}^{k_{100\%}} > \text{Power}^{k_{0\%}} >> \text{Power}^{k_{\text{off}}} = 0 \text{ Watts} \]  

(3.1)

**Table. 3.3: Notations: Power, state transition times and thermal model details**

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Power}^{k_{\text{off}}} )</td>
<td>Power consumption (in Watts) of server of serverType ( k ) at off state = 0 Watts; ( 0 &gt; k \geq ST );</td>
</tr>
<tr>
<td>( \text{Power}^{k_{\text{util%}}} )</td>
<td>Power consumption (in Watts) of server of serverType ( k ) at active on state when operating at CPU util% without factoring power consumption required for cooling; ( 0 &gt; k \geq ST ); ( 0% \geq \text{util%} \geq 100% );</td>
</tr>
<tr>
<td>( \text{Power}^{k_{\text{SETUP}}} )</td>
<td>Power consumption (in Watts) of server of serverType ( k ) during transitioning from one power state to another without factoring power consumption required for cooling; ( 0 &gt; k \geq ST );</td>
</tr>
<tr>
<td>( \text{Time}^{k_{\text{off-\text{on}}}} )</td>
<td>Transition time (in Seconds) of server of serverType ( k ) to transition from off state to on state; ( 0 &gt; k \geq ST );</td>
</tr>
<tr>
<td>( \text{Time}^{k_{\text{on-\text{off}}}} )</td>
<td>Transition time (in Seconds) of server of serverType ( k ) to transition from active state to off state; ( 0 &gt; k \geq ST );</td>
</tr>
<tr>
<td>( \text{Time}^{i_{\text{SETUP}}} )</td>
<td>Total time (in Seconds) of server ( i ) is in transitioning mode; ( 0 &gt; i \geq N );</td>
</tr>
<tr>
<td>( \text{Time}^{i_{\text{off}}} )</td>
<td>Total time (in Seconds) the server ( i ) is in off state; ( 0 &gt; i \geq N );</td>
</tr>
<tr>
<td>( \text{Time}^{i_{\text{util%}}} )</td>
<td>Total time (in Seconds) the server ( i ) is active on state and at utilization util%; ( 0 &gt; i \geq N ); ( 0% \geq \text{util%} \geq 100% );</td>
</tr>
<tr>
<td>( \text{Temperature}_{\text{threshold}} )</td>
<td>Upper threshold temperature (in Kelvin) of servers in the datacenter (also can be referred as critical or safe temperature);</td>
</tr>
<tr>
<td>( \text{Temperature}^{i_{\text{current}}} )</td>
<td>Current Temperature (in Kelvin) due to heat dissipated in processing workloads by server ( i ) processor @ Time(_{\text{current}}^i); ( 0 &gt; i \geq N );</td>
</tr>
<tr>
<td>( \text{Temperature}^{i_{\text{prev}}} )</td>
<td>Temperature (in Kelvin) due to heat dissipated in processing workloads by server ( i ) processor in the previous server temperature calculation time @ Time(_{\text{prev}}^i); ( 0 &gt; i \geq N );</td>
</tr>
<tr>
<td>( \text{Temperature}_{\text{supply}} )</td>
<td>Computer room air conditioner supply temperature; default value is set at 293 Kelvin or 20 Celsius;</td>
</tr>
<tr>
<td>( \text{Temperature}^{i_{\text{hrc}}} )</td>
<td>Temperature (in Kelvins) due to heat re-circulation (from self and other servers in the spatial thermal vicinity) into server ( i ) processor; ( 0 &gt; i \geq N );</td>
</tr>
<tr>
<td>( \text{Temperature}^{i_{\text{inlet}}} )</td>
<td>Inlet temperature (in Kelvin) of server ( i ); ( 0 &gt; i \geq N );</td>
</tr>
<tr>
