A novel cloud workflow scheduling algorithm based on stable matching game theory

Lorenzo Cappetti, Chiara Peppicelli

January 2026

UNIVERSITÀ
DEGLI STUDI
FIRENZE

UNIVERSITÀ DEGLI STUDI FIRENZE

Table of Contents

Scientific Workflows in Cloud Computing

- Many scientific applications (e.g., bioinformatics, astronomy, physics) are executed as **scientific workflows**.
- These workflows are commonly modeled as **Directed Acyclic Graphs (DAGs)**.
- Workflow tasks are executed on **heterogeneous cloud resources** (Virtual Machines).
- The assignment of DAG tasks to cloud VMs is known as **workflow scheduling**.
- Scheduling decisions strongly affect execution time and resource utilization.

$\Rightarrow$ **Workflow scheduling is a key challenge in cloud computing.**

Cloud Workflow Scheduling

1 Introduction

Cloud workflow scheduling is **challenging** due to:

- Resource heterogeneity;
- Task dependencies;
- Communication overhead;
- Conflicting optimization objectives.

Traditional scheduling lacks a balance between global goals and task-level equity.

The Challenge: Efficiency vs Task Fairness

Limitations of makespan-focused scheduling:

- Task-level fairness is frequently neglected.
- Imbalanced allocations may assign suboptimal resources to certain tasks.
- Ignoring individual task requirements can reduce overall user satisfaction.

Key Research Questions:

- How can *efficiency* and *fairness* be jointly optimized?
- Can game theory provide an effective solution?

Proposed Solution: SM-CPTD Algorithm

SM-CPTD = Stable Matching with Critical Path and Task Duplication.

Proposed in:

A novel cloud workflow scheduling algorithm based on stable matching game theory - Jia et al.

This innovative scheduling approach combines:

- **Stable Matching Game Theory** (SMGT) that models task-to-VM assignment as a two-sided matching problem.
- Critical-path optimization with Task Duplication.

$\Rightarrow$ **Balances workflow performance and task-level fairness.**

Project Objectives

This work aims to:

1. **Reproduce** the core algorithmic components of SM-CPTD following original specifications.
2. **Validate** reported performance improvements through independent experimentation.
3. **Evaluate** on standard scientific workflow benchmarks: *Montage*, *CyberShake*, *LIGO*, *Epigenomics*.
4. **Analyze** efficiency (SLR), utilization (AVU), and fairness (VF) of the scheduling.

Table of Contents

2 Problem Formulation

Workflow and Cloud Model

2 Problem Formulation

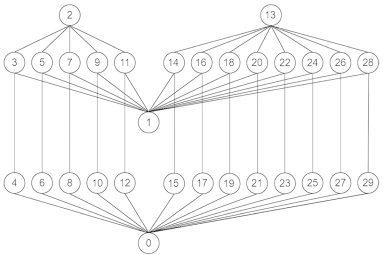


Figure: CyberShake Workflow (30 tasks).

A workflow is represented as **DAG** $= (T, E)$ where:

- $T = \{t_0, t_1, \ldots, t_{n-1}\}$: set of n tasks.
- E: directed edges encoding precedence constraints and data dependencies.
  - For each task t_i: pre(t_i) and succ(t_i) define the set of predecessor and successor tasks.
  - t_{entry}: tasks with no predecessors.
  - t_{exit}: tasks with no successors.
- Each task t_i has computational size s_i (MI).

The DAG structure defines execution order and communication requirements.

Data Transfer

2 Problem Formulation

Size of Data Transmission Matrix:

$$TT = \begin{bmatrix} 0 & TT_{0,1} & \cdots & TT_{0,n-1} \\ TT_{1,0} & 0 & \cdots & TT_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ TT_{n-1,0} & TT_{n-1,1} & \cdots & 0 \end{bmatrix} \in \mathbb{R}^{n \times n}$$

where

$$TT_{i,j} = \begin{cases} s_{t_i} \times \text{CCR} & \text{if } (t_i, t_j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

where **CCR** (Compute-to-Communication Ratio) is the ratio of the average communication cost to the average computation cost.

Cloud Infrastructure Model

2 Problem Formulation

Cloud Model: a set of m virtual machines (VMs)

$$V = \{VM_0, VM_1, \ldots, VM_{m-1}\}$$

- Each VM_k has a processing capacity of p_k (in MIPS).
- The VMs are generally heterogeneous: $p_i \neq p_j$ for $i \neq j$.
- Communication between VMs is represented by a bandwidth matrix B (in Mbps).

