A Toolkit for Modeling and Simulation of Real-time Virtual Machine Allocation in a Cloud Data Center

Main Contents of this Chapter

- CloudSched architecture and main features
- Performance metrics for different scheduling algorithms Status and trends of cloud computing
- Design and implementation of CloudSched
- Performance evaluation

11.1 Introduction of the cloud data center

Cloud computing is developing based on various recent advancements in virtualization, grid computing, web computing, utility computing, and related technologies. Cloud computing provides both platforms and applications on demand through the internet or intranet [1]. Some key benefits of cloud computing include the hiding and abstraction of complexity, virtualized resources, and efficient use of distributed resources. Some examples of emerging cloud computing platforms are the Google App Engine [2], the IBM blue cloud [3], Amazon EC2 [4], and Microsoft Azure [5]. Cloud computing allows the sharing, allocation, and aggregation of software, computational, and storage network resources on demand. Cloud computing is still considered in its infancy, as there are many challenging issues to be resolved [1,6,7,8]. Youseff et al. [9] established a detailed ontology of dissecting the cloud into five main layers from top to down, as shown in Figure 11.1:

1. cloud application (SaaS)
2. cloud software environment (PaaS)
3. cloud software infrastructure (IaaS)
4. software kernel
5. hardware (HaaS)

Figure 11.1 also illustrates the interrelations, as well as the interdependency, on preceding technologies. In this chapter, we focus on infrastructure as a service (IaaS) in cloud data centers (CDCs).
A CDC can be a distributed network in structure, which is composed of many computing nodes (such as servers), storage nodes, and network devices. Each node is formed using a series of resources such as CPU, memory, network bandwidth, etc. Each resource has its own corresponding properties. There are many different types of resources for cloud providers. This chapter focuses on IaaS. The definition and model defined in this chapter are aimed to be general enough to be used by a variety of cloud providers. In a traditional data center, applications are tied to specific physical servers that are often over-provisioned to deal with workload surges and unexpected failures. Such configuration rigidity makes data centers expensive to maintain because of wasted energy and floor space, low resource utilization, and significant management overhead.

Using virtualization technology, current CDCs become more flexible, secure, and allow on-demand allocating. With virtualization, CDCs should have the ability to migrate an application from one set of resources to another in a nondisruptive manner. Such agility becomes important in modern cloud computing infrastructures that aim to efficiently share and manage extremely large data centers. A technology plays an important role in CDCs is resource scheduling.

Much research has been conducted in scheduling algorithms. Most of them are for the load balancing of traditional web servers or server farms. One of the challenging scheduling problems in CDCs is to consider allocation and migration of reconfigurable virtual machines (VMs) and integrated features of hosting physical machines (PMs). Unlike traditional load balancing scheduling algorithms, which consider only physical servers with one factor (such as CPU), new algorithms treat CPU, memory, and network bandwidth integrated for both PMs and VMs. In addition, real-time VM allocation for multiple parallel jobs and PMs is considered.

With the development of cloud computing, the size and density of the CDC became large and problems that need to be solved therewith. Examples of these problems include: how to intensively manage physical resources and virtual resources and dynamically use them, how to improve elasticity and flexibility (which can improve service and reduce cost and risk management), and how to help customers build flexible, dynamic, and adaptive infrastructure that allows

![Layered architecture of cloud computing](image-url)
enterprises to ensure sustainable future development without an increase in spending. It is extremely difficult to research widely for all these problems in real internet platforms because the application developers cannot control and process the network environment. What’s more, the network conditions cannot be predicted or controlled, but they still affect the quality evaluation of the strategies. The research of dynamic and large-scale distributed environments can be achieved by building a data center simulation system, which supports visualized modeling and simulation in large-scale applications in cloud infrastructure. A data center simulation system can describe the application workload statement, which includes user information, data center position, the amount of users and data centers, and the amount of resources in each data center. Using this information, the data center simulation system generates response requests and allocates these requests to VMs. By using a data center simulation system, application developers can evaluate suitable strategies, such as distributing reasonable data center resources, selecting a data center to match special requirements, reducing costs, etc.

Buyya et al. [7] introduced the GridSim toolkit for the modeling and simulation of distributed resource management for grid computing. Dumitrescu and Foster [8] introduced the GangSim tool for grid scheduling. Buyya et al. [7] introduced the modeling and simulation of cloud computing environments at the application level, in which simple scheduling algorithms, such as time-shared and space-shared, are discussed and compared. CloudSim [7] is a cloud computing simulator, which has the following functions:

1. supporting modeling of large-scale cloud computing infrastructure, both in a single physical computing node and a Java VM data center
2. modeling of the data center, service agency, and scheduling and distributing strategies
3. providing virtual engines, which is helpful for creating and managing several independent and collaborative virtual services in a data center node
4. be able to switch flexibly between processing cores with space-sharing and time-sharing

CloudAnalyst [12] aims to achieve the optimal scheduling among user groups and data centers based on the current configuration.