<td>( \text{Time}^{i_{\text{prev}}} )</td>
<td>Time when “temperature dissipation routine” was initiated in server ( i ) previously; ( 0 &gt; i \geq N );</td>
</tr>
<tr>
<td>( \text{Time}_{\text{current}} )</td>
<td>Current clock time “temperature dissipation routine” is run;</td>
</tr>
<tr>
<td>( \text{SCoverage}^{i} )</td>
<td>Temperature coverage flag for server ( i ); ( 0 &gt; i \geq N );</td>
</tr>
<tr>
<td>Notation</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SCoverage(i)</td>
<td>( \begin{cases} \text{true, if server}^i \text{ temperature is within threshold} &amp; \ \text{false, otherwise} \end{cases} )</td>
</tr>
<tr>
<td>(C^i)</td>
<td>Capacitance ((\text{in Joules/Kelvin})) of Server(i); (0 &gt; i \geq N);</td>
</tr>
<tr>
<td>(R^i)</td>
<td>Resistance ((\text{in Kelvin/Watt})) of Server(i); (0 &gt; i \geq N);</td>
</tr>
</tbody>
</table>
| COP(\(Temperature_{\text{supply}}\)) | Coefficient of cooling performance = 
\(0.00068 \times Temperature_{\text{supply}} \times Temperature_{\text{supply}} + 0.0008 \times Temperature_{\text{supply}} + 0.458 [65]\)  
Temperature is in Celsius                                                      |
| State_Status\(i\) | Current power State status of the server\(i\) \((\text{possible states: on, off, SETUP})\)  
SETUP states are on to off and off to on                                       |
| Power\(i^k\)  | Instantaneous power consumed by server\(i\) \((\text{in Watts})\)  
if \((\text{State\_Status}\(i\) == on) Return Power\(i^k_{\text{util%}}\)  
\times \text{serverTypeMap}(i, k)  
if \((\text{State\_Status}\(i\) == SETUP) Return Power\(i^k_{\text{SETUP}}\)  
\times \text{serverTypeMap}(i, k)  
if \((\text{State\_Status}\(i\) == off) Return 0  
Where \(\text{serverTypeMap}(i, k) = 1; \forall k\)                                           |
| \(E^k\)       | Total Energy consumed by the server\(i\) of serverType\(k\); \((\text{in kWh})\)  
\(= \text{serverTypeMap}(i, k) \times \left( \sum_k \text{Power}_{\text{SETUP}}^k \times \text{Time}_{\text{SETUP}}^i \right) \)  
\(+ \sum_{k, \text{util%}} \text{Power}_{\text{util%}}^k \times \text{Time}_{\text{util%}}^i \)  
\(\times \left(1 + \frac{1}{\text{COP}(Temperature_{\text{supply}})}\right) \)  
\(\times \frac{1}{3600 \times 1000} \)  
Where \(\text{serverTypeMap}(i, k) = 1; \forall k\)                                           |
| \(E\)         | Total Energy consumed by datacenter \((\text{kWh})\)  
\(= \sum_i \sum_k E^k\) ; \(\forall i \geq N; 0 > k \geq \text{ST}\)                                                                                          |
In this model, physical servers are equipped with multi-core CPUs. Multi-core CPU is modeled to have \( n \) cores; with each core's capacity as \( m \) MIPS and total capacity of \( nm \) MIPS. This is justified as applications and VMs, are not tied down to processing cores and can be executed on an arbitrary core using a time-shared scheduling algorithm. The only limitation is that the CPU capacity required for a VM must be less than or equal to the available capacity of a single core.