The bandwidth matrix B is defined as:

$$B(\mathrm{VM}_k, \mathrm{VM}_l) = \begin{cases} 0, & \text{if } k = l, \\ b_{k,l}, & \text{if } k \neq l, \end{cases} \qquad \text{with } b_{k,l} \neq b_{l,k} \text{ in general.}$$

Execution and Communication Timing

Execution and Transmission Time:

$$ET(t_i, VM_k) = \frac{s_i}{p_k} \qquad T_{trans}(t_i, t_j) = \begin{cases} \frac{TT_{i,j}}{B(VM_k, VM_l)} & \text{if } k \neq l \\ 0 & \text{if } k = l \end{cases}$$

Start and Finish Time:

$$ST(t_i, \mathrm{VM}_k) = \begin{cases} 0 & t_i = t_{\text{entry}} \\ \max\limits_{t_j \in \mathrm{pre}(t_i)} \{FT(t_j) + T_{\text{trans}}(t_i, t_j)\} & \text{otherwise} \end{cases}$$

$$FT(t_i, \mathrm{VM}_k) = ST(t_i, \mathrm{VM}_k) + ET(t_i, \mathrm{VM}_k)$$

Timing models account for both computational work and data movement latency.

Problem Definition and Constraint

Goal : Find a mapping function $\sigma : T \rightarrow V$ that assigns each task to exactly one VM.

Constraint:

- **Precedence:** a task starts only after all predecessors complete.
- **VM availability:** one task per VM at a time.
- **No migration:** each task is assigned to exactly one VM.

Evaluation Metrics (1/3)

1. Makespan - Total execution time of the workflow:

$$\text{makespan} = \max_{k \in \{0,\ldots,m-1\}} \{MS(VM_k)\}$$

where $\text{MS}(\text{VM}_k)$ denotes the finish time of the last task assigned to VM_k

2. Scheduling Length Ratio (SLR) - Normalized metric comparing makespan to theoretical lower bound:

$$SLR = \frac{\text{makespan}}{\sum_{t_i \in CP} \min_{VM_j \in V} ET(t_i, VM_j)}$$

$SLR \geq 1$ **lower is better** — makespan closer to theoretical minimum.

Evaluation Metrics (2/3)

3. Average VM Utilization (AVU) - Effectiveness of resource usage:

$$VU(\mathrm{VM}_k) = \frac{\sum_{t_i \in \mathrm{VM}_k.\mathrm{waiting}} ET(t_i, \mathrm{VM}_k)}{\mathrm{makespan}}, \quad AVU = \frac{1}{m} \sum_{k=1}^{m} VU(\mathrm{VM}_k)$$

where VU is defined as the ratio between the total execution time of tasks assigned to it and the workflow makespan.

$AVU \leq 1$ **higher is better** — less idle time.

Evaluation Metrics (3/3)

4. Task Satisfaction - ratio between its actual execution time and its expected execution time (the execution time of the task on the fastest available virtual machine):

$$S_i = \frac{AET_i}{EET_i},$$

5. Variance of Fairness (VF) - Task satisfaction dispersion:

$$VF = \frac{1}{n}\sum_{i=1}^{n}(M - S_i)^2 \quad \text{where} \quad M \quad \text{is the mean satisfaction}$$

$VF \leq 1$ **lower is better** — more balanced allocation.

UNIVERSITÀ DEGLI STUDI FIRENZE

Table of Contents

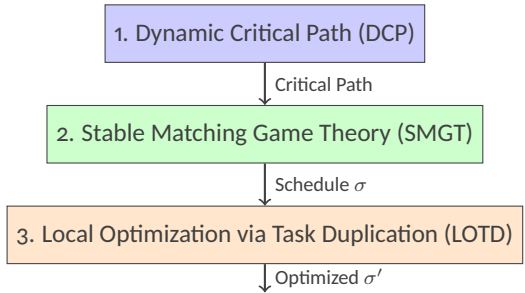
3 SM-CPTD Algorithm

Algorithm Overview

3 SM-CPTD Algorithm

Three-stage pipeline:

1. Identify critical tasks impacting makespan.
2. Fair and efficient task-VM matching.
3. Reduce communication delays and improve scheduling performance, via strategic duplication.



Phase 1: Dynamic Critical Path (DCP)

3 SM-CPTD Algorithm

Goal: Identify tasks that impact the makespan more.

