2025 International Conference on Military Communication and Information Systems (ICMCIS) | 979-8-3315-3786-9/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICMCIS64378.2025.11048181

Evaluating Scheduling Algorithms for Adaptive Orchestration in Federated Tactical Edge Cloud Environments

Alessandro Amato*, Harrie Bastiaansen†, Jared Coleman‡, Willem Datema†, Mattia Fogli§,
Johan van der Geest†, Bhaskar Krishnamachari¶, Thomas Kudla‖, Pablo Sanchez**, Niranjan Suri*††,
* *Florida Institute for Human and Machine Cognition (IHMC)*, Pensacola, FL, USA
{aamato, nsuri}@ihmc.org
† *The Netherlands Organisation for Applied Scientific Research (TNO)*, The Hague, The Netherlands
{harrie.bastiaansen, willem.datema, johan.vandergeest}@tno.nl
‡ *Loyola Marymount University*, Los Angeles, CA, USA
jared.coleman@lmu.edu
§ *University of Ferrara*, Ferrara, Italy
mattia.fogli@unife.it
¶ *USC Viterbi School of Engineering*, Los Angeles, CA, USA
bkrishna@usc.edu
‖ *Fraunhofer Institute for Communication, Information Processing and Ergonomics*, Wachtberg, Germany
thomas.kudla@fkie.fraunhofer.de
** *University of Cantabria*, Santander, Spain
sanchez@teisa.unican.es
†† *US Army DEVCOM Army Research Laboratory (ARL)*, Adelphi, MD, USA
niranjan.suri.civ@army.mil

***Abstract*—The IST-193 Research Task Group (RTG) has focused on adapting the popular Kubernetes cloud native platform to operate in military tactical network environments consisting of transient, mobile nodes and intermittent and limited network connectivity. A key focus area for the group has been designing the Federated Adaptive Orchestrator, which is responsible for deciding where services should be run among the tactical cloud environments that are available. This paper focuses on the evaluation of a variety of scheduling algorithms in terms of their performance during the orchestration process and compares them to the performance of a brute-force algorithm that tries to find the most optimal solution.**

This paper was originally presented at the NATO Science and Technology Organization Symposium (ICMCIS) organized by the Information Systems Technology (IST) Scientific and Technical Committee, IST-209-RSY – the ICMCIS, held in Oeiras, Portugal, 13-14 May 2025.

***Index Terms*—Tactical Edge Computing, Federated Orchestration, Scheduling Algorithms**

I. Introduction

Tactical Edge Computing is an approach to process large volumes of data at (or close to) the edge of the Tacical Edge Network (TEN), where the data is generated, in order to prevent the unnecessary transmission of large volumes of raw, potentially irrelevant, data over constrainted tactical networks. To address this problem, the IST-193 Research Task Group (IST-193 RTG) on "*Edge Computing at the Tactical Edge*" has been focused on extending the popular Kubernetes cloud computing platform into an adaptive and federated cloud infrastructure to provide better performance when deployed on TENs [1] [2] [3].

A key contribution of IST-193 RTG has been the development of the Federated Adaptive Orchestrator component (FAO), which makes decisions on where to deploy services (processing tasks) within a TEN, taking into account (on a continuous basis) the availability of compute, memory, and storage resources within instances of Kubernetes clouds, as well as the capacity, latency, and reliability of the communication links between those clouds. This application problem is NP-hard (it can easily be mapped to any number of other NP-hard problems such as bin-packing). For small networks and workflows that involve only a handful processing tasks to be scheduled on the nodes of the federated clouds operating in a TEN, brute-force algorithms (such as those based on Satisfiability Modulo Theories (SMT) [4]) are sufficient as long as the number of nodes and services in the workflow is small, since the execution time is still small compared to the rest of the system and actions that must be taken, such as querying about the resource availability and deploying services. However, as the number of workflows, processing tasks, nodes, communication links, and services increase, the computational cost of the scheduling algorithm can become significant.

This paper examines a number of other algorithms for scheduling processing tasks and services that are more efficient than a brute-force algorithm and presents performance

results as the number of workflows to be deployed/optimized increases. It extends upon previous related work (as described in the following section) with two main contributions:

1) *Enhancement of the IST-193 Orchestration Framework:* The integration of the parametric scheduling code (SAGA [5]) into the FAO [3] enables the use of over one hundred scheduling algorithms currently implemented in SAGA, providing the FAO with significant flexibility to adapt to different mission requirements and optimization objectives.
2) *Experiments:* The results and analysis of extensive experiments to quantify the runtime performance of various scheduling algorithms in TENs. The analysis includes a detailed evaluation of algorithmic components, revealing their impact on performance, and a comparison to a constraint-based SMT solution [4]. Results highlight the practical limitations of SMT solvers with respect to their runtimes, emphasizing the value of heuristic-based scheduling for rapid and reliable decision-making.

This paper is organized as follows. Section II discusses related work, placing the contributions as presented in this paper in the context of existing research on workflow orchestration and job edge scheduling in cloud computing and orchestration frameworks, with a focus on TENs. Subsequently, section III describes the adaptive and federated cloud architecture for TENs as developed by the IST-193 RTG, after which Section IV outlines the challenges of adaptive orchestration in TENs, associated to limited and transient resources, resource-sharing policies, trust dynamics, and deployment costs. Section V presents the formal algorithm scheduling problem definition applicable to two illustrative and representative workflows, evaluates a variety of scheduling algorithms through extensive experiments, and compares and analyzes their performance results under these different workflow scenarios. Finally, section VI concludes the paper and outlines directions for future work.