### 3.2.3 Thermal coverage model

Increase in datacenter temperature over a threshold, increases the energy consumption required to cool or dissipate the generated heat. This increase in server temperature also negatively impacts the reliability of the server hardware. Hence it is a necessity to maintain datacenter temperature
to be within upper threshold bounds. Server’s temperature is due to the power consumption owing to workloads being processed by the VMs hosted in the server. When the server’s temperature exceeds the threshold-temperature (Equation 3.2), it implies that cooling coverage to the server is insufficient. Server with no cooling coverage or insufficient cooling coverage over a time period duration could create hot-spots, which impacts not just one server but other servers in its thermal reach or vicinity.

Function setServerCoolingCoverage()

\[
\text{SCoverage}^i \leftarrow \begin{cases} 
\text{true, if } \text{Temperature}^i_{\text{current}} < \text{Temperature}_{\text{threshold}} \\
\text{false, otherwise}
\end{cases} 
\]  

Server compute temperature owing to processing of the workload is modeled as a lumped RC thermal model [74] in this initial work due to its simplicity. This model is however limited to a single-core processor. This approach uses analogy between electrical circuit phenomena and a heat transfer phenomena. The heat flow between the internal CPU chip blocks are modeled by connecting thermal resistors and capacitance in blocks. Server power consumption is due to its underlying VMs and the applications running in the VM. Server processors’ temperature at a high level is a function of power consumed which is dissipated as heat. Each server’s current temperature is based upon server power consumption (Power^k_i), time duration (Time^i_{diff}) between current time (Time^i_{current}) and previous time (Time^i_{prev}), temperature (Temperature_{inlet}) of the datacenter; and resistance R^i and capacitance C^i of the server. Current server temperature is derived using the computation model (Equation 3.3) and server thermal parameter values [75] (Table 3.5) is used to set Server^i ‘s current temperature (Temperature^i_{current}).

Function setTcurr(server^i)

\[
\text{Temperature}^i_{\text{current}} = \text{Power}^k_i \times \text{ServerTypeMap}(i,k) \times R^i + \text{Temperature}^i_{\text{inlet}} - (\text{Power}^k_i \times \text{ServerTypeMap}(i,k) \times R^i + \text{Temperature}^i_{\text{inlet}} - \text{Temperature}^i_{\text{prev}}) \times e^{-\frac{\text{Time}^i_{\text{diff}}}{R^i \times C^i}} 
\]  

where ServerTypeMap(i,k) = 1; \forall k \in \{1..ST\} 
\text{Time}^i_{\text{diff}} = \text{Time}^i_{\text{current}} - \text{Time}^i_{\text{prev}}
Table 3.5: Thermal parameter values [75]

<table>
<thead>
<tr>
<th>Server thermal parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ambient Temperature</td>
<td>308 (in Kelvin)</td>
</tr>
<tr>
<td>Capacitance C of Server</td>
<td>340 (in Joules/Kelvin)</td>
</tr>
<tr>
<td>Resistance R of Server</td>
<td>0.34 (in Kelvins/Watt)</td>
</tr>
</tbody>
</table>

This proposed work additionally considers thermal controls to improve server resilience factor in VM allocation and server consolidation logic. This thermal control maintains the server temperature below threshold temperature [76]. Experimental evaluation of fixing the threshold temperature is discussed in Chapter 7. The VM provisioning and allocation logic selects a server for use, if and only if the server is operating below the threshold temperature. Consolidation logic also considers server temperature and migrates VMs from server which is above critical temperature to another server which would operate below critical temperature post migration process.

3.2.4 Approach steps

The target compute environment is an Infrastructure as a Service (IaaS) where the cloud service provider is unaware of applications and workloads served by the VMs, and can only observe them from outside.

![Figure 3.3: System process flow for VM placement and consolidation](image)
The proposed approach (Figure. 3.3) performs steps comprising of dynamic VM provisioning, VM consolidation, VM deconsolidation, and server DPM normal power state transitions. Detailed steps are captured in Figure 3.4.

- Initial step, places and allocates VMs on to specific active servers (Section 3.2.7)
- Post initial step, run-time optimization step performs VM consolidation and deconsolidation. This step identifies server’s that are under-loaded based on utilization, and also server’s that are over-loaded based on utilization and temperature threshold values (Section 3.2.5). This step also, identifies VMs that need to be migrated from overloaded server (Section 3.2.6). In case of under-loaded and over-temperature servers, all the VMs in these hosts are selected for migration. These VMs selected for migration are migrated or placed on different target active servers (Section 3.2.7).
- At regular intervals, DPM normal state transition step (Section 3.2.8) is performed with possible server states are: on, off and SETUP, and allowed server state transitions are: on to off and off to on.
3.2.5 Virtual machine consolidation approach

