Chapter 3
Workflow Scheduling in Clouds

Cloud Computing has evolved as one of the most promising approach to execute large scale workflow applications. For successful implementation of any workflow application in cloud environment, one of the most significant tasks is to generate an efficient schedule before its execution. The main goal of workflow scheduling is to assign tasks to accessible resources in a finite time with the satisfaction of users’ specified QoS constraints. In this chapter, we are presenting an application of Intelligent Water drops (IWD) algorithm, a novel metaheuristic technique, to solve workflow scheduling problem focusing on reduction of makespan. The probability function of IWD algorithm is modified to improve the quality of solution and the convergence speed.

The rest of the chapter is organized as follows. Section 3.1 discusses cloud model and application model proceeded by workflow scheduling problem. In Section 3.2, we propose the methodology based on IWD algorithm to solve the formulated problem. Section 3.3 presents the performance evaluation of the proposed work. Finally, the chapter is concluded in Section 3.4.

3.1 System Model and Problem Formulation

This section describes the Cloud model and application model considered for the presented study followed by the workflow scheduling problem.

3.1.1 Cloud Model

Cloud environment consists of different types of virtual machines where a virtual machine \( VM_j \) of specific type is identified by its computing capacity \( CP(VM_j) \). The length of the tasks in Million Instructions and processing capacity of virtual machines in MIPS (Million Instructions Per Second) is known in advance. It is supposed that a VM can handle single task at a time and a task cannot be preempted i.e. a VM once assigned to a task will not be
taken back until it is finished. VMs acquired for executing a workflow belong to a single data center. Furthermore, bandwidth between the VMs in the same data center is nearly same.

3.1.2 Application Model

A workflow $W = (T, E)$ is illustrated as a Directed Acyclic Graph (DAG) where node set $T = \{t_1, t_2 \ldots t_n\}$ represents tasks and edge set $E$ represents precedence constraints and data dependencies among tasks. Each directed edge $e_{ij} = (t_i, t_j) \in E$ indicates that task $t_j$ cannot be executed until task $t_i$ has been executed. $t_i$ is called as the parent task of $t_j$ and $t_j$ is defined as the child task of $t_i$. A child task can execute only if all of its parent tasks have been executed and the required data from parent tasks is available at the virtual machine on which it is going to execute. A task having no parent task is termed as an entry task and a task without any child task is known as an exit task. As our algorithm considers only one entry and one exit task, we inserted two dummy tasks as entry and exit tasks with negligible execution time. These dummy tasks are connected to actual entry and exit tasks respectively. An example workflow is shown in Figure 3.1.

![Figure 3.1. An example workflow with nodes representing tasks and directed edges representing dependencies between tasks](image)

Execution time $ET(t_i, VM_j)$ of a task $t_i$ of size $Length(t_i)$ on a virtual machine $VM_j$ with processing capacity $CP(VM_j)$ can be computed as shown below:

$$ET(t_i, VM_j) = \frac{Length(t_i)}{CP(VM_j)}$$  \hspace{1cm} (3.1)

$TT(t_i, t_k)$ is the data transfer time between task $t_i$ and $t_k$ and can be estimated as follows:

$$TT(t_i, t_k) = \frac{OutputFileSize(t_i)}{BW}$$  \hspace{1cm} (3.2)
where $OutputFileSize(t_i)$ is amount of data needs to be moved from $t_i$ to $t_k$ and $BW$ is the mean bandwidth between the VMs in data center. $TT(t_i, t_k)$ is zero if both the tasks are scheduled on the same VM.