II. Related Work

Workflow orchestration and job scheduling are well-researched topics in data center and cluster environments, however, literature on tactical edge environments is limited. Therefore, we are extending the related work to standard edge environments.

In Bartolomeo et al. [6], the author proposes Oakestra, a hierarchical orchestration framework for edge computing. This framework comprises three main components organized in a hierarchical structure: the Root Orchestrator, the Cluster Orchestrator, and the Worker Nodes.

The Root Orchestrator serves as a centralized control plane, managing resource clusters, providing high-level coarse control, and facilitating interactions across multiple clusters. The Cluster Orchestrator operates as a logical twin to the Root Orchestrator, managing resources within its local cluster and periodically updating the Root Orchestrator with aggregated statistics, including overall cluster utilization and the health or quality of service (QoS) of deployed services. Lastly, the Worker Nodes represent the edge computational resources tasked with executing services.

Oakestra provides two scheduling approaches. The first, Resource-Only Match (ROM), allows the cluster scheduler to identify resources that meet a service's capacity requirements using a greedy-fit, knapsack-based algorithm. The second approach, Latency & Distance Aware Placement (LDP), extends ROM by incorporating latency and geographical distance considerations to filter deployment options, optimizing service placement.

From an architectural standpoint, Oakestra differs from the IST-193 architecture (as elaborated in the following section) in its reliance on a single centralized control plane. In the event of a Root Orchestrator failure, while existing services can continue to locally replicate, scale, and migrate, Oakestra does not support the registration of new applications or instances in any other cluster.

The IST-193 architecture, on the other hand, in the case of a FAO failure in one cloud, continues to work, with the exception of creating new deployments with the failed FAO, i.e., users of the cloud with the failed FAO, cannot create new deployments. Due to the decentralized and modular approach, other clouds can still utilize all available resources of the cloud with the failed FAO, as long as the other three services continue to work, ensuring continuity while the affected FAO is restored.

From a service scheduling perspective, both Oakestra and the IST-193 framework are built on a modular architecture, allowing users to extend their functionality by integrating custom scheduling policies.

In Edinger et al. [7] the authors introduce a novel decentralized, low-latency task scheduling method for ad-hoc edge computing. Their focus is on interactive applications that require sub-second job completion times to ensure a smooth user experience, such as face detection and recognition, natural language processing, and speech recognition. While the strict timing requirements of these scenarios may not directly apply to the tactical domain, the distributed nature of the approach could be advantageous for purposes in a TEN.

The authors' underlying idea is that, unlike standard approaches that rely on a centralized entity for service orchestration and deployment decisions, their method allows service consumers to perform orchestration at the edge. In this setting, the delays typically caused by sending requests to a centralized orchestrator, the computational time required to find the best scheduling solution, and the time needed for the orchestrator to respond are eliminated, resulting in a faster deployment time. To achieve this, their approach involves having the centralized entity periodically forward a list of available edge resources to the service consumer nodes. Subsequently, the consumer decides where to deploy a service based on a policy designed to minimize resource contention, specifically reducing service rejection due to multiple consumers competing for the same edge resource. The authors propose two policies: "Drift", based on sliding windows, and "Bandit", inspired by the multi-armed stochastic bandit model.

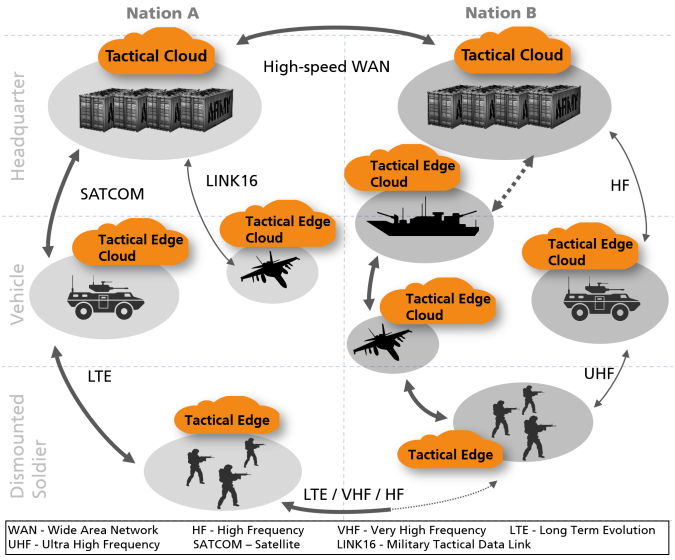

Fig. 1: Overview of the reference operational environment.