Dynamic VM optimization approach performs consolidation of servers in the datacenter to improve resource utilization and energy efficiency at regular epoch. In this work, CPU (from amongst other system resources, such as memory, disk, network bandwidth) is considered as the main resource which usually is oversubscribed by cloud service providers and are the critical cause for performance bottlenecks. Performance of server degrades, when the load approaches 100% [51] resulting in additional VMs requested capacities not being provisioned. Sever consolidation process involves the following steps: a) identify servers and time period when these servers are over utilized or above temperature threshold criteria value; b) identify the servers and time when these servers are under-utilized; c) determine the VMs that need to be migrated from servers identified in step a), so that post VM migration the server would be within threshold temperature and utilization limits. Also, select all VMs from servers identified in step b) for migration to another server; d) find the new server where-in the selected VMs from overloaded servers and all the VMs from under-loaded servers would need to be migrated; e) determine when and which servers need to be switched off; and f) determine which servers need to be switched on and when to switch on these servers due to increase in VM workload requirements.

The pseudocode for VM consolidation approach depicted in Algorithm 3.1 consists of two phases. In the first phase, over-utilized servers or servers operating above threshold temperature in the datacenter are identified. One or more VMs from these overloaded serves are selected using VM selection policy (Section 3.2.6) and tracked into vmToMigrateList. In case of servers which have been identified as operating above threshold temperature, all the VMs in these servers are tracked into vmToMigrateList. VM placement algorithm is applied with input as vmToMigrateList to arrive at the new vm to server allocation list. The second phase is to find under-loaded servers and the hosted VMs captured into vmToMigrateList as-well. VMs selected cannot be migrated to servers marked as overloaded or under-loaded or above temperature threshold for the epoch interval and VMs cannot be migrated to
more than one server for the epoch interval. VM placement algorithm (Section 3.2.7) is applied with input as vmToMigrateList to arrive at the new VM to server allocation list.

Algorithm. 3.1: VM consolidation optimization approach

**Input**: serverList  
**Output**: migrationMap  

//phase 1 – over-load or over-temperature server
1. foreach server in serverList do  
2. if (server.getStatus != on) then CONTINUE; // proceeds only if servers power state is on  
3. vmsToMigrateList.clear()  
   //checks if server is running above utilization threshold or current temperature is above temperature threshold  
4. if (server.isOverThresholdTemperature_Or_Utilization() ) then  
   //fetch one or more VMs from the server so as to bring the server within thermal and utilization limits  
5. vmsToMigrateList.add(server.getVMsToMigrateFromOverThresholdServer()); //other than migrating in or out VMs  
   //for each of the VMs fetched, get the new VM to server mapping  
6. migrationMap.add(getNewVmPlacement(vmsToMigrateList) )  
7. end if  
8. end foreach//serverList  
9. vmsToMigrateList.clear()  
//phase 2 – under-load server  
10. foreach server in serverList do  
11. if (server.getStatus != on) then CONTINUE; // proceeds only if servers power state is on  
12. if (server.isUnderLoaded() && !server.isVMMigrationProgress_In_or_Out) then  
13. vmsToMigrateList.add(server.getVMs()) ;  
14. //for each of the VMs fetched from vmsToMigrateList, get the VM to new server mapping  
15. migrationMap.add(getNewVmPlacement(vmsToMigrateList) )  
16. end if  
17. end foreach//serverList  
18. RETURN migrationMap

Function server.isOverThresholdTemperature_Or_Utilization()

**Input**: threshold_temp, threshold_upper_util, n // from reference data  
**Output**: Boolean: Whether the server is overloaded  
1: if Server is on && temperature > threshold_temp then  
2: RETURN true  
3: if utilization is not empty then  
   // server utilization for the last n epoch's  
4: utilizationM ← last n values of utilization  
5: meanUtilization ← sum(utilizationM) / n  
6: RETURN meanUtilization > threshold_upper_util  
7: RETURN false
Function server. isUnderLoaded()

Input  : threshold_temp, n // from reference data
Output: Whether the server is underloaded
1: if server is on && temperature <= threshold_temp then
3:  if server is overloaded then
4:    if utilization is not empty then
// server utilization for the last n epoch’s
5:      utilizationM ← last n values of utilization
6:      meanUtilization ← sum(utilizationM) / n
7:      if server is with the least meanutilization from amongst all
     on servers then
8:        RETURN true
9:      RETURN false

3.2.6 Virtual machine selection approach

After identifying the servers that are overloaded either due to above
threshold utilization usage or above threshold server temperature, VMs that
would need to be migrated are iteratively selected from these overloaded or
over temperature servers to a different target server. This target server would
need to satisfy the capacity and temperature criteria and post migration would
to reduce the load on the over-loaded servers. In this work, minimum migration
time (MMT) policy [11] is used to select the VM which would require the least
time to complete a migration relatively to other VMs allocated to the server.
Details of VM selection approach are captured in functions
getVMsToMigrateFromOverThresholdServer() and vmSelection().