In a workflow, a task can commence its execution provided that its parent tasks have completed their processing and the required VM is available. Thus the start time $ST(t_i, VM_j)$ of a task $t_i$ on virtual machine $VM_j$ can be calculated as:

$$ST(t_i, VM_j) = \max\{Avail(VM_j), ReadyTime(t_i, VM_j)\}$$  \hspace{1cm} (3.3)

Here $Avail(VM_j)$ is the time when virtual machine $VM_j$ is finished with its previously assigned tasks and $ReadyTime(t_i, VM_j)$ denotes the time when all the parent tasks of task $t_i$ are completed and required data is transferred to the $VM_j$, which can be described as:

$$ReadyTime(t_i, VM_j) = \max_{t_{\text{parent}} \in \text{pred}(t_i)} \{FT(t_{\text{parent}}, VM_k) + TT(t_{\text{parent}}, t_i)\}$$  \hspace{1cm} (3.4)

where $FT(t_{\text{parent}}, VM_k)$ is finish time of a task $t_{\text{parent}}$ on virtual machine $VM_k$. It can be calculated as:

$$FT(t_{\text{parent}}, VM_k) = ST(t_{\text{parent}}, VM_k) + ET(t_{\text{parent}}, VM_k)$$  \hspace{1cm} (3.5)

For entry tasks, $ST(t_i, VM_j)$ will be equal to $Avail(VM_j)$.

### 3.1.3 Problem Formulation

Makespan of a workflow can be described as the completion time of the last task $t_i$ in the workflow on the assigned virtual machine $VM_j$ and could be computed as

$$Makespan = \max\{FT(t_i, VM_j)\}$$  \hspace{1cm} (3.6)

The problem can be formulated as: *Find a task to resource mapping such that makespan of a schedule generated for a workflow application should be minimized.*

Mathematically, it can be described as:

Minimize: \hspace{0.5cm} $F = (Makespan)$ \hspace{1cm} (3.7)

### 3.2 Proposed Methodology

In this section, we describe our proposed methodology for workflow scheduling based on Intelligent Water Drops (IWD) Algorithm. The first step for achieving solution is to encode the given problem in terms of IWD algorithm. Numerous IWDs begin with an entry task...
and seek a resource for the task based on the probabilistic function. Each IWD gradually builds a solution by moving from one task to another satisfying the precedence constraints and finding suitable resource for each task. The task to resource mapping is done based on probabilistic function. Thus each IWD builds the solution incrementally in $n$ steps, where $n$ denotes the number of tasks. Soil between task and its resource, and the soil and velocity of IWD is updated after each step of assigning resource to a task. The procedure for the entire IWDs to obtain feasible solutions is called an iteration. After all IWDs have found the solutions, the solution which is best out of these i.e. giving minimum makespan is selected. The global soil updation on the paths of current iteration-best-solution is done at the end of each iteration. Iterations are continued until the termination criteria are fulfilled and the solution which is best of all the iteration-best-solutions is selected as the final solution. The pseudocode of the algorithm is given in Algorithm 3.1. The details of each step of the algorithm are given below:

**Algorithm 3.1: IWD_ Main**

1. Initialize static parameters
2. Repeat steps 3 to 8 until $count <= Max_iter$
3. Initialize dynamic parameters
4. Place $N$ IWDs on entry node
5. Compute each IWD’s solution.
6. Determine the iteration best solution
7. Update the soils on the paths that constitute iteration-best solution
8. Update the overall best solution
9. Return overall best solution

**Step 1: Initialize static parameters**

The static parameters are initialized only once at the start of algorithm.

- The number of water drops $N$ is taken as 100.
- Maximum number of iterations, $Max_iter$ is initialized to 50.
- The parameters for updating velocity are set as $av=cv=1$ and $bv=0.01$.
- The parameters for updating soil are set as $as=cs=1$ and $bs=0.01$.
- Local soil updating parameter $LocalSoilUpdat$ is chosen as 0.9 and global soil updating parameter $GlbSoilUpdat$ is set as 0.1.
• Initial soil between each task and resource pair \( \text{soil}(t_i, VM_j) \) is randomly assigned to promote the diversity of initial search space.