Method: Compute rank values recursively:

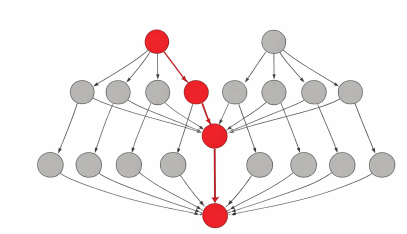
$$rank(t_i) = \overline{W}_i + \max_{t_j \in succ(t_i)} (\overline{c_{i,j}} + rank(t_j))$$

Where:

- $\overline{W}_i$: average computation time of task t_i.
- $\overline{c_{i,j}}$: average communication time between t_i and t_j.
- For exit task: $rank(t_{exit}) = \overline{W}_{exit}$.

Phase 1: Dynamic Critical Path (DCP)

3 SM-CPTD Algorithm



- **Critical Path** is defined by selecting task with maximum rank at each workflow level.
- Critical tasks are scheduled first on *fastest* VMs.

Reducing the execution time of the critical path is the most effective way to lower makespan.

Phase 2: Stable Matching (SMGT)

3 SM-CPTD Algorithm

Two-sided matching problem: Tasks $\leftrightarrow$ VMs

Preference Lists for both tasks and VMs are based on *earliest* estimated finish times.

Hierarchical Strategy:

1. Process workflow level-by-level.
2. Critical tasks assigned to fastest VMs first.
3. Other tasks matched using stable matching.
4. Capacity threshold prevents overloading:

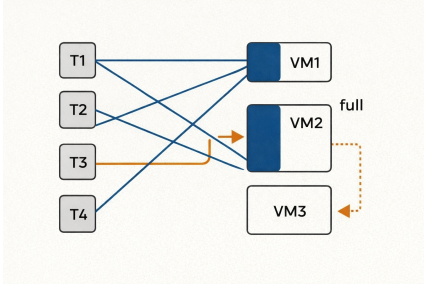
$$threshold(VM_k, l) = \left\lceil \frac{\sum_{v=0}^{l-1} n_v}{\sum_{j=0}^{m-1} p_j} \cdot p_k \right\rceil$$

Phase 2: Stable Matching (SMGT)

3 SM-CPTD Algorithm

Matching Process for each workflow level l of the non-critical tasks:

- Task proposes to most-preferred available VM.
- If VM below threshold: accept task.
- If VM at threshold: compare with worst current task.
- If task preferred: replace; otherwise try next VM.



Result: Fair allocation with balanced VM utilization.

Phase 3: Task Duplication (LOTD)

Goal: Reduce communication overhead by duplicating entry tasks.

Duplication Conditions: Task t_i can be duplicated on VM_k if:

1. At least one successor of t_i is on VM_k.
2. VM_k has sufficient idle time for t_i.
3. There is no increase in completion times of other tasks.

Why Only Entry Tasks?

- No predecessors $\Rightarrow$ no additional input communication.
- Avoids excessive resource waste.

Leverage unused VM capacity to reduce inter-VM communication.

Computational Complexity

3 SM-CPTD Algorithm

Phase	Best Case	Worst Case
Dynamic Critical Path	$O(n)$	$O(n^2)$
Stable Matching	$O(nml)$	$O(n^2 l)$
Task Duplication	$O(nm)$	$O(nm)$
Overall		$O(n^2 l)$

Where:

- n — number of tasks
- m — number of VMs
- l — number of workflow levels

Worst case scalability: quadratic in tasks, linear in levels.

UNIVERSITÀ DEGLI STUDI FIRENZE

Table of Contents

4 Experimental Setup

Implementation and Tools

4 Experimental Setup

Technology Stack:

- **Java 23.0.2** (OpenJDK): Core scheduling algorithms
- **Python 3**: Visualization and statistical analysis
- **Libraries:** NumPy, Pandas, Matplotlib, Seaborn, NetworkX

Design Principles:

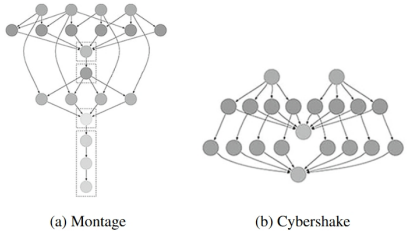
- Modular architecture.
- Clear separation of concerns.
- Reproducible experiments.

Workflow Structures

4 Experimental Setup

Scientific Workflow Considered:

- *Montage*: Astronomy image mosaics (parallel pipelines).
- *CyberShake*: Seismic hazard characterization (multi-level fork-join).
- *LIGO*: Gravitational wave detection (pipeline with synchronization).
- *Epigenomics*: Genomics data processing (parallel with aggregation).