Both CloudSim and CloudAnalyst are based on SimJava [11] and GridSim [10], which makes them complicated. In addition, CloudSim and CloudAnalyst treat a CDC as a large resource pool and consider only application-level workloads. Therefore, they may not suitable for an IaaS simulation where each VM as resource is considered requested and allocated.


There is a lack of tools that enable developers to evaluate the requirements of large-scale cloud applications in terms of comparing different resource scheduling algorithms regarding the geographic distribution of both computing servers and user workloads. To fill this gap in tools for evaluation and modeling of cloud
environments and applications, in this chapter we propose CloudSched to be used for dynamic resource scheduling in a CDC. CloudSched supports multiple scheduling algorithms and it is suitable for the use and comparison of different scheduling algorithms. Unlike traditional scheduling algorithms that consider only one factor (such as CPU), which can cause hotspots or bottlenecks in many cases, CloudSched treats multidimensional resources (such as CPU, memory, and network bandwidth integrated for both PMs and VMs). Real-time constraint of both VMs and PMs, which is often neglected in the literature, is considered in this chapter. The main contributions of this chapter are:

1. proposing a simulation system for modeling cloud computing environments and performance evaluation of different resource scheduling policies and algorithms;
2. focusing on the simulation of scheduling in an IaaS layer where related tools are still lacking;
3. designing and implementing a lightweight simulator combining real-time multidimensional resource information.

CloudSched offers the following novel features:

1. Modeling and simulation of large-scale cloud computing environments, including data centers, VMs, and PMs
2. Providing a platform for modeling different resource scheduling policies and algorithms at the IaaS layer for clouds
3. Both graphical and textual outputs are supported

The organization of remaining parts of this chapter is as follows: Section 11.2 introduces the CloudSched architecture and its main features. Section 11.3 discusses performance measurements of different scheduling algorithms. Section 11.4 presents the design and implementation of CloudSched. Section 11.5 discusses the simulation results by comparing a few different scheduling algorithms. Finally, conclusions are provided in Section 11.6.

11.2 The architecture and main features of CloudSched

The simplified layered architecture is shown in Figure 11.2:

1. Web portal. At the top layer is a web portal for users to select resources and send requests; essentially, a few types of VMs are preconfigured for users to choose.
2. Core layer of scheduling. Once user requests are initiated, they go to next level CloudSched scheduling, which is for selecting appropriate data centers and PMs based on user requests. CloudSched provides support for modeling and simulation of CDCs, especially allocating VMs (consisting of CPU, memory, storage, bandwidth, etc.) to suitable PMs. This layer can manage a large scale of CDCs consisting of thousands of PMs. Different scheduling algorithms can be applied in different data centers based on customers’ characteristics.
3. Cloud resource. At the lowest layer are cloud resources that include PMs and VMs, both consisting of certain amounts of CPU, memory, storage, and bandwidth.
Some other tools, such as CloudSim and CloudAnalyst, are based on existing simulation tools such as JavaSim and GridSim, which makes the simulation system very large and complicated. Considering these, CloudSched uses a lightweight design and is focused on resource scheduling algorithms.

The main features of CloudSched are the following:

1. Focus on the IaaS layer. Unlike existing tools that focus on the application (task) level, such as CloudSim and CloudAnalyst, CloudSched focuses on scheduling VMs at the IaaS layer, i.e., each request needs one or more VMs, whereas each request only occupies a portion of the total capacity of a VM in CloudSim and CloudAnalyst.

2. Providing a uniform view of all resources. Similar to Amazon EC2 real applications, CloudSched provides a uniform view of all physical and virtual resources so that both system management and user selections are simplified. We will explain this in detail in the following section.

3. Lightweight design and scalability. Compared to other existing simulation tools, such as CloudSim and CloudAnalyst, which are built on GridSim (may cause complications), CloudSched focuses on resource scheduling policies and algorithms. CloudSched can simulate tens of thousands of requests in a few minutes.

4. High extensibility. Modular design is applied in CloudSched. Different resource scheduling policies and algorithms can be plugged into and compared with each other for performance evaluation. In addition, multiple CDCs are modeled and can be extended to a very large distributed architecture.

5. Easy to use and repeatable. CloudSched enables users to set up simulations easily and quickly with easy-to-use graphical user interfaces and outputs. It can accept inputs from

---

![CloudSched Diagram](image-url)

**Figure 11.2** Simplified layered architecture of CloudSched.
text files and output to text files. CloudSched can save simulation inputs and outputs so that modelers can repeat experiments. CloudSched ensures that repeated simulation yields identical results. Some GUIs are shown in Figure 11.3 and illustrated in Figure 11.4.

6. Easy to configure and evaluate different algorithms. CloudSched provides a high degree of control over the simulation. Entities and configuration options are modeled with major features: CDC is defined in terms of PMs consisting of CPU, memory, and bandwidth (or storage); VM is defined in terms of CPU, memory, and bandwidth (or storage), a few typical types of VMs are preconfigured; different resource scheduling policies and algorithms are dynamically selectable for different data centers. Using identical inputs for different scheduling policies and algorithms, CloudSched can collect results and automatically plot different outputs to compare performance indices.