Function server. getVMsToMigrateFromOverThresholdServer()

//How many VMs can be selected from the overloaded server for
migration
Input  : server which is overloaded or above threshold termperature
Output: VMs that can be migrated (VmSet)
1: while(true)
2:  vm ← server.vmSelection() //which VM to select
3:  if vm == NULL then BREAK
4:  host.Remove(vm) //test
5:  VmSet.add(vm)
6:  if !server.isOverThresholdTemperature_Or_Utilization() ||
     no more VMs in server then BREAK//till the server becomes normally
loaded
7:  end while
8:  RETURN VmSet
Function server. vmSelection()

Input : server
Output: A VM to migrate
1: minRam ← MAX_VAL; maxUtil ← 0; selectedVm ← NULL
2: foreach vm on this server
3: if vm is migrating out then CONTINUE;//ignore if vm is migrating
4: if vm.getRam() < minRam then //minimum Ram
5:          minRam ← vm.getRam()
6: if vm.getUtil() > maxUtil then //max utilization
7:          maxUtil ← vm.getUtil()
8: selectedVm ← vm
9: end foreach
10: RETURN selectedVm

3.2.7 Virtual machine placement approach

The VM placement can be seen as a bin packing problem with variable bin sizes and prices, where bins represent the physical servers, items are the VMs that have to be allocated, bin sizes are the available resource capacities of the servers and prices correspond to the power consumption or energy efficiency characteristics of the server [41]. Server type heterogeneity, server and datacenter thermal considerations open up different approaches to achieve energy efficiency. In this Chapter, following allocation policies and the corresponding VM placement approach algorithms have been used:

- Proposed approach (StaticPPMMax): Performance to power metrics using server’s peak power consumption

  The proposed new approach and metric referred to as StaticPPMMax is used to track energy efficiency coefficient value of server as follows:

  Performance to power static metric of server (StaticPPMMax)

  \[ \text{StaticPPMMax}_i = \frac{\text{Total MIPS of server}_i}{\text{Power}_{100\%} \times \text{serverTypeMap}(i,k) \times \left(1 + \frac{1}{\text{COP(Temperature_{supply})}}\right)} \]  

  where serverTypeMap(i, k) = 1 ; 0 > k ≥ SK; 0 > i ≥ N; ∀k

  The pseudocode for initial VM allocation into servers using the proposed approach (StaticPPMMax): metric is presented in Algorithm 3.2.
Algorithm 3.2: VM allocation approach

Input: servers, VMs (in vmList or vmsToMigrateList)
Output: vm to server allocation

1. Sort (vmList or vmToMigrateList) in descending order based on CPU utilization
2. Sort serverList in descending order based on PPM heuristics (Equation 3.4 or Equation 3.5)
3. foreach vm in vmList do
   4. allocatedServer ← NULL
   5. foreach server in serverList do
      6. if (server.getStatus() != on) then CONTINUE;
      7. if (!server.getCoolingCoverage()) then CONTINUE;
      8. if (!server.isSuitableForVm(vm)) then CONTINUE;
      9. if (isServerOverUtilizedAfterAllocation(server, vm) ||
           isServerOverTemperatureThresholdAfterAllocation(server, vm)) then
         10. CONTINUE
      else
         11. allocatedServer ← server
      12. BREAK
      13. end if
   14. end foreach//serverList loop
   15. if(allocatedServer == NULL) then
      16. initiate serverTransitionUpgrade()//switch on a server which is in off power state
   17. else
      18. allocationMap.put(vm, allocatedServer);//server found
   19. end if
   20. end if
21. end foreach//vmList loop
22. RETURN allocationMap

- **Beloglazov (PowerMin)**: Power aware best fit decreasing (PABFD)
  This approach is an existing work proposed by Beloglazov et al. [11], which selects the server with the minimum increment power to process the workload request and is considered as a baseline to compare and contrast power minimization gains using heterogeneous aware energy efficient approaches. This approach has been modified to account for server temperature control criteria.