(a) Montage (b) Cybershake

(c) LIGO (d) Epigenomics

Experimental Parameters

4 Experimental Setup

Simulation Setup and Parameters

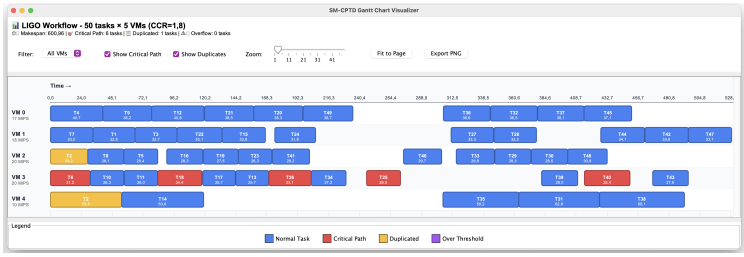
Parameter	Range	Distribution
Task size (MI)	500–700	$\mathcal{U}(500, 700)$
VM capacity (MIPS)	10–20	$\mathcal{U}(10, 20)$
Bandwidth (Mbps)	20–30	$\mathcal{U}(20, 30)$
CCR	0.4–2.0	fixed per experiment

Three **workflow scales** considered:

- **Small-scale**: 50 tasks executed on 5 VMs.
- **Medium-scale**: 100 tasks executed on 10 VMs.
- **Large-scale**: 1000 tasks executed on 50 VMs.

Statistical Reliability

4 Experimental Setup



Execution:

- 10 independent runs per scenario.
- Optional random seed for reproducibility.
- Gantt chart generation for visual inspection.

UNIVERSITÀ DEGLI STUDI FIRENZE

Table of Contents

5 Experimental Results

Communication Impact : SLR vs CCR

Experiment: CCR varies from 0.4 to 2.0 (step = 0.2) across small-, medium-, and large-scale workflow settings.

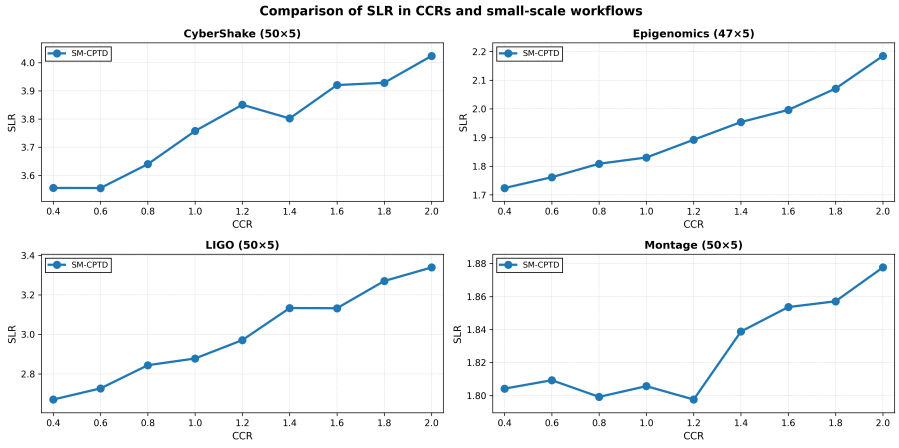
Question: How does increasing communication overhead affect scheduling efficiency?

Observation: SLR increases with higher CCR

- Higher CCR $\Rightarrow$ more communication overhead.
- Larger workflows show smoother trends (higher parallelism).
- Small workflows exhibit more variability.