In recent years, various machine learning (ML) approaches have been proposed for service scheduling and workflow life cycle orchestration. In Amato et al. [8], a novel decentralized partially observable Markov decision process (DEC-POMDP) formulation and a preliminary study utilizing multi-agent reinforcement learning (MARL) are introduced to perform workflow orchestration over a Tactical Edge Network (TEN). Their work focuses on and is limited to tripartite graph workflows, similar to the hierarchical workflow presented in Figure 3. In their DEC-POMDP formulation, agents receive rewards based on the number of instructions a service can execute in a single step and the time required to transfer data from the head to the tail of the workflow graph (e.g., in the context of Figure 3, from Sensor 1 to Data Fusion 1). The agents aim to maximize the cumulative reward during training, which is conducted using graph convolutional reinforcement learning. This graph-based architecture and training approach enables agents to share information within their K-hop neighborhood while only directly communicating with their first-hop neighbors, making it practical for real-world tactical scenarios.

While a direct performance comparison between scheduling models is infeasible due to differing optimization metrics, exploring potential extensions to the architecture and framework as presented in this paper remains an important direction for future work.

III. Adaptive and Federated Cloud Architecture in TENs

The operational environment of TENs includes centrally deployed tactical cloud systems at headquarters for high-level data processing and situational awareness, and tactical edge clouds for units like vehicles, aircraft, ships, and soldiers. These tactical edge clouds enable localized processing, reduce latency, and support independent operations. The communication infrastructure includes a high-speed WAN linking tactical clouds across nations for data exchange and technologies like SATCOM, LINK16, LTE, VHF, UHF, and HF for connectivity among units at the tactical edge (see Fig. 1).

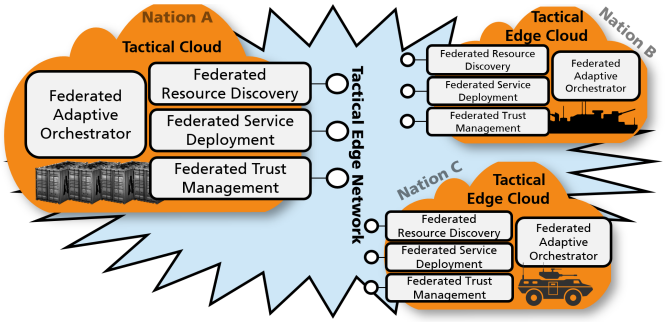

Fig. 2: Platform services.

The developed infrastructure leverages Kubernetes for its orchestration, scalability, and multi-cloud support, with specialized implementations like KubeEdge for edge computing [9]. While Kubernetes lacks native federation and adaptivity for multi-cloud, multi-operator environments, the setup incorporates four platform services developed by the IST-193 RTG to enable these cloud federation capabilities in TENs, whilst preserving each cloud's sovereignty over its infrastructure (see Fig. 2). These services are:

- Federated Resource Discovery (FRD): The FRD aggregates information on available services and resources (e.g., CPUs, RAM, storage) by querying its own cloud and other federated clouds.
- Federated Adaptive Orchestrator (FAO): The FAO plans deployments across the federation and dynamically adapts existing deployments to address changes.
- Federated Service Deployment (FSD): The FSD handles service deployment within and across the federation.
- Trust Resource Management (TRM): The TRM monitors federated cloud activities, gathers security metrics, computes dynamic trust values for partner clouds, and updates policies for the platform services.

IV. Challenges to Adaptive Scheduling and Orchestration across Federated Clouds in TENs

The fundamental scheduling challenge can be summarized as follows: Given a heterogeneous set of partner clouds, some of which lack sufficient resources to handle all information processing locally, the objective is to adaptively orchestrate and schedule information processing tasks across the federation of partner clouds. Adaptiveness involves recognizing either a failure affecting a deployed workflow or a change in circumstances that results in a suboptimal situation. In other words, adaptive orchestration reacts when the desired system state does not match the actual one or when there are opportunities to improve performance. Therefore, adaptive orchestration is a continuous process that ensures the requested workflows are deployed optimally over time. What is considered to be "optimal" may vary depending on the specific partner and mission objectives.

A. Limited, Transient, and Heterogeneous Resources

Unlike traditional cloud environments, computational and network resources at the tactical-operational level are limited. TENs interconnect tactical units in self-organizing wireless multi-hop networks, enabling battlefield communications without relying on pre-existing network infrastructures. However, TENs are characterized by limited bandwidth, intermittent connectivity, and variable latency. Consequently, networking often becomes the primary bottleneck in this scenario.

The dynamic nature of tactical networks introduces another layer of complexity. In fact, resources are transient. This means that resources once allocated may become unavailable due to network partitions, adversarial actions, or energy constraints. This volatility demands orchestration strategies that are resilient and adaptable, eschewing static, one-time deployment models in favor of dynamic, responsive approaches.

Another critical aspect of coalition tactical operations is the heterogeneity of resources, which vary widely in capability - from resource-rich platforms like aircraft, battleships, and vehicles to resource-constrained units such as dismounted soldiers and UAVs. In this context, containerization offers a practical solution, enabling consistent deployment and execution across diverse platforms.

B. Resource Sharing Policies

In coalition tactical operations, multiple administrative domains collaborate, each governed by its own sovereignty concerns over resources. A cloud federation enables coalition partners to share resources within the federation. For example, partner A might agree to share a subset of its resources with partner B. However, partner A may wish to impose restrictions, such as limiting the execution to specific services, capping the amount of CPU and memory allocated to those services, or controlling the network operations those services are permitted to perform. This creates a complex landscape where resource-sharing policies can vary significantly and sometimes conflict. Effective orchestration must navigate these variations, ensuring compliance with local policies while promoting seamless inter-domain collaboration.