11.2.1 Modeling CDCs

The core hardware infrastructure related to the clouds is modeled in the simulator by a data center component for handling VM requests. A data center is mainly composed by a set of hosts, which are responsible for managing VMs during their life cycles. Host is a component that represents a physical computing node in a cloud: it is assigned a preconfigured processing capability (expressed in computing power in CPU units), memory, bandwidth, storage, and a scheduling policy for allocating processing cores to VMs. A VM can be represented in a similar way.

![Figure 11.3 Main interface of CloudSched [1].](image-url)
11.2.2 Modeling VM allocation

With virtualization technologies, cloud computing provides flexibility in resource allocation. For example, a PM with two processing cores can host two or more VMs on each core concurrently. VMs can only be allocated if the total used amount of processing power by all VMs on a host is not more than the one available in that host.

Taking the widely used example of Amazon EC2, we show that a uniform view of different types of VMs is possible. Table 11.1 provides eight types of VMs from Amazon EC2 online information. Amazon EC2 does not provide information on its hardware configuration. However, we can therefore form three types of different PMs (or PM pools) based on compute units. In a real CDC, for example, a PM with $2 \times 68.4$ GB memory, $16 \text{ cores} \times 3.25$ units, and $2 \times 1690$ GB storage can be provided. In this way, a uniform view of different types of VMs is possibly formed. This kind of classification provides a uniform view of virtualized resources for heterogeneous virtualization platforms, e.g., Xen, KVM, VMWare, and brings great benefits for VM management and allocation. Customers only need to select suitable types of VMs based on their requirements. There are eight types of VMs in

![Figure 11.4 Main interface of CloudSched [2].](image-url)
EC2, as given in Table 11.1, where MEM stands for memory with unit GB, CPU is normalized to unit (each CPU unit is equal to 1 Ghz 2007 Intel Pentium processor [4]) and Sto stands for hard disk storage with unit GB. Three types of PMs are considered for heterogeneous cases, as given in Table 11.2. Currently, CloudSched implements dynamic load balancing, maximizing utilization, and energy-efficient scheduling algorithms. Other algorithms, such as reliability-oriented and cost-oriented, can be applied as well.

### Table 11.1 Eight types of VMS in Amazon EC2

<table>
<thead>
<tr>
<th>MEM</th>
<th>CPU (units)</th>
<th>BW(or Sto)</th>
<th>VM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.7</td>
<td>1 (1 cores × 1 units)</td>
<td>160</td>
<td>1-1(1)</td>
</tr>
<tr>
<td>7.5</td>
<td>4 (2 cores × 2 units)</td>
<td>850</td>
<td>1-2(2)</td>
</tr>
<tr>
<td>15.0</td>
<td>8 (4 cores × 2 units)</td>
<td>1690</td>
<td>1-3(3)</td>
</tr>
<tr>
<td>17.1</td>
<td>6.5 (2 cores × 3.25 units)</td>
<td>420</td>
<td>2-1(4)</td>
</tr>
<tr>
<td>34.2</td>
<td>13 (4 cores × 3.25 units)</td>
<td>850</td>
<td>2-2(5)</td>
</tr>
<tr>
<td>68.4</td>
<td>26 (8 cores × 3.25 units)</td>
<td>1690</td>
<td>2-3(6)</td>
</tr>
<tr>
<td>1.7</td>
<td>5 (2 cores × 2.5 units)</td>
<td>350</td>
<td>3-1(7)</td>
</tr>
<tr>
<td>7.0</td>
<td>20 (8 cores × 2.5 units)</td>
<td>1690</td>
<td>3-2(8)</td>
</tr>
</tbody>
</table>

### Table 11.2 Three types of PMs suggested

<table>
<thead>
<tr>
<th>PM</th>
<th>CPU (units)</th>
<th>MEM</th>
<th>BW (or Sto)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16 (4 cores × 4 units)</td>
<td>160</td>
<td>1-1(1)</td>
</tr>
<tr>
<td>2</td>
<td>52 (16 cores × 3.25 units)</td>
<td>850</td>
<td>1-2(2)</td>
</tr>
<tr>
<td>3</td>
<td>40 (16 cores × 2.5 units)</td>
<td>1690</td>
<td>1-3(3)</td>
</tr>
</tbody>
</table>

EC2, as given in Table 11.1, where MEM stands for memory with unit GB, CPU is normalized to unit (each CPU unit is equal to 1 Ghz 2007 Intel Pentium processor [4]) and Sto stands for hard disk storage with unit GB. Three types of PMs are considered for heterogeneous cases, as given in Table 11.2. Currently, CloudSched implements dynamic load balancing, maximizing utilization, and energy-efficient scheduling algorithms. Other algorithms, such as reliability-oriented and cost-oriented, can be applied as well.