Communication Impact : SLR vs CCR

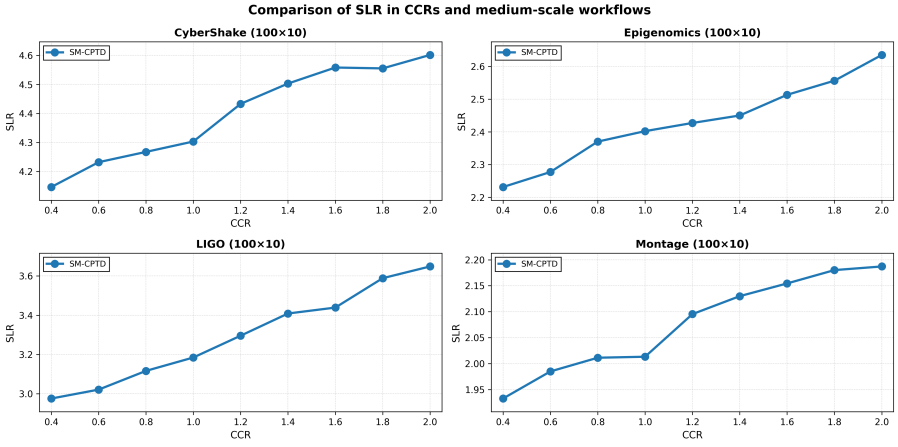
5 Experimental Results



SLR increases with CCR

Communication Impact : SLR vs CCR

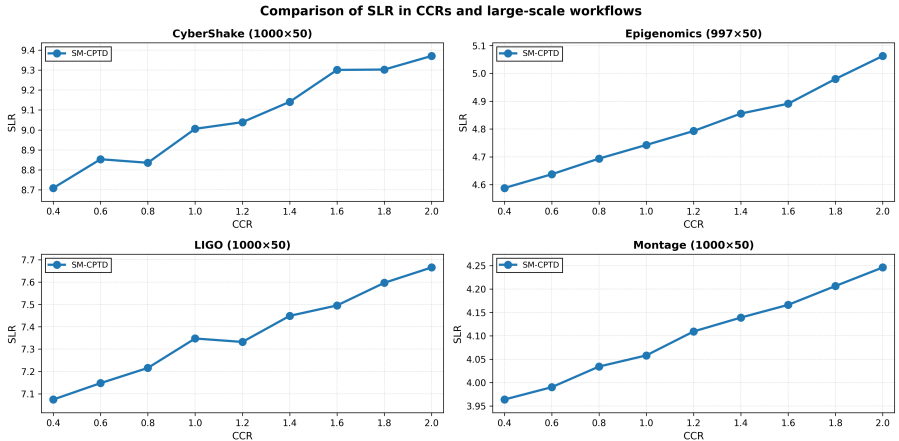
5 Experimental Results



As problem size increases, communication costs have greater impact on performance

Communication Impact : SLR vs CCR

5 Experimental Results



Wide parallel structures suffer most from communication overhead

Key finding

Question: Why do different DAG structures react differently to communication costs?

- **Wide DAGs** (*CyberShake*, *LIGO*): multiple synchronization points, high CCR amplifies delays where parallel branches converge.
- **Sequential DAGs** (*Montage*): overhead affects tasks more uniformly.
- Pipeline structures keep SLR growth more modest.

Workflows with wide parallel structures are more vulnerable to communication bottlenecks.

Scalability: SLR vs Number of VMs

Experiment: Fixed CCR = 1.0, vary number of VMs (from 30 to 70).

Question: Can additional computational resources improve the SLR?

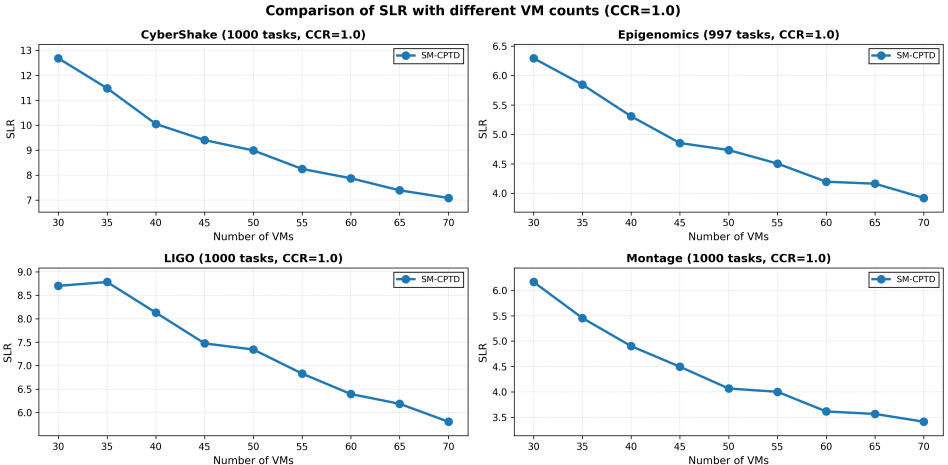
Observation: SLR decreases as VMs increase

- More VMs $\Rightarrow$ better parallelism $\Rightarrow$ lower SLR.
- Plateau reached when parallelism is limited by workflow structure.
- Diminishing returns after certain threshold.

SM-CPTD effectively exploits additional parallelism to approach the critical path bound.

Scalability: SLR vs Number of VMs

5 Experimental Results



VM Utilization Analysis: AVU vs CCR

Experiment: CCR varies from 0.4 to 2.0 (step = 0.2) across small-, medium-, and large-scale workflow settings.