C. Trust of Coalition Partners

Coalition partners may not all be equally trusted. For instance, partner A might trust partner B more than partner C. Moreover, trust levels can change throughout a mission due to resource misuse or external circumstances. For example, a previously trusted partner B might lose trust if it repeatedly preempts resources granted to partner A (resource misuse) or enters enemy territory (external circumstance). Trust diminishes in the former case because of repeated delays in service deployment, and in the latter due to an increased likelihood of adversarial attacks. Consequently, partner C may become more trusted than B at some point during the mission, requiring the system to adapt its orchestration decisions accordingly. Reputation and monitoring mechanisms are crucial for estimating trust but come with a performance cost. In resource-constrained tactical networks, this trade-off becomes particularly critical, necessitating a careful balance between resource profiling and resource consumption.

D. Deployment Cost Estimation

Given the limited and transient resources in this context, accurately estimating the cost of a (re)deployment is crucial. An efficient deployment must consider factors often overlooked in traditional cloud environments, which typically assume fast and reliable communication links. For example, the deployment cost should account not only for the time the scheduling algorithm takes to determine a solution (i.e., the deployment plan) but also for the time required to transfer data (e.g., container images) to enable the deployment; the expected runtime of the workflow once deployed (e.g., it would be inefficient to deploy a workflow expected to run for hours on a platform that goes offline every 10 minutes); the time needed to configure the partner clouds hosting the workflow services (e.g., setting up ingress and egress configurations); and the activation time required by the selected partner clouds to initiate the scheduled workflow services.

E. Multi-Workflow Optimization

In a federation, multiple workflows may be deployed concurrently, each with unique resource demands, priorities, and performance objectives. Conflicts can arise when workflows compete for limited resources, requiring prioritization based on mission objectives, partner policies, and trust levels. Furthermore, workflows may have interdependencies, where the execution of one affects the performance or scheduling of others. Adaptive orchestration mechanisms must account for the collective impact of all workflows within the federation. This involves dynamically adjusting deployments to ensure that critical workflows achieve their objectives while minimizing disruptions to others. Multi-workflow optimization necessitates a holistic approach that balances resource efficiency, mission priorities, and the operational constraints of a TEE.

V. Workflow Scenarios, Scheduling Challenges, Results and Assessment of Adaptive Scheduling and Orchestration in TENs

The formalized scheduling problem as considered in this paper directly addresses the real-world challenges faced in TENs, requiring adaptive resource scheduling and orchestration across heterogeneous, constrained, and often dynamic federated clouds, challenged by the constraints as described in the previous section. The subsequent sections in this section subsequently present two illustrative and representative workflow scenarios requiring adaptive scheduling and orchestration over federated clouds in TENs, provide the formal problem job scheduling statement and provides the performance experiments, results and assessment for a broad variety of scheduling algorithms under these different workflow scenarios.

A. Representative workflows

The performance experiments for scheduling algorithms as analyzed in this section are based on two illustrative and

representative mission-related workflow scenarios for scheduling algorithms for adaptive and federated clouds operating in disadvantaged TENS.

In the first scenario, a military surveillance operation secures a forested area using acoustic and tripwire sensors. Acoustic sensors monitor for unusual noises, such as footsteps, vehicle movements, or voices, continuously gathering sound data to detect activity. Tripwire sensors, employing invisible laser beams or pressure-sensitive cables, are positioned at key choke points along trails likely to be used by intruders. When a tripwire is crossed, it triggers an alert pinpointing the exact location and movement direction.

Initially, data from each sensor type is collected by a sensor transformation service, standardizing it into a common format. Subsequently, a data fusion service integrates acoustic detections with tripwire activations. For example, footsteps detected from the north, followed by a tripwire alert on a northern trail, confirm an intrusion and precisely locate the adversary. This fusion significantly enhances situational awareness.

This workflow scenario involves two services for sensor data transformation and a third service for integrating the data into a unified situational picture. Services are deployed hierarchically, with the transformation services positioned close to the sensors due to limited communication range. This is a fan-in type workflow involving four sensors, two transformation tasks, and one final fusion task. It represents a so-called *Hierarchy Workflow*, as illustrated in Figure 3.

The second workflow scenario involves a joint NATO military mission in an urban area utilizing a federated adaptive tactical cloud. A dismounted soldier captures video footage of a suspicious car and sends a report to mission HQ. The video is uploaded to a nearby vehicle in the soldier's national cloud. Due to limited local processing resources, a License Plate Recognition (LPR) service at the HQ processes the video to identify the license plate. The Federated Resource Discovery (FRD) service identifies available resources and services across mission partner clouds, while the Federated Adaptive Orchestrator (FAO) formulates a deployment plan for the LPR service. The results are then returned to the soldier's cloud.