### 11.2.3 Modeling customer requirements

CloudSched models customer requirements by randomly generating different types of VMs and allocating VMs based on appropriate scheduling algorithms in different data centers. The arrival process, service time distribution, and required capacity distribution of requests can be generated according to random processes. The arrival rate of customers' requests can be controlled. Distribution of different types of VM requirements can also be set. A real-time VM request can be represented in an interval vector: vmID(VM typeID, start-time, end-time, requested capacity). For example, vm1(1, 0, 6, 0.25) shows that the request ID is 1, VM is of type 1 (corresponding to integer 1), start-time is 0, and end-time is 6 (here, 6 can mean the sixth slot ended at time 6) and 0.25 for the capacity of a VM occupies from a given PM. Other requests can be represented in similar ways. Figure 11.5 shows the life cycles of VM allocation in a slotted time window using two PMs, where PM1 hosts vm4, vm5, and vm6, whereas PM2 hosts vm1, vm2, and vm3.
11.3 Performance metrics for different scheduling algorithms

Unlike traditional scheduling algorithms that consider only one aspect, which can cause hotspots or bottlenecks in many cases, CloudSched treats multidimensional resources, such as CPU, memory, and network bandwidth integrated for both PMs and VMs. There is lack of related metrics for scheduling algorithms considering multidimensional resources. For different scheduling objectives, there are different metrics. In the following, we consider metrics for load balancing, energy efficiency, and maximizing utilization. Other metrics for different objectives can be extended easily.

11.3.1 Metrics for multidimensional load balancing

In the following, we review some existing metrics and then develop an integrated measurement for the total imbalance level of the CDC, as well as the average imbalance level of each server. Wood et al. [13] introduced a few VM migration techniques. One integrated load balance metric is applied as follows:

\[
V = \frac{1}{(1 - \text{CPU}_u)(1 - \text{MEN}_u)(1 - \text{NET}_u)}
\]  

(11.1)

where CPU\(_u\), MEN\(_u\), and NET\(_u\) are the average utilization of CPU, memory, and network bandwidth, respectively, during each observed period. The large value \(V\) is, the higher of integrated utilization. Migration algorithms can therefore be based on this measurement. This actually is a strategy of minimizing integrated resource utilization.
by converting three-dimensional (3D) resource information into a one-dimensional (1D) value. This conversion may cause multidimensional information loss.

Zheng et al. [16] proposed another integrated load balancing metric as follows:

\[
B = \frac{aN_1C_i}{N_{1m}C_m} + \frac{bN_2M_i}{N_{2m}M_m} + \frac{cN_3D_i}{N_{3m}D_m} + \frac{dNet_i}{Net_m}
\]  \hspace{1cm} (11.2)

The referred physical server \( m \) is selected first. Then, other physical servers \( i \) are compared to server \( m \). \( N_1 \) is the CPU capability, \( N_2 \) is the memory capability, and \( N_3 \) is the hard disk. Here, \( C_i \) and \( M_i \) denote the average utilization of CPU and memory, respectively. \( D_i \) represents the transferring rate of hard disk and \( Net_i \) represents the network throughput. Here, \( a, b, c, \) and \( d \) denote the weighting factors for CPU, memory, hard disk, and network bandwidth, respectively. The major idea of this algorithm is to select the smallest value \( B \) among all physical servers to allocate VMs. This technique is also converting 3D resource information into a 1D value.

Singh et al. [14] introduced a novel Vector Dot algorithm to consider integrating factors of load balance for flow paths in data centers. For a server node, the node fraction vector \( \langle \text{CPUU}/\text{CPUCap}, \text{memU}/\text{memCap}, \text{netU}/\text{netCap} \rangle \) is defined, where CPUU, memU, and netU denote the average utilization of CPU, memory, and network bandwidth of a server, respectively. CPUCap, memCap, and netCap denote the total capacity of CPU, memory, and network bandwidth of a server, respectively. And the node utilization threshold vector is given by \( \langle \text{CPUT}, \text{memT}, \text{netT}, \text{ioT} \rangle \), where CPUT, memT, netT, and ioT represent the utilization threshold of CPU, memory, network bandwidth, and IO, respectively. To measure the degree of overload in a node and the system, the notion of an imbalance score is used. The imbalance score for a node is given by:

\[
\text{IBscore}(f, T) = \begin{cases} 
0, & \text{if } f < T \\
\frac{e^{(f-T)/T}}{e^{(T-T)/T}}, & \text{otherwise}
\end{cases}
\]  \hspace{1cm} (11.3)

By summing imbalance scores of all nodes, the total imbalance score of the system is obtained. This nonlinear measurement has the advantage of distinguishing between a pair of nodes at 3T and T and a pair of nodes both at 2T. The imbalance score is a good measurement for comparing average utilization to its threshold. Considering the advantages and disadvantages of existing metrics for resource scheduling, an integrated measurement for the total imbalance level of a CDC, as well as the average imbalance level of each server, has been developed for load balancing strategy. Other metrics for different scheduling strategies can be developed as well. The following parameters are considered:

1. Average CPU utilization \( \text{CPU}_i^u \) of a single server \( i \). This is defined as the averaged CPU utilization during an observed period. For example, if the observing period is 1 min and the CPU utilization is recorded every 10 s, then \( \text{CPU}_i^u \) is the average of six recorded values of server \( i \).
2. Average utilization of all CPUs in a CDC. Let CPU\textsubscript{n} be the total number of CPUs of server \textit{i},

\[
\text{CPU}\textsubscript{a} = \frac{\sum_i^{N} \text{CPU}\textsubscript{U} \text{CPU}\textsubscript{n}}{\sum_i^{N} \text{CPU}\textsubscript{n}}
\]  

(11.4)

where \(N\) is the total number of physical servers in a CDC. Similarly, the average utilization of memory, network bandwidth of server \textit{i}, all memories, and all network bandwidth in a CDC can be defined as MEM\textsubscript{U}, NET\textsubscript{U}, MEM\textsubscript{A}, and NET\textsubscript{A}, respectively.

3. Integrated load imbalance value (ILB\textsubscript{i}) of server \textit{i}. Variance is widely used as a measure of how far a set of numbers are spread out from each other in statistics. Using variance, an integrated load imbalance value (ILB\textsubscript{i}) of server \textit{i} is defined:

\[
\frac{(\text{Avg}\textsubscript{i} - \text{CPU}\textsubscript{A})^2 + (\text{Avg}\textsubscript{i} - \text{MEM}\textsubscript{A})^2 + (\text{Avg}\textsubscript{i} - \text{NET}\textsubscript{A})^2}{3}
\]

where

\[
\text{Avg}\textsubscript{i} = \frac{(\text{CPU}\textsubscript{U} + \text{MEM}\textsubscript{U} + \text{NET}\textsubscript{U})}{3}
\]

(11.5)

(ILB\textsubscript{i}) is applied to indicate load imbalance level comparing utilization of CPU, memory, and network bandwidth of a single server itself.

4. The imbalance value of all CPUs, memories, and network bandwidth. Using variance, the imbalance value of all CPUs in a data center is defined as

\[
\text{IBL}\textsubscript{cpu} = \sum_i^{N} (\text{CPU}\textsubscript{U} - \text{CPU}\textsubscript{A})^2
\]

(11.7)

Similarly, imbalance values of memory and network bandwidth can be calculated. Then total imbalance values of all servers in a CDC is given by

\[
\text{IBL}\textsubscript{tot} = \sum_i^{N} \text{ILB}\textsubscript{i}
\]

(11.8)

5. Average imbalance value of a physical server \textit{i}. The average imbalance value of a physical server \textit{i} is defined as

\[
\text{IBL}\textsubscript{PM} = \frac{\text{IBL}\textsubscript{tot}}{N}
\]

(11.9)

where \(N\) is the total number of servers. As its name suggests, this value is used to measure imbalance level of all physical servers.

6. Average imbalance value of a CDC. The average imbalance value of a CDC is defined as

\[
\text{IBL}\textsubscript{avg} = \frac{\text{IBL}\textsubscript{cpu} + \text{IBL}\textsubscript{mem} + \text{IBL}\textsubscript{net}}{N}
\]

(11.10)

7. Average running times. Average running time of the proceeding same amount of tasks can be compared for different scheduling algorithms.

8. Makespan. This is defined as the maximum load (or average utilization) on any PM.

9. Utilization efficiency. In this case, this is defined as the minimum load on any PM divided by the maximum load on any PM.
11.3.2 Metrics for energy efficiency

11.3.2.1 Power consumption model

1. The power consumption model of a server. Most power consumption in data centers comes from computation processing, disk storage, network, and cooling systems. In Ref. [17], the authors proposed a power consumption model for blade server, where $P$ is defined as

$$
14.5 + 0.2U_{cpu} + (4.5E - 8)U_{mem} + 0.003U_{disk} + (3.1E - 8)U_{net}
$$

(11.11)

where $U_{CPU}$, $U_{mem}$, $U_{disk}$, and $U_{net}$ are the utilization of CPU, memory, hard disk, and network interface, respectively. It can be seen that other factors such as memory, hard disk, and network interface have a very small impact on the total power consumption. In Ref. [3], the authors found that CPU utilization is typically proportional to the overall system load, and proposed the following power model:

$$
P(U) = kP_{max} + (1 - k)P_{max}U
$$

(11.12)

where $P_{max}$ is the maximum power consumed when the server is fully utilized, $k$ is the fraction of power consumed by the idle server (studies show that on average it is about 0.7), and $U$ is the CPU utilization. This chapter focuses on CPU power consumption, which accounts for the main part of energy compared to other resources such as memory, disk storage, and network devices.

In the real environment, CPU utilization may change over time due to the workload variability. Thus, the CPU utilization is a function of time and is represented as $u(t)$. Therefore, the total energy consumption by a PM ($E_i$) can be defined as an integral of the power consumption function over a period of time as:

$$
E_i = \int_{t_0}^{t_1} P(u(t))dt
$$

(11.13)

If $u(t)$ is constant over time (e.g., average utilization is adopted, $u(t) = u$), then $E_i = P(u)(t_1 - t_0)$.