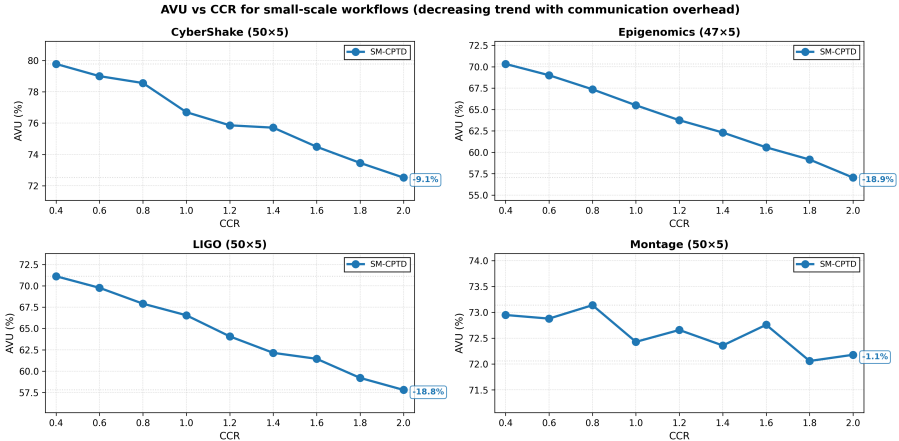
Question: How does communication overhead affect VM utilization?

Observation: AVU decreases with higher CCR.

- Communication costs dominate $\Rightarrow$ increased idle time.
- Effect more pronounced in small-scale workflows.
- Large-scale workflows maintain higher utilization (more parallelism).

VM Utilization Analysis: AVU vs CCR

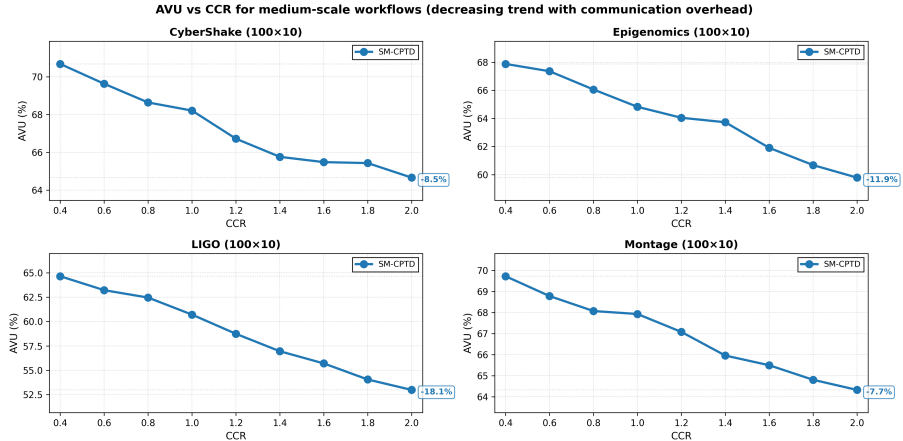
5 Experimental Results



Small-scale workflows show highest sensitivity (up to 19% degradation)

VM Utilization Analysis: AVU vs CCR

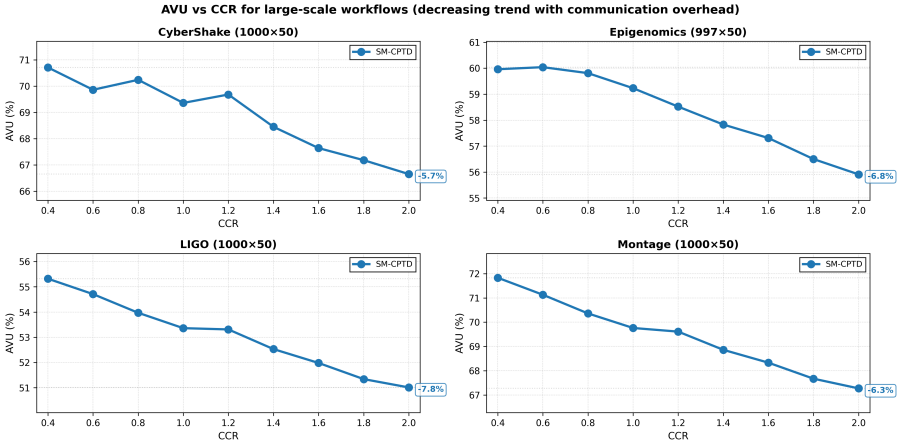
5 Experimental Results



Larger workloads better amortize communication costs

VM Utilization Analysis: AVU vs CCR

5 Experimental Results



Large-scale workflows show more moderate reduction

Scalability: AVU vs Number of VMs

Experiment: Fixed CCR = 1.0, vary number of VMs (from 30 to 70).

Question: Does adding more VMs always improve resource efficiency?