**Step 2: Initialize dynamic parameters**

Dynamic parameters are reset following each iteration. In original IWD algorithm, each IWD is initially assigned zero soil and same velocity whereas in the proposed approach, to enhance the diversity of the solution space, velocity is randomly assigned to all IWDs. Initial soil assigned to each IWD is kept zero as before \( (\text{soil}(IWD) = 0) \).

**Step 3: Place N IWDs on entry node of workflow.**

**Step 4: For each IWD, repeat steps 4.1 to 4.4 until it completes its solution i.e. assigns resource to each task in the workflow.**

**Step 4.1:** Choose a task \( t_i \) from the workflow tasks, which is ready to execute. It is ready to execute if all its parent tasks have completed their execution.

**Step 4.2:** From the available resources, select a resource \( VM_j \) for the execution of task \( t_i \) based on the following probabilistic function:

\[
P(t_i, VM_j) = \begin{cases} 
\frac{f(\text{soil}(t_i, VM_j)) \times \frac{1}{FT(t_i, VM_j)}}{\sum_{k \in \text{available VMs}} f(\text{soil}(t_i, VM_k)) \times \frac{1}{FT(t_i, VM_k)}} & \text{if } rd > 0.4 \\
\min\left(1, \frac{f(\text{soil}(t_i, VM_j)) \times \frac{1}{FT(t_i, VM_j)}}{\sum_{k \in \text{available VMs}} f(\text{soil}(t_i, VM_k)) \times \frac{1}{FT(t_i, VM_k)}} \right) + R & \text{otherwise}
\end{cases}
\]  

(3.8)

where

\[
f(\text{soil}(t_i, VM_j)) = \frac{1}{\varepsilon + g(\text{soil}(t_i, VM_j))}
\]

(3.9)

and

\[
g(\text{soil}(i, j)) = \begin{cases} 
\text{soil}(t_i, VM_j) & \text{if } \min_{m \in \text{available VMs}} \text{soil}(t_i, VM_m) \geq 0 \\
\text{soil}(t_i, VM_j) - \min_{m \in \text{available VMs}} \text{soil}(t_i, VM_m) & \text{otherwise}
\end{cases}
\]  

(3.10)

For increasing the diversity of the selection process, a random number \( rd \in (0,1) \) is used to decide the approach for calculating the probability. For the case of \( rd \leq 0.4 \), a
random number R is added to probability to augment its randomness. A constant $\varepsilon = 0.01$ is used in function $f(\text{soil}(t_i, VM_j))$ to avert a possible division by zero and $\text{soil}(t_i, VM_j)$ represents the soil between task $t_i$ and resource $VM_j$. $f(\text{soil}(t_i, VM_j))$ is the inverse of the $\text{soil}(t_i, VM_j)$. The function $g(\text{soil}(t_i, VM_j))$ is used to move the $\text{soil}(t_i, VM_j)$ towards positive values. The original IWD algorithm calculates the probability as a function of $f(\text{soil}(t_i, VM_j))$ only. To improve the convergence speed of the algorithm, the proposed approach computes the probability using $f(\text{soil}(t_i, VM_j))$ and finish time of task $t_i$ on resource $VM_j$.

**Step 4.3:** Update the velocity of the IWD as follows:

$$\text{vel}(IWD) = \text{vel}(IWD) + \frac{\text{av}}{\text{bv} + \text{cv} \times \text{soil}^2(t_i, VM_j)} \quad (3.11)$$

Here $\text{av}$, $\text{bv}$ and $\text{cv}$ are constant parameters and $\text{bv}$ ensures that the equation is not divided by zero. The reason behind selecting an even power for $\text{soil}(t_i, VM_j)$ is to avoid negative values for updated velocity.