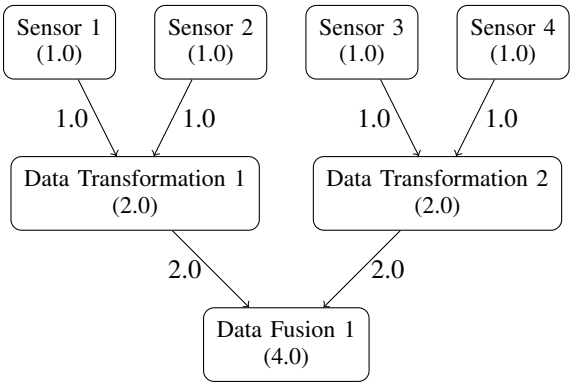

Fig. 3: Hierarchy workflow: task costs are shown in parentheses below the task name and dependency costs (data size) are shown on the edges.

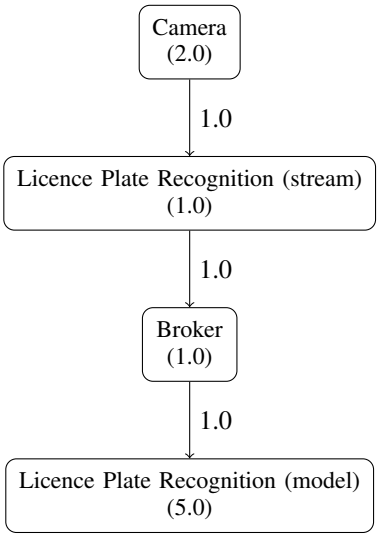

Fig. 4: Chain workflow: task costs are shown in parentheses below the task name and dependency costs (data size) are shown on the edges.

The deployment plan includes a service for extracting frames from the video, a broker to forward the frames, and the LPR service for recognition. It is a sequence of four dependent tasks, executed sequentially. It represents a so-called Chain Workflow, as illustrated in Figure 4.

B. Algorithm scheduling problem statement

The problem, defined over a Directed Acyclic Graph (DAG) [10] of tasks and a network of clouds, encapsulates the core technical issues arising from the operational background: limited network capacity, transient resource availability, and variable node capabilities. Furthermore, it reflects the complexities introduced by human, organizational, and political dimensions, such as resource-sharing policies, trust dynamics among coalition partners, and the need for cost-effective deployment strategies. By formalizing the problem in this manner, we provide a structured approach to analyzing and addressing these challenges.

Formally, we study a scheduling problem defined over a directed acyclic graph (DAG) of tasks $G = (T, D)$, where T represents the set of tasks and D denotes dependencies between them, and a network $N = (V, E)$, where V is the set of nodes and E is the set of communication links. Each task $t \in T$ has a computation cost $c(t)$, and each dependency $d \in D$ has a data transfer cost $c(d)$. The execution time of a task t on a node v is given by $c(t)/s(v)$, where $s(v)$ is the compute speed of the node. Similarly, the time to transfer data between nodes u and v is $c(d)/s(u, v)$, where $s(u, v)$ is the speed of the communication link between the nodes.

We do not explicitly define units for these quantities, as their absolute values are unimportant as long as they remain consistent across the formulation. Instead, the weights serve

as relative measures: $c(t)$ represents computational cost, $c(d)$ quantifies the amount of data to be transferred, $s(v)$ denotes the compute speed of a node (i.e., the rate at which computational cost is processed per unit time), and $s(u, v)$ represents the link speed (i.e., the amount of data that can be transferred per unit time).

Finally, each task $t \in T$ can only be run on a subset $V_t \subseteq V$ of the nodes in the network. Constraints such as trust, resource availability, and policy compliance can be abstracted into the formalism $V_t \subseteq V$, where V_t represents the subset of nodes eligible for running task t. For instance, trust levels can restrict a task to nodes within V_t that are operated by trusted coalition partners. Similarly, resource limitations can exclude nodes lacking the necessary computational or storage capabilities, and policy constraints can enforce execution only on nodes adhering to specific operational guidelines. This abstraction simplifies the incorporation of diverse real-world constraints into the scheduling problem.

A *schedule* for a task graph G on a network N is defined as a set of tuples (t, v, r), where $t \in T$ is the task, $v \in V$ is the node where t is executed, and $r \in \mathbb{R}^+$ is the start time of t. A schedule S is valid if it satisfies the following conditions:

- *Task Assignment:* Each task is scheduled exactly once:

$$\forall t \in T, \exists (t, v, r) \in S,$$
$$\forall (t, v, r), (t', v', r') \in S, t = t' \Rightarrow (v, r) = (v', r').$$

- *Dependency Constraints:* A task cannot start until all its dependencies have completed execution and their outputs have been transferred to the appropriate node:

$$\forall (t, v, r), (t', v', r') \in S, (t, t') \in D,$$
$$r + \frac{c(t)}{s(v)} + \frac{c(t, t')}{s(v, v')} \leq r'.$$

- *Node Constraints:* Every task t is scheduled to run on a permissible node $v \in V_t$
- *No Preemption:* Only one task can run on a node at a time (nodes can be an arbitrarily small computational unit), and tasks cannot be preempted.