- **Moreno (DynamicPPM)**:
  This approach is an existing work proposed by Ismael Moreno et al. in [41] as one of the baselines to compare and contrast power minimization gains using heterogeneous aware energy efficient approaches. This approach has been modified to account for server temperature control criteria.
• **Secondary baseline (StaticPPMMin)***:

StaticPPMMin is an approach similar to StaticPPMMax to track energy efficiency coefficient value of server as follows:

Performance to power static metric of server (StaticPPMMin)

\[
\text{Performance to power static metric of server } \text{ (StaticPPMMin)} = \frac{\text{Total MIPS of server}^*}{\text{Power}^* \times \text{serverTypeMap}(i,k) \times (1 + \frac{1}{\text{COP(Temperature}_{\text{supply}})})}
\]

where \(\text{serverTypeMap}(i, k) = 1; 0 > k \geq SK; 0 > i \geq N; \forall k\)

This approach uses server's idle power consumption in computing the Performance to power metrics.

• **Hybrid PPMMax with PowerMin**

Earlier discussed VM allocation approaches using StaticPPMMax and StaticPPMMin metrics is more suited for heterogeneous server type cloud environment in allocating the VM to the best energy efficient server. This proposed hybrid approach (Hybrid PPMMax with PowerMin) works well both in heterogeneous and homogeneous server type environments. The pseudocode for initial VM allocation into servers using this hybrid approach is presented in Algorithm 3.3.

• **Hybrid PPMMax with max utilization**

This proposed hybrid approach (Hybrid PPMMax with max utilization) considers server’s performance to power characteristics and server utilization. This approach works well both in heterogeneous and homogeneous server type environments. The pseudocode for initial VM allocation into servers using this hybrid approach is presented in Algorithm 3.4.
Algorithm 3.3: Hybrid PPMax with PowerMin approach

//initial VMs and its resource requirements or VMs that need to be migrated based on consolidation results

**Input**: serverList, serverTypeList, requested VMs in vmList, or vmToMigrateList

**Output**: vm to server allocation //allocationMap that tracks the vm to server allocation

1. Sort (vmList or vmToMigrateList) in descending order based on CPU utilization
2. Sort serverList in descending order based on StaticPPMax metric (Equation 3.4)
3. Sort serverTypeList in descending order based on StaticPPMax metric (Equation 3.4)

4. foreach vm in (vmList or vmToMigrateList) do
5.     foreach serverType in serverTypeList do
6.         allocatedServer ← NULL; minPower ← MAX;
7.         foreach server in serverList(serverType) do //fetch the servers which are of //the serverType value
8.             if (server.getStatus() != on) then CONTINUE; //proceeds only if servers power //state is on
9.             if (!server.getCoolingCoverage()) then CONTINUE; //depending on cooling //coverage strategy
10.            if (!server.isSuitableForVm(vm)) then CONTINUE;
11.            power ← estimatePower(vm, server) //server should not be over utilized or over temperature post allocation
12.            if ((getUtilizationOfCpuMips(server) !== 0 &&
                    (isServerOverUtilizedAfterAllocation(server, vm) ||
                    isServerOverTemperatureThresholdAfterAllocation(server, vm))))
                    then CONTINUE;
13.            else
14.                if (power < minPower) then
15.                    minPower ← power
16.                    allocatedServer ← server
17.                end if
18.            end if
19.         end foreach //serverList(ServerType) loop
20.     if (allocatedServer != NULL) then //found server for the vm
21.         BREAK
22. end if
23. end foreach //serverType loop
24. if (allocatedServer == NULL) then
25.     initiate serverTransitionUpgrade() //switch on a server which is either in off or //sleep power state
26. else allocationMap.put(vm, allocatedServer); //found server for the vm
27. end if
28. end foreach/vmList loop
29. RETURN allocationMap
Algorithm. 3.4: Hybrid PPMMax with maximum utilization approach