2. The total energy consumption of a CDC is computed as

$$
E_{cdc} = \sum_{i=1}^{n} E_i
$$

(11.14)

It is the sum of all energy consumed by all PMs. Note that energy consumption of all VMs on PMs is included.

3. The total number of PMs used. This is the total number of PMs used for the given set of VM requests. It is important for energy efficiency.

4. The total power-on time of all PMs used. Based on the energy consumption equation of each PM, the total power-on time is the key factor.

11.3.3 Metric for maximizing resource utilization

1. Average resource utilization. Average utilization of CPU, memory, hard disk, and network bandwidth can be computed and an integration utilization of all these resources can also be used.

2. The total number of PMs used. It is closely related to the average and entire utilization of a CDC.
11.4 Design and implementation of CloudSched

In this section, we provide details related to the design and implementation of CloudSched. A Java discrete simulator is implemented. In the following, major building blocks of the CloudSched are described briefly.

11.4.1 IaaS resources considered

IaaS resources considered in this chapter include:

1. PMs: Physical computing devices that form data centers. Each PM can provide multiple VMs and each PM can have a multiple composition of CPU, memory, hard drives, network cards, and related components.
2. Physical clusters: These consist of a number of PMs, necessary network, and storage infrastructure.
3. VM: A virtual computing platform on the PM that uses virtualization software. It has a number of virtual CPUs, memory, storage, network cards, and related components.
4. Virtual cluster: consists of a number of VMs and necessary network and storage infrastructure.

11.4.2 Scheduling process in CDC

Figure 11.6 provides a referred architecture of CDCs and major operations of resource scheduling:

1. User requests: The user initiates the request through the internet (such as login cloud service provider’s web portal).
2. Scheduling management: Scheduler Center makes decisions based on the user’s identity (geographic location, etc.) and the operational characteristics of the request (quantity and quality requirements). The request is submitted to the appropriate data center and then the data center management program submits it to Scheduler Center. The Scheduler Center allocates the request based on scheduling algorithms applied in CDCs.
3. Feedback: The scheduling algorithm provides available resources to the user.
4. Execute scheduling: The scheduling results (such as deploying steps) are sent to the next stage.
5. Updating and optimization: Scheduler updates resource information and optimizes resources among different data centers according to the optimizing objective functions.

Figures 11.7 and 11.8 show general and detailed UML diagrams of the main resources in CDCs, respectively. Figure 11.7 shows the major resources and their relationships in CDCs and Figure 11.8 shows the properties of each major resource (classes).

11.4.3 Scheduling algorithms: taking the LIF algorithm as an example

Figure 11.9 shows the pseudocodes of least imbalance level first (LIF) algorithm for dynamic load balance of a CDC. Inputs to the algorithm include current
Figure 11.6 Referred architecture of CDCs.

Figure 11.7 UML diagram of main resources in CDCs.
Algorithm: Lowest-Average-Value-First(R)

Input: placement request \( r = (id, t_s, t_e, k) \);
status of current active tasks and PMs

Output: placement scheme for \( r \) and IBL_tot.

1) initialization: LowestAvg = large number;
2) For \( i=1:N \) Do
3) If request \( r \) can be placed on PM \( (i) \)
4) Then
5) compute \( \text{avg}(i) \) utilization value of PM\((i)\) it using equations (4)-(6);
6) If \( \text{avg}(i)<\text{LowestAvg} \)
7) Then
8) \( \text{LowestAvg}=\text{avg}(i) \);
9) allocatedPMID\(=i\);
10) Else
11) Endif
12) Else //find next PM
13) Endfor
14) IF LowestAvg == large number \( L \) // cannot allocate
15) Then put \( r \) into waiting queue or reject
16) Else place \( r \) on PM with allocatedPMID and compute IBL_tot

Figure 11.8 Detailed UML diagram of main resources in CDCs.

Figure 11.9 LIF algorithm.
VM request $r$, status of current active tasks, and PMs. For dynamic scheduling, the output is placement scheme for request $r$. Basically, the algorithm dynamically finds the lowest total imbalance value of the data center when placing a new VM request by comparing different imbalance values if the request is allocated to different PMs. The algorithm finds a PM with the lowest integrated load. This will make the total imbalance value of all servers in a CDC the lowest.

Figures 11.10 and 11.11 show the main class diagram and sequence diagram, respectively, of the LIF algorithm. Class ScheduleDomain consists of main methods and handles tasks in each queue by calling other classes. Class CreateRandVM and VmTaskInfo generate task requests. Class Allocate and Sort allocate the requests of VMs. Class Migrate and Allocate-Alg can migrate VMs. Record, PrintPM, and BalanceLevel are responsible for printing and output functions. Server, PM, and VM accomplish functions of physical servers and VMs.