Two metrics are used to evaluate the quality of a schedule:

- *Makespan:* The total time required to complete all tasks:

$$m(S) = \max_{(t,v,r) \in S} r + \frac{c(t)}{s(v)}.$$

- *Throughput:* The total amount of work completed per unit time, measured as the reciprocal of the schedule's bottleneck (the maximum busy time of any node or link):

$$t(S) = \frac{1}{\max\{\text{node\_bottleneck}, \text{link\_bottleneck}\}},$$

where

$$\text{node\_bottleneck} = \max_{v \in V} \left\{ \sum_{(t,v,r) \in S} \frac{c(t)}{s(v)} \right\},$$

$$\text{link\_bottleneck} = \max_{(u,v) \in E} \left\{ \sum_{(t,u,r),(t',v,r') \in S} \frac{c(t, t')}{s(u, v)} \right\}.$$

While makespan is suitable for batch-style workflows, throughput is more relevant for continuously running workflows (e.g., long-running services).

We evaluate the two illustrative and representative workflow scenarios as described in the previous paragraph, i.e. the *Hierarchy Workflow* scenario depicted in 3, which is a fan-in type workflow involving four sensors, two fusion tasks, and one final fusion task, and the *Chain Workflow* scenario depicted in 4, consisting of a sequence of four dependent tasks, executed sequentially.

The network model comprises seven homogeneous nodes (speed 1.0) with fully connected links of varying speeds: high-speed (4.0), average-speed (2.0), and low-speed (0.65). Random networks are generated by uniformly selecting link speeds from this set.

The parametric scheduler introduced in [11] enables customizable scheduling by allowing different scheduling algorithms to be composed from individual algorithmic components. Each component controls a specific aspect of scheduling, thereby enabling flexibility to optimize for different workflow characteristics and network conditions.

- *Priority Function:* Determines the order in which tasks are scheduled. The priority function assigns a rank to each task based on workflow dependencies and task execution characteristics. A higher-ranked task is scheduled earlier than a lower-ranked one.
  - *UpwardRank*: This method, as used in HEFT [12], prioritizes tasks based on their longest path to an exit node. Tasks with higher computational and communication costs in their downstream dependencies receive higher priority.
  - *CPoPRank*: Used in CPoP [12], this method prioritizes tasks on the *critical path* (the longest sequence of dependent tasks that would determine the workflow's total execution time if executed on a single processor). By reserving the fastest nodes for these tasks, CPoP aims to minimize delays and optimize overall makespan.
  - *ArbitraryTopological*: A simple heuristic that assigns priorities based on any valid topological ordering of the DAG. While fast and simple, it may yield suboptimal schedules compared to other priority functions.
- *Comparison Function:* Determines which node a given task should be assigned to after it has been selected for scheduling. The function may evaluate the suitability of different nodes based on execution time, availability, and/or network constraints.
  - *EFT (Earliest Finish Time):* Assigns the task to the node that results in the earliest possible completion time.
  - *EST (Earliest Start Time):* Prefers nodes where the task can start execution as soon as possible, without necessarily minimizing total finish time.
  - *Quickest*: Selects the node with the highest computational speed, ignoring network latency and other

constraints.
  - *Minimum Execution Time*: Assigns the task to the node that minimizes its execution time.
  - *Bottleneck*: Prioritizes the assignment that minimizes the throughput impact.
- *Insertion vs. Append Scheduling:* Determines whether tasks can be inserted into existing schedule gaps or must be scheduled in a strictly sequential order.
  - *Insertion-Based*: Allows tasks to be placed in earlier gaps in the schedule if resources become available.
  - *Append-Only*: Ensures that tasks are scheduled strictly in sequence, without attempting to fill gaps.
- *Critical Path Reservation:* Determines whether tasks on the critical path (i.e., the longest sequence of dependent tasks in the workflow DAG) should be given special treatment.
  - *Enabled*: Ensures that critical tasks are scheduled on the fastest available nodes.
  - *Disabled*: Does not prioritize critical tasks, treating all tasks equally in scheduling decisions.
- *Sufferage:* Determines whether to prioritize tasks where the second-best node assignment would result in a significant performance degradation.
  - *Enabled*: Uses the sufferage heuristic [13] to prioritize tasks for which the performance difference between the best and second-best node assignment is highest.
  - *Disabled*: Assigns tasks based purely on their best assignment, without considering the impact of alternative placements.

We extended this parametric framework to support node-level constraints, requiring certain tasks to execute on specified subsets of nodes. This is essential in federated cloud environments where security policies, resource ownership, or trust levels may dictate which tasks can run on which nodes.

To evaluate the performance of different scheduling strategies, we systematically varied the parameters across 120 unique scheduler configurations. Specifically, we tested the following combinations:

- *Priority Function:* Three options (UpwardRank, CPoPRank, ArbitraryTopological).
- *Comparison Function:* Five options (EFT, EST, Quickest, Minimum Execution Time, Bottleneck).
- *Insertion vs. Append Scheduling:* Two options (Append-Only True/False).
- *Critical Path Reservation:* Two options (Enabled/Disabled).
- *Sufferage:* Two options (Enabled/Disabled).

These configurations were tested across both workflow scenarios (Hierarchy Workflow and Chain Workflow) and evaluated using two performance metrics:

- *Makespan:* Measures the total time required to complete all tasks in the workflow.
- *Throughput:* Measures the total number of tasks completed per unit time.