//initial VMs and its resource requirements or VMs that need to be migrated based on consolidation results
Input : serverList, serverTypeList, requested VMs in vmList, or vmToMigrateList
Output : vm to server allocation //allocationMap that tracks the vm to server allocation

1. Sort (vmList or vmToMigrateList) in descending order based on CPU utilization
2. Sort serverList in descending order based on StaticPPMMax metric (Equation 3.4)
3. Sort serverTypeList in descending order based on StaticPPMMax metric (Equation 3.4)
4. foreach vm in (vmList or vmToMigrateList) do

5. foreach serverType in serverTypeList do

6. allocatedServer ← NULL
7. serversOfServerTypeList = getServerList(serverType);//servers //tied to the serverType
8. Sort serversOfServerTypeList in descending order based on CPU utilization

// fetch the servers which are of the serverType value with CPU utilization in descending order
9. foreach server in (serversOfServerTypeList) do
10. if(server.getStatus() != on) then CONTINUE;//proceeds only if //servers power state is on
11. if (!server.getCoolingCoverage()) then CONTINUE;//depending on cooling coverage strategy
12. if (!server.isSuitableForVm(vm)) then CONTINUE;
13. if (!isServerOverUtilizedAfterAllocation(server, vm) || isServerOverTemperatureThresholdAfterAllocation(server, vm))) then CONTINUE;
14. allocatedServer ← server
15. BREAK
16. end foreach//serverList(ServerType) loop
17. if (allocatedServer != NULL) Then //found server for the vm
18. BREAK
19. end if
20. end foreach//serverType loop
21. if (allocatedServer == NULL) initiate serverTransitionUpgrade();//switch on a server which is either in //off or sleep power state
22. else allocationMap.put(vm, allocatedServer);//found server for the vm
23. end if
24. end foreach//vmList loop
25. RETURN allocationMap
3.2.8 Server dynamic power management approach

To improve datacenters’ power efficiency, servers are utilized effectively and when not in use are to be switched off. Focus is pertaining to a) upgrade scenario deals with server state transitions, specifically as to which off server to switch on and when to transition, and b) downgrade scenario deals with server state transitions, specifically as to which active or on server to switch off and when to transition.

The detailed server power state downgrade approach pseudocode (Algorithm 3.5) is used to identify a server (in active on power state), transition the servers’ power state from on to off state and when to do this transition. Power state transitioning of servers’ processor involves a transition time latency or time overhead, which is highly processor dependent and this could vary from one processor type to another (Table 3.4). The server stays in an interim power state called SETUP state for the period between each such transition and consumes power called SETUP power [70]. This downgrade process is initiated at regular epoch intervals. Also, a conservative downgrade process is adopted while transitioning server power state from sleep to off or on to sleep state transition operations by accounting for a wait-time. This time to wait (wait-time) retains the server in a higher power state as long as the time and power transition overhead cost is lesser than to transition to a lower power state and back to higher operational power state if workload happens to arrive during this time. The wait-time $\text{Time}_{\text{wait-on-off}}^k$ tracks the time delay (Equation 3.6) to stay in on state before the server of server type k can be transitioned to off state [15].

$$\text{Time}_{\text{wait-on-off}}^k = \frac{\text{Power}_{\text{setup}}^k \times \text{Time}_{\text{off-on}}^k}{\text{Power}_{0\%}^k}$$ (3.6)
Algorithm. 3.5: Server power state downgrade transition approach - serverTransitionDowngrade()

**Input** : serverList  
**Output** : server power state transition updates  
1. foreach server in serverList do  
2. if (server.getStatus() != on) then CONTINUE; //process only if server  
//power state is on  
3. if (getServerStatusCount( off-on) > 0) then RETURN; // there is need  
//for on servers so do not transition  
4. if (server.getVMs().size() == 0 && 
   server.getVmsMigratingIn().size() == 0 && 
   server.getVmsMigratingOut().size() == 0) then  
5. wait for Time_{wait-on-off} //wait-time  
6. server.setStatus(off); //this transition takes Time_{on-off}^k Seconds  
7. end if  
8. foreach end  

The proposed server state upgrade approach psuedocode (Algorithm 3.6) is used to identify the server that should be transitioned from off state to on state and when to do this transition.