Sequence diagram shows the following sequences of the algorithm:

1. Initialize the system
2. Obtain task requests
3. Allocate VM requests in the waiting queue
4. Operate migrating queues
5. Operate requesting queues
6. Operate deleting queues
First, a data center is selected (by the manager) using different IDs, then the number of and types of PMs are set up. Manager can also add/delete data centers.

Figure 11.12 shows one of the interfaces of configuring CDCs in CloudSched. Figure 11.13 shows one of the interfaces of configuring user requests. Probability
distribution of each type of VMs, the total number of simulated VMs, and preferred
data centers can be set up. The design diagram of main classes is depicted in
Figure 11.10.

11.5 Performance evaluation

We use regular Pentium PC with CPU 2 Ghz and 2 GB of memory for the simulation.

11.5.1 Random configuration of VMs and PMs

In this section, we provide simulation results for comparing four different scheduling
algorithms for load balance. For convenience, short name is given for each
algorithm as follows:

1. ZHJZ algorithm: As introduced in Ref. [16], the algorithm always chooses PMs with the
   lowest V value (as defined in Eq. (11.1)) and available resources to allocate VMs
   (Figure 11.14).
2. ZHJZ algorithm: Selects a referring PM [16], calculates the value, and chooses PMs with
   lowest B value (as defined in Eq. (11.2)) and available resources to allocate VMs.
3. LIF algorithm: Based on demands characteristics (e.g., CPU intensive, high memory, high
   bandwidth requirements etc.), always selects PMs with lowest integrated imbalance value
   (as defined in Eq. (11.5)) and available resource to allocate VMs.
4. Rand algorithm: randomly assigns requests (VMs) to PMs that have available resources.
5. Round-Robin algorithm: One of the simplest scheduling algorithms, it assigns tasks to
each physical server in equal portions and in circular order, handling all tasks without pri-
oiry (also known as cyclic executive).

For the simulation, three types of heterogeneous PMs are considered, each PM pool consists of some amount of PMs (can be dynamically configured and extended). For the simulation of a large number of VM requests,
both CPU and memory are configured with a large size, which can be set dynamically:

PM type 1: CPU 6 GHz, memory 8 G, and bandwidth 1000 M
PM type 2: CPU 12 GHz, memory 16 G, and bandwidth 1000 M
PM type 3 CPU 18 GHz, memory 32 G, and bandwidth 1000 M.

Similar to eight Amazon EC2 instances with high CPU, high memory, and standard configurations (but not exactly the same), eight types of VMs with equal probability of requests are generated randomly as follows (can be dynamic configured):

Type 1: CPU 1.0 GHz, memory 1.7 G, bandwidth 100 M
Type 2: CPU 4.0 GHz, memory 7.5 G, bandwidth 100 M
Type 3: CPU 8.0 GHz, memory 15.0 G, bandwidth 100 M
Type 4: CPU 5.0 GHz, memory 1.7 G, bandwidth 100 M
Type 5: CPU 20.0 GHz, memory 7.0 G, bandwidth 100 M
Type 6: CPU 6.5 GHz, memory 17.1 G, bandwidth 100 M
Type 7: CPU 13.0 GHz, memory 34.2 G, bandwidth 100 M
Type 8: CPU 26.0 GHz, memory 68.4 G, and bandwidth 100 M.

For all simulations, the number of PMs ranges from 100 to 600, the number of requests of VMs varies from 1000 to 6000, a Pentium PC with CPU 2 Ghz and 2 GB of memory is used for all simulations. The input data of user requests is generated using a program by considering equal probabilities of the previously
mentioned eight types of VMs. Of course, different (random) probabilities of different types of VMs can be generated. For steady-state analysis, a warm-up period (initial 2000 requests) is used to drop the transient period.

Figure 11.15 shows the average imbalance level, defined in Eq. (11.10), of a CDC. It can be seen that the LIF algorithm has the lowest average imbalance level when the total number of VMs and PMs are varied.

Figure 11.16 shows the average imbalance level of the entire physical server defined in Eq. (11.5). The LIF algorithm again has lowest average imbalance level for all PMs when the total number of VMs and PMs are varied.

Figure 11.15 Average imbalance values of a CDC.

Figure 11.16 Average imbalance values of each physical server.
Figure 11.17 shows the average imbalance level, defined in Eq. (11.10), of a CDC when the total number of physical servers is fixed but the number of VMs is varied.

Figure 11.18 shows the average imbalance level of the entire physical server, defined in Eq. (11.5), when the total number of physical servers is fixed but the number of VMs is varied. Through extensive simulation, similar results are observed.
11.5.2 **Divisible size configuration of PMs and VMs**

The configuration of VMs and PMs are explained in section 11.2.2. In Figures 11.19–11.21, we show the average utilization of CPU, memory, bandwidth, and the average of these three utilizations. We also show the imbalance value (IBL, as in Eq. (11.10)) of the entire data centers by running five different algorithms: Rand, Round-Robin, ZHJZ, ZHCJ, and LIF. It can be seen that in all the cases (when the total number of VMs and PMs are varying), that LIF has highest average utilization of CPU, memory, and bandwidth but has the lowest imbalance value. These results demonstrate that metrics obtained in divisible cases are much better than random configuration cases. Therefore, cloud providers such as Amazon can adopt these configurations to provide better quality of service regarding load balancing, energy efficiency, and other performance related requirements.