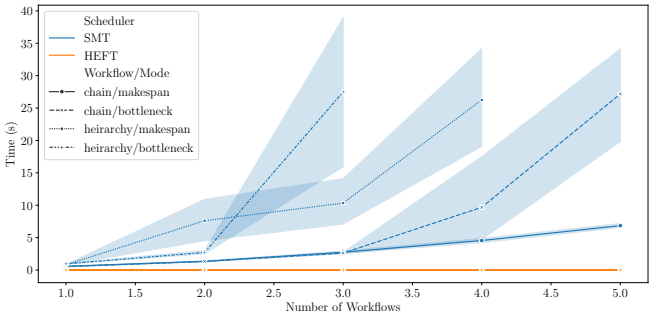

Fig. 5: Comparison of SMT scheduler and HEFT runtimes for chain and hierarchy workflows, under makespan minimization and throughput maximization objectives. The X-axis represents the number of workflows, and the Y-axis shows runtime in seconds. The shaded regions indicate the standard deviation, representing variability in runtime across different instances.

For benchmarking, we compared these schedulers against two additional algorithms:

- *Baseline Scheduler:* A non-optimized scheduler that produces valid schedules without considering execution time or resource constraints.
- *SMT Scheduler:* A constraint-based approach using an SMT solver with binary search to find an ε-optimal schedule. This approach produces near-optimal schedules but has high computational overhead, making it impractical for large-scale problems.

Since the scheduling problem is NP-hard [14], the runtime of the SMT scheduler can grow exponentially with the problem size. Figure 5 compares the runtime of the SMT scheduler with HEFT [12], one of 120 schedulers evaluated (i.e., UpwardRank, EFT, Insertion-Based, No Critical Path Reservation, No Sufferage), across ten randomly generated problem instances for each workflow type and optimization objective (makespan minimization and throughput maximization). Problem size is varied by increasing the number of workflows to schedule (1 to 5). A one-minute timeout was set for each evaluation.

As shown in Figure 5, the SMT scheduler's runtime increases rapidly with problem size, in contrast to HEFT's negligible runtime. Notably, for throughput optimization, the SMT scheduler times out when scheduling just three hierarchy workflows.

C. Results

For makespan minimization on the chain workflow, results indicate that the comparison function was the only algorithmic component to significantly impact performance (Figure 6). The EFT and EST comparison functions outperformed the Quickest, Minimum Execution Time, and Bottleneck functions. Results do also indicate that there is an interesting interaction between the comparison function and the critical path reservation component, with non-critical path reservation performing significantly better with the EFT and EST

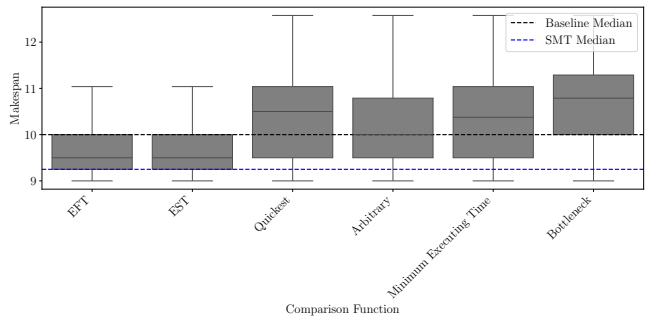

Fig. 6: Comparison of comparison functions for makespan minimization on the chain workflow. The dashed black line represents the median makespan of the Baseline algorithm and the blue dashed line represents the median near-optimal makespan found using the SMT scheduler.

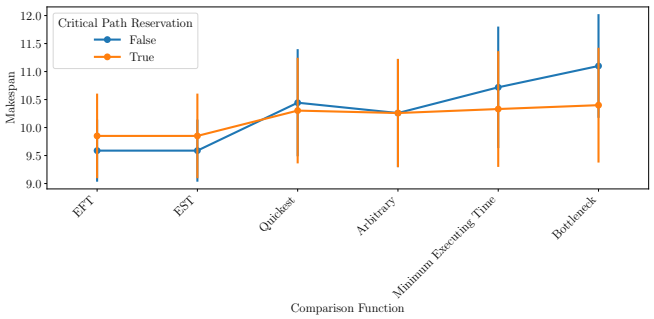

Fig. 7: Interaction between the Critical Path Reservation and Comparison Function algorithmic components on Average Makespan in the Chain Workflow. The plot illustrates how makespan varies across different comparison functions, with and without critical path reservation.

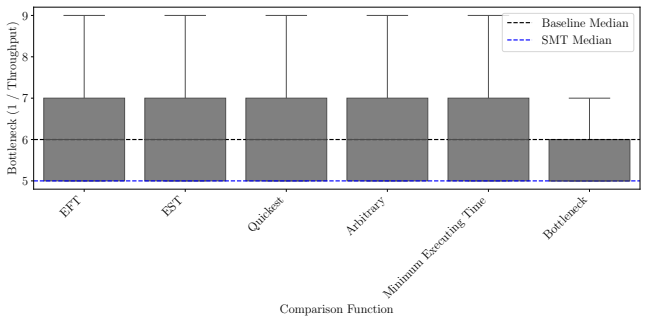

Fig. 8: Comparison of comparison functions for throughput maximization on the chain workflow. The dashed black line represents the median makespan of the Baseline algorithm and the blue dashed line represents the median near-optimal makespan found using the SMT scheduler.

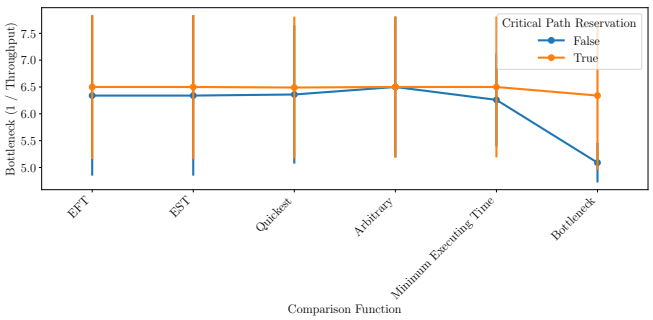

Fig. 9: Interaction between the Critical Path Reservation and Comparison Function algorithmic components on Throughput Maximization in the Chain Workflow. The plot illustrates how throughput varies across different comparison functions, with and without critical path reservation.

comparison functions and critical path reservation performing significantly better with the Quickest, Minimum Execution Time, and Bottleneck comparison functions (Figure 7).

For throughput maximization (bottleneck minimization) on the chain workflow, results again indicate that the comparison function was the only algorithmic component to significantly impact performance (Figure 8). The Bottleneck comparison function outperformed the other comparison functions, especially when combined with no critical path reservation (Figure 9).

For makespan minimization on the hierarchy workflow, results indicate that the comparison function was the only algorithmic component to significantly impact performance (Figure 10). The EFT and EST comparison functions outperformed the Quickest, Minimum Execution Time, and Bottleneck functions. Results also indicate that there is an interesting interaction between the comparison function and the critical path reservation component, with non-critical path reservation performing significantly better with the EFT and EST comparison functions and critical path reservation performing significantly better with the Quickest, Minimum Execution Time, and Bottleneck comparison functions (Figure 11).

For throughput maximization (bottleneck minimization) on the hierarchy workflow, results again indicate that the comparison function was the only algorithmic component to significantly impact performance (Figure 12). The Bottleneck comparison function outperformed the other comparison functions, especially when combined with no critical path reservation (Figure 13).

D. Discussion

The findings highlight that the performance of scheduling algorithms is predominantly influenced by the choice of comparison function, with its interaction with the critical path reservation component playing a substantial role. Conversely, the priority function, insertion versus append scheduling strategies, and Sufferage components appear to have minimal or negligible impact on schedule quality.

For makespan minimization, the EFT and EST comparison functions consistently outperformed other approaches. This result is consistent across both the chain and hierarchy workflows. Interestingly, the performance of these functions

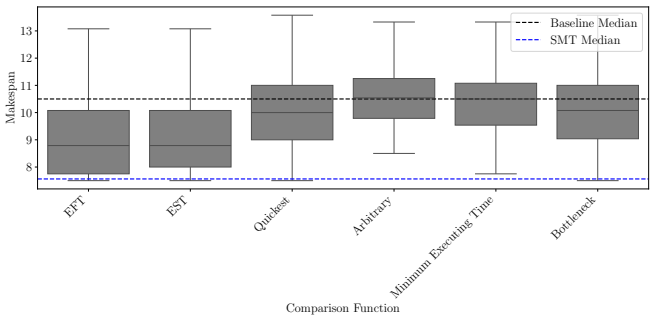

Fig. 10: Comparison of comparison functions for makespan minimization on the hierarchy workflow. The dashed black line represents the median makespan of the Baseline algorithm and the blue dashed line represents the median near-optimal makespan found using the SMT scheduler.

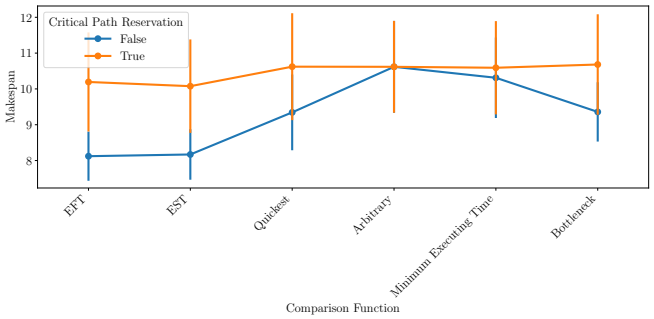

Fig. 11: Interaction between the Critical Path Reservation and Comparison Function algorithmic components on Makespan Minimization in the Hierarchy Workflow. The plot illustrates how makespan varies across different comparison functions, with and without critical path reservation.

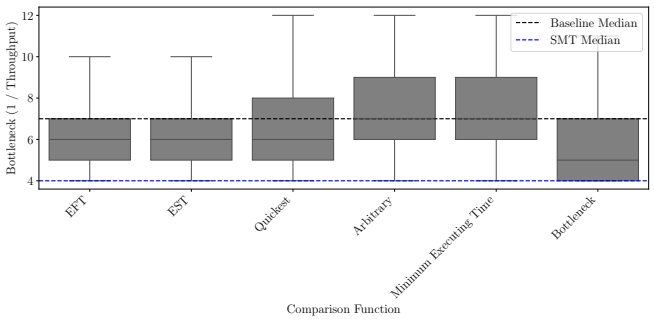

Fig. 12: Comparison of comparison functions for throughput maximization on the hierarchy workflow. The dashed black line represents the median makespan of