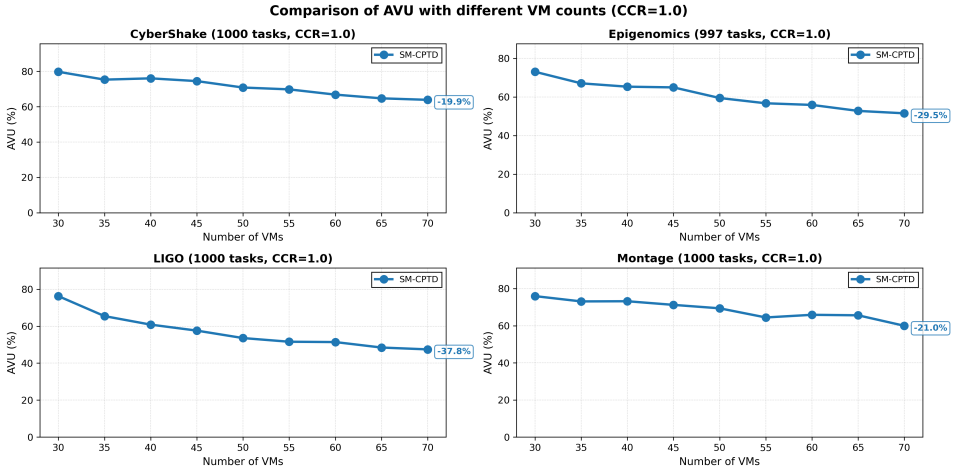
Observation: AVU decreases as the number of VMs increases.

- More VMs $\Rightarrow$ workload spread thinner $\Rightarrow$ lower AVU.
- Execution time per VM decreases relative to total makespan.
- Trend observed consistently across all workflow types.

Adding VMs improves makespan but leads to diminishing utilization efficiency.

AVU vs Number of VMs

5 Experimental Results



Fairness Analysis: VF

5 Experimental Results

Experiment: Large Scale Workflows and fixed CCR=1.

Observation: All workflows exhibit low VF values (well below 1).

- Low dispersion indicates no task is systematically penalized.
- Satisfaction values cluster tightly around the mean.

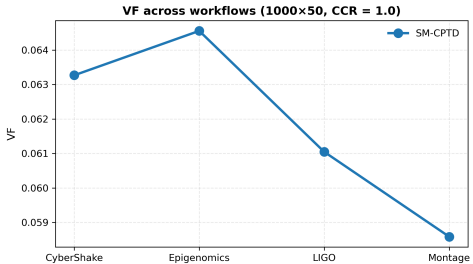


Figure: Variance of Fairness across workflows.

Fairness Analysis

5 Experimental Results

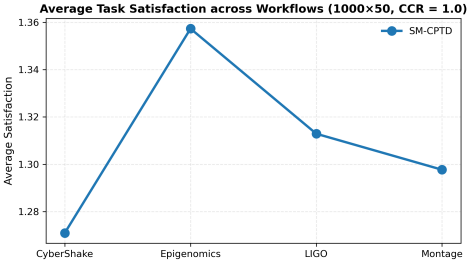


Figure: Average task satisfaction across workflows.

Our Results: VF values < 1 (typically 0–0.1)

Original Paper Issue: Reports VF $\approx$ 1.25–1.47

- Mathematically inconsistent! Variance > 1 impossible given constraints.
- **Hypothesis:** Figure likely shows *average satisfaction*, not variance.
- Our avg. satisfaction values match paper's "VF" range.

Ablation Study

Experiment: Large Scale Workflows and fixed CCR=1. Metrics: VF, SLR, AVU.

Four Configurations

1. **SMGT only** (baseline)
2. **DCP + SMGT** (SM-CP)
3. **SMGT + LOTD** (SM-TD)
4. **DCP + SMGT + LOTD** (full SM-CPTD)

Goal: Evaluate contribution of each component.

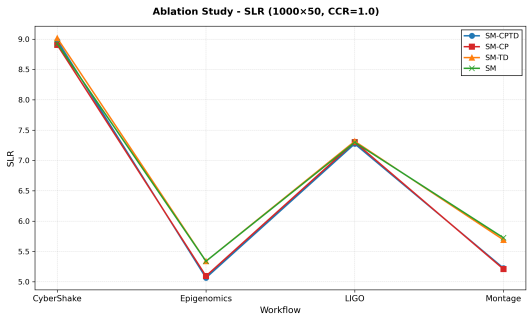
Ablation Study SLR

5 Experimental Results

Question: Which components contribute most to makespan reduction?

Key Findings:

- Full SM-CPTD consistently achieves the lowest SLR.
- Removing CP causes the largest performance degradation.