11.5.3 **Comparing energy efficiency**

We considered four algorithms here:

1. Round-Robin: The Round-Robin is the most commonly used scheduling algorithm (e.g., by Eucalyptus and Amazon EC2 [18]), which allocates VM requests in turn to each PM. The advantage of this algorithm is that it is simple to implement.

2. Modified Best Fit Decreasing (MBFD): MBFD is a bin-packing algorithm. Best Fit Decreasing is shown to use no more than 11/9 optimal solution (OPT)+1 bins (where OPT is the number of bins given by the optimal solution) [6]. The MBFD algorithm [6] first sorts all VMs in decreasing order of their current CPU utilizations and allocates each VM to a host that provides the least increase of power consumption due to this allocation. This allows leveraging the heterogeneity of resources by choosing the most power-efficient nodes first. For homogenous resources (PM), the VM can be allocated to any running PM that can still host because the power increasing is the same for homogenous

![Figure 11.19](image) Utilization and imbalance value of the entire data center when PMs = 100 and VMs = 1000.
resources. The complexity of the allocation part of the algorithm is \( nm \), where \( n \) is the number of VMs that must be allocated and \( m \) is the number of hosts. MBFD needs sorting requests so that it is only suitable for offline (or semi-offline) scheduling.

3. Offline Without Delay (OFWID): OFWID knows all requests in advance and follows the requests exactly without delay. It firstly sorts requests in increasing order of their start-times and allocates requests to PMs in increasing order of their IDs. If all running PMs cannot host the request, then a new PM is turned on.

4. Online Without Delay (ONWID): ONWID knows one request each time. It allocates requests to PMs in increasing order of their IDs. If all running PMs cannot host the

![Figure 11.20](image1)

**Figure 11.20** Utilization and imbalance value of the entire data center when PMs = 200 and VMs = 4000.

![Figure 11.21](image2)

**Figure 11.21** Utilization and imbalance value of the entire data center when PMs = 500 and VMs = 5000.
request, a new PM is powered on. When the total number of PMs is fixed, if all PMs still
cannot host the request, then the request is blocked.

11.5.3.1 Impact of varying maximum duration of VM requests

In this case, eight types of VMs are considered, as given in Table 11.1, which is based on Amazon EC2. The total number of arrivals (requests) is 1000 and each type of VMs has an equal number, i.e., 125. All requests follow the Poisson arrival process and have exponential service time, the mean interarrival period is set as 5, the maximum intermediate period is set as 50, and the maximum duration of requests are set as 50, 100, 200, 400, and 800 slots, respectively. Each slot is 5 min. For example, if the requested duration (service time) of a VM is 20 slots, actually its duration is \( \frac{20}{5} = 100 \text{ min} \). For each set of inputs (requests), experiments are run three times and all the results shown in this chapter are the average of the three runs. The configuration of PMs is based on eight types of VMs, as given in Table 11.2. In this configuration, there are three different types of PMs (heterogeneous case) and the total capacity of a VM is proportional to the total capacity of a PM. For comparison, we assume that all VMs are running using their requested capacity. Figure 11.22 shows the total energy consumption (in kilowatt hours) of the four algorithms as the maximum duration varies from 50 to 800, while all other parameters are the same.

11.5.3.2 Impact of varying the total number of VM requests

Next, we fix the total number of each type of PM but vary the total number of VM requests. The system load is defined as the average arrival rate (\( \lambda \)) divided by the average service rate (\( u \)). The arrival process follows the Poisson distribution and service time follows uniform distribution. To increase the system load, we vary the maximum duration of each request, whereas the total number of PMs remains fixed as 15 (each type has 5). Figure 11.23 provides the total energy consumption comparison.

11.6 Conclusions

In this chapter, we introduced a lightweight cloud resources scheduling emulator, CloudSched. Its major features and design and implementation details are presented. Simulation results are discussed for load balance and energy-efficient algorithms. CloudSched can help developers to identify and explore appropriate solutions considering different resource scheduling policies and algorithms. In the near future, we will develop more indices to measure the quality of related algorithms for different scheduling strategies such as maximization utilization of multi-dimensional resource. In addition, more simulation results, such as varying the
**Figure 11.22** Total energy consumption (in kilowatt hours) by varying maximum duration of VM requests.

**Figure 11.23** Total energy consumption (in kilowatt hours) by varying the number of VM requests.
probability of each VM request, fixing total number of physical servers, with a varying number of VMs are collected. Currently, different scheduling algorithms are compared inside a CDC but they can be extended to multiple data centers easily. CloudSched is designed for comparing different resource scheduling algorithms regarding IaaS. As for modeling and comparing features in SaaS (software as a service), PaaS (platform as a service), and other domains, the system needs to be extended as well.

References


