CP3: Progress Report
Modified Min-Min Based Workflow Scheduling in Cloud Computing

Team 18: Muneeba Badar and Shifa Shah

April 20, 2025

Implementation Summary

The implementation follows a two-phase scheduling algorithm based on the modified Min-Min heuristic proposed in the paper. The key components developed include:

- Workflow input parser for tasks, dependencies, ECT table, and cloud servers.
- Priority-based task selection using a modified Min-Min heuristic that preserves precedence constraints.
- Task allocation mechanism that duplicates the entry task across all VMs and selects the VM with minimum EFT for other tasks.
- QoS metric calculations: makespan, load balancing, speedup, efficiency, and resource utilization.

All essential components from the algorithm are implemented and tested.

Correctness Testing

To ensure the correctness of the modified Min-Min scheduling algorithm, we tested it on Montage and randomly generated DAG workflows of sizes 10, 25, 50, 100, 200, 300, and 400 tasks, representing a diverse range of workflow complexities across different sizes and task dependencies.

Example Test Workflow Details

- **Tasks:** J1 to J10
- **Entry Task:** J1
- **Exit Task:** J10
- **Structure:** Tasks split after J1 into three branches (J2, J3, J4), with multiple join operations at J5, J7, and J8, forming a deep dependency chain.
- **Cloud Infrastructure:** 2 servers (R1, R2) with 3 virtual machines (R1_V1, R1_V2, R2_V3).
- **ECT Table:** Each task has distinct execution times on each VM to reflect heterogeneity.
- **Communication Delays:** Applied between dependent tasks based on VM-to-VM communication across servers.

Validation Strategy

1. **Dependency Verification:** Each task was only scheduled once all its dependencies were satisfied. For example, J5 was scheduled only after both J2 and J3 were completed, and J10 followed a complete chain from J1 to J9.

2. **Entry Task Duplication:** J1, as the entry task, was correctly duplicated across all three VMs (R1_V1, R1_V2, R2_V3), consistent with the algorithm's goal to reduce communication latency in subsequent tasks.

3. **EST/EFT Validation:** EST and EFT values were manually verified. For example:

   - J3 on R1_V1 had an EST of 14 (after J1) and EFT of 25, considering the execution time and any necessary delays.
   - J10 on R1_V2 started at EST 63 and finished at EFT 70.

4. **Correct Resource Allocation:**

   - R1_V1 was assigned tasks J1, J3, J6, J2, J5, J7, J8 in an order consistent with precedence and execution availability.
   - J9 and J10 were allocated to R1_V2 for faster makespan.

5. **QoS Metrics Validation:**

   - Makespan = 70
   - Load Balancing = 0.605
   - Speedup = 1.3; Efficiency = 43.3%
   - Resource Utilization = 0.604

Edge Cases

Edge cases tested include:

- No dependencies
- Equal ECT values on all VMs
- Entry and exit tasks only

The implementation adhered strictly to precedence constraints, minimized makespan by selecting optimal VMs based on EFT, and successfully managed entry task duplication. The algorithm's output for this test case confirms its correctness across temporal, structural, and performance dimensions.

Complexity & Runtime Analysis

Theoretical Complexity:
Each scheduling round evaluates n tasks across m VMs, leading to a complexity of $\mathcal{O}(n \cdot m)$.

Empirical Performance:
Scheduling workflows with 100–500 tasks across 20–25 VMs completes in under 1 second on a standard CPU (i5, 8GB RAM).

The results align with the paper's findings, validating the effectiveness of modifying Min-Min to handle dependencies and entry task duplication.

Comparative Evaluation

The traditional Min-Min heuristic is for independent tasks and doesn't consider load distribution, which can cause imbalanced resource utilization. The modified Min-Min heuristic, proposed in, addresses this by incorporating load-balancing principles considering both task completion time and resource load to achieve better load distribution and reduce execution time. It also extends to dependent tasks (DAG) while preserving precedence constraints (PC) and duplicates the entry task to minimize communication time. The evaluation in the paper shows that the modified Min-Min outperforms HEFT and PETS in terms of load balancing, makespan, speedup, efficiency, and resource utilization

Challenges & Solutions

- **Challenge:** Maintaining precedence while selecting tasks.
- **Solution:** Explicit dependency check before computing task priority.
- **Challenge:** Choosing the optimal VM considering communication delays.
- **Solution:** EFT computation includes inter-server communication time based on VM prefixes.

Enhancements

- The code is implemented in separate parts, each handling a specific task, making it easy to manage and update. It uses input and output files, so the process can be repeated with consistent results, and the structure allows easy modifications without affecting the whole system.
- Output structured to include task allocation, EST/EFT timings, and detailed metrics.
- Tested edge cases and on Montage and random workflows to verify consistency across datasets.

Conclusion: The implemented system accurately reflects the design of the proposed algorithm and demonstrates its functionality through successful execution and integration.