Algorithm. 3.6: Server power state upgrade transition approach- serverTransitionUpgrade()

**Input** : serverList  
**Output** : server power state transition updates  
1. foreach server in serverList do  
2. if (getServerStatusCount( off-on) > 0) then RETURN  
3. if (server.getStatus() == off) then  
4. server.setStatus(on); //this transition takes Time_{off-on}^k Seconds  
5. RETURN; //only one server is switched on  
6. end foreach  

3.2.9 Other controls

Factors required by VM consolidation approach, thermal coverage model are such as: a) type of temperature coverage control, b) number of times or frequency a VM can be considered for migration from one server to another server within a time period (a.k.a VM oscillation), and c) the mean server utilization that is important and influence the energy efficiency of the datacenter. These factors have been found and used in the VM consolidation approach and thermal coverage models to achieve better. Empirical analysis
to arrive at factor values has been captured in Appendix. A brief information on each of these factors are as follows:

3.2.9.1 Rack level temperature coverage controls

As discussed in Section 3.2.3, if server’s temperature exceeds the threshold temperature, then the server is marked to be with insufficient cooling coverage. This ensures that the server is not considered for workload processing till such time the server’s temperature reduces to an operational temperature below the threshold temperature value. This approach is termed as server level temperature coverage control.

Another conservative approach is to restrict or isolate the server rack as a whole, even if one server in the rack is with current temperature that exceeds the threshold temperature. Once the server rack is marked to be with insufficient cooling coverage, none of the servers in the rack is considered for workload processing till such time that the current temperature of all the servers in the rack reduces to an operational temperature below the threshold temperature value.

3.2.9.2 Virtual machine migration and oscillation controls

Server de-consolidation process involves migrating one or more VMs from one server (when the server becomes overloaded), to another server (that can accommodate the VMs). This is done to ensure that the performance of VMs in the overloaded server is not affected. Also, server consolidation process involves migrating VMs from a server which is under-loaded or which has the least utilization, to another server that can accommodate the VMs. This is done to reduce the number of active on servers to save on energy consumption.

Due to both server de-consolidation and consolidation process, VMs are selected for migration from one server to another. After effect, is that some VMs might be considered for migration more frequently than others over a given time period. Increase in number of migrations impacts the performance
of the VM due to context checkpoint management activity involved with migration process and continuity of service hosted in the VM.

In this process, it is important to ensure that VMs so selected for migration between servers are not considered frequently. In essence, it is required to limit the number of times or the frequency a VM can be selected for migration within a time period.

3.2.9.3 Server utilization history based consolidation controls

As part of the proposed server de-consolidation and consolidation process, server utilization at an epoch interval is used to arrive at a decision as to whether the server is over-loaded or under-loaded. Just because the server's utilization is high at an epoch interval, it does not imply that the server will have high utilization at subsequent epoch intervals as well, and similarly, if a server exhibits low utilization at an epoch interval, it does not imply that the server will have low utilization at subsequent epoch intervals also. As a conservative approach, the mean utilization over a period between epoch intervals is considered to identify over loaded and under loaded servers.

3.3 SUMMARY

This Chapter, as a first step, proposed a heuristic approach to account for server heterogeneity to rank energy efficient servers considering the server's performance and server's combined compute and cooling power consumption. This Chapter, as the second step, presented a dynamic VM placement, dynamic server consolidation, dynamic server de-consolidation, VM selection from over-loaded or over-temperature or under-loaded servers for migration and VM migration approaches constraint by server energy efficiency based rank from the first step, server over-load, server over-temperature, and server under-load threshold criteria's. This Chapter, as the third step, presented a dynamic DPM server state transition control approach to conservatively (soft reactive approach using wait-time delays) switch off servers when VMs hosted in it are not processing workloads or switch the
servers on when the server is required to process workloads. Finally, the evaluation of all the proposed algorithms is elaborated in Chapter 7.