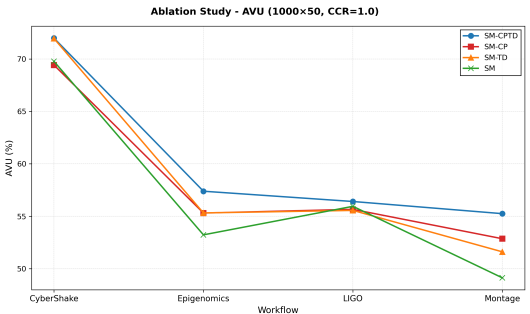


Critical Path optimization is the most critical component for minimizing execution time.

Ablation Study AVU

5 Experimental Results

Question: How do CP and TD interact to improve utilization?



Key Findings:

- Highest AVU reached only when both CP and TD are enabled.
- CP enhances global scheduling efficiency.
- TD improves VM-level utilization through load balancing.

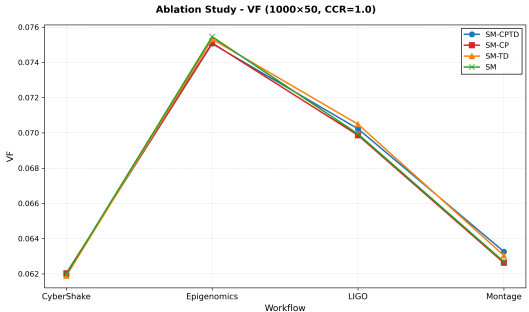
Components are complementary and essential for robustness.

Ablation Study VF

5 Experimental Results

Key Findings:

- VF remains consistently low for all variants.
- Observed differences are small in magnitude.
- Fairness preserved regardless of components.



Interpretation: VF serves as a **sanity check** confirming all variants maintain high fairness. Cannot reliably differentiate between algorithmic variants.

UNIVERSITÀ DEGLI STUDI FIRENZE

Table of Contents

6 Conclusion

Main Contributions

Full reimplementation of SM-CPTD

- Complete Java reimplementation from scratch
- Modular, documented, and extensible architecture
- Publicly released code ensuring full reproducibility

Extensive experimental evaluation

- Validation on standard scientific workflow benchmarks
- Analysis across multiple workflow sizes, CCR values, and VM counts
- Statistical evaluation over multiple independent runs

Limitations and Challenges

Implementation Challenges

- Some algorithmic details not fully specified in original paper
- Required interpretation and design decisions
- Fairness metric definition inconsistency

Experimental Limitations

- Simulated environment (not real cloud deployment)
- Fixed workflow structures from benchmarks
- Uniform distributions for parameters
- Limited to four workflow types

Key Experimental Results: CCR Increase

Performance Trends: SLR vs CCR

- SLR increases for all evaluated workflows.
- Communication-intensive and wide workflows are most impacted.
- Large-scale workflows exhibit smoother and more stable trends.

Resource Utilization: AVU vs CCR

- AVU decreases as communication overhead grows.
- Larger workflows better amortize communication costs.

Key Experimental Results: Scalability Trade-offs

Increasing number of VMs

- **SLR** *decreases* due to higher available parallelism.
- **AVU** *decreases* as idle resources become more frequent.

Key insight

- Clear trade-off between makespan reduction and resource efficiency.
- Scalability benefits strongly depend on workflow structure.

Algorithmic Insights

Critical Path optimization (CP)

- Primary contributor to makespan and SLR reduction.
- Most effective in CP-dominated workflows.

Task Duplication (TD)

- Improves resource utilization and scheduling stability.
- Provides complementary gains rather than dominating performance.

Best performance achieved by combining CP and TD

Final Takeaways

SM-CPTD is an effective workflow scheduling strategy that:

- Reduces makespan-related performance metrics.
- Improves overall resource utilization.
- Maintains stable and fair task allocation behavior.

Code and results publicly available

`https://github.com/Cappetti99/stable-matching-game-theory`

Appendix: References

- adnanetalha. DAX-file: workflows dax format. `https://github.com/adnanetalha/DAX-file`
- Cappetti99, chiarapepp. stable-matching-game-theory. `https://github.com/Cappetti99/stable-matching-game-theory`
- Z. Jia, L. Pan, X. Liu, and X. Li. A novel cloud workflow scheduling algorithm based on stable matching game theory. *The Journal of Supercomputing*, 77(10):11597–11624, 2021. DOI: 10.1007/s11227-021-03742-3.

UNIVERSITÀ DEGLI STUDI FIRENZE

A novel cloud workflow scheduling algorithm based on stable matching game theory

Thank you for listening!