Computation of delta soil is done as follows:

$$\Delta \text{soil}(t_i, VM_j) = \frac{\text{as}}{\text{bs} + \text{cs} \times \text{time}^2(t_i, VM_j)} \quad (3.12)$$

Here $\text{as}$, $\text{bs}$ and $\text{cs}$ are constant parameters and $\text{bs}$ ensures that the equation is not divided by zero and $\text{time}(t_i, VM_j)$ is the time taken by the IWD to traverse the edge between $t_i$ and $VM_j$ with the velocity $\text{vel}(IWD)$ and is defined as $\text{time}(t_i, VM_j) = \frac{\text{HUD}(t_i, VM_j)}{\text{vel}(IWD)}$. For the given problem, $\text{HUD}(t_i, VM_j)$, heuristic undesirability measure can be determined as

$$\text{HUD}(t_i, VM_j) = \text{FT}(t_i, VM_j) \quad (3.13)$$

When the finish time of a task $t_i$ on a resource $VM_j$ is less, $\text{HUD}(t_i, VM_j)$ turns out to be small and thus time taken to traverse the path from $t_i$ to $VM_j$ reduces.

**Step 4.4:** Remove delta soil from the traversed path and add it to IWD.

$$\text{soil}(t_i, VM_j) = (1 - \text{LocalSoilUpdate}) \times \text{soil}(t_i, VM_j) - \text{LocalSoilUpdate} \times \Delta \text{soil}(t_i, VM_j) \quad (3.14)$$

and

$$\text{soil}(IWD) = \text{soil}(IWD) + \Delta \text{soil}(t_i, VM_j) \quad (3.15)$$

where $\text{soil}(IWD)$ is the soil carried by the IWD.

**Step 5: Find the iteration best solution**
The highest quality solution out of all solutions generated by IWDs in an iteration (step 4) is considered as the iteration best solution.

\[ \text{itr}_\text{best}_\text{sol} = \min_{\forall \text{IWD}} F(\text{IWD}) \]  

(3.16)

where \( F(\text{IWD}) \) represents the quality (fitness) of a solution found by a given IWD. For the specified problem, \( F(\text{IWD}) = \text{Makespan} \). Hence the solution, which gives the minimum makespan is considered as the solution with highest quality.

**Step 6: Update the soil of the edges that are part of the iteration best solution.**

\[
\text{soil}(t_i, VM_j) = (1 + \text{GlbSoilUpdat}) \times \text{soil}(t_i, VM_j) - \text{GlbSoilUpdat} \times \text{soil}(\text{IWD}_{\text{itr}_\text{best}}) \times \frac{1}{\text{itr}_\text{best}_\text{sol}}
\]

(3.17)

Here \( \text{soil}(\text{IWD}_{\text{itr}_\text{best}}) \) denotes the soil of the IWD which has given iteration best solution. By reducing the soil of the edges of iteration best solution, these solutions are steadily strengthened and they guide the IWDs to find the near good solutions, which finally lead to finding global optimal solution.

**Step 7: Update the overall best solution.**

If the makespan of current iteration best solution \( \text{itr}_\text{best}_\text{sol} \) is lower than the overall best solution \( \text{best}_\text{sol} \), then value of \( \text{best}_\text{sol} \) is substituted with \( \text{itr}_\text{best}_\text{sol} \) else it remains as before. It ensures that \( \text{best}_\text{sol} \) will hold the best solution acquired up to now by the IWD algorithm.

### 3.3 Performance Evaluation

In this section, performance of the proposed IWD algorithm is evaluated by comparing it with two existing algorithms: PSO [121] and SGA [98]. The experiments are conducted in a cloud simulation environment created by using WorkflowSim tool [164] considering heterogeneous resources. Three different workflows are chosen from diverse scientific areas: Montage, LIGO Inspiral and Epigenomics.

Figure 3.2 shows the organization of these workflows. Montage is a workflow application for research in astronomy which creates mosaics of the sky by processing input images. LIGO Inspiral workflow is used to detect and measure gravitational waves predicted in the physics domain. Epigenomics workflow is used in bioinformatics field to automate the execution of different genome sequencing operations. All these workflows are having
different data and computational requirements. Montage consists of I/O intensive tasks requiring less CPU utilization whereas Epigenomics is having CPU-intensive tasks. LIGO workflow is having large CPU, memory and resource requirements. A detailed description of these workflows is given in [30].

![Figure 3.2: Organization of workflows used in experiments][30]

We considered a single data center having five different VM types. The capacity of VMs is varied from 400 MIPS to 2000 MIPS with a step size of 400. Each workflow application is evaluated using four different sizes, small, medium, large and extra large. Small workflows have 24 to 30 tasks, medium workflows have 45 to 50 tasks, large workflows have 100 tasks and extra large ones have 995 to 1000. Each experiment was executed 10 times and the results shown are the average of these 10 runs.

![Figure 3.3: Makespan of different algorithms for Montage Workflow for different number of tasks][30]
The results achieved by the different algorithms for the Montage Workflows are presented in Figure 3.3. For Montage_Small workflow, the reduction in average makespan of IWD over PSO is 29% approximately, for the Montage_Medium there is an improvement of 41% in comparison to PSO and for the Montage_Large, IWD gives 53% better performance than PSO. If we consider the case of Montage_ExtraLarge, IWD performs 58% better than PSO. Here SGA performs much better than PSO and marginally better than IWD.

The results for the Epigenomics workflow are displayed in Figure 3.5. For Epi_Small workflow, the reduction in average makespan of IWD over PSO and SGA is 6% and 2% approximately and for the Epi_Medium, there is an improvement of 5% and 3% in

![Figure 3.4: Makespan for LIGO Workflow for different number of tasks](image)

The average makespan obtained by considered algorithms for the LIGO workflows are shown in Figure 3.4. For LIGO_small workflow, the reduction in average makespan of IWD over PSO and SGA is 14% and 6% approximately and for the LIGO_Medium, there is an improvement of 13% and 3% in comparison to PSO and SGA respectively. IWD gives 21% better performance than PSO, and 6% better performance than SGA for LIGO_Large workflow. If we consider the case of LIGO_ExtraLarge workflow, IWD performs 54% and 25% better than PSO and SGA respectively.

The results for the Epigenomics workflow are displayed in Figure 3.5. For Epi_Small workflow, the reduction in average makespan of IWD over PSO and SGA is 6% and 2% approximately and for the Epi_Medium, there is an improvement of 5% and 3% in
comparison to PSO and SGA respectively. IWD gives 8% and 6% better performance than PSO and SGA for Epi_Large workflow. If we consider the case of Epi_ExtraLarge workflow, IWD performs 42% and 35% better than PSO and SGA respectively.

Experimental results indicate that IWD performs better than PSO and SGA for most of the cases. Only in case of Montage workflow, SGA performs slightly better than IWD algorithm. IWD outperforms all these algorithms for LIGO and Epigenomics workflow, which are having CPU intensive tasks.

![Figure 3.5: Makespan for Epigenomics Workflow for different number of tasks](image)

In addition, the performance of IWD is more significant when the size of the workflow is extra large. The main rationale behind degraded performance of these algorithms in comparison to IWD in most of the cases is the random generation of initial population in these algorithms. This randomness increases the convergence time to find the quality
solution. IWD is a constructive-algorithm based approach which avoids this shortcoming by building problem solution stepwise by searching through the constructive graph.

3.4 Conclusion

In this chapter, we presented IWD based scheduling algorithm for minimizing makespan of the scientific workflows in cloud computing environment. The experiments are conducted in a cloud simulation environment created using WorkflowSim. The suggested algorithm is compared with PSO as well as SGA based scheduling strategies considering three scientific workflow applications from diverse domains. From the achieved results, it is clear that the IWD algorithm significantly outperforms PSO and SGA algorithms in terms of makespan through better allocation of resources to tasks.

In the next chapter, the algorithm is modified to further improve quality of solutions as well as extended to include another QoS parameter i.e. resource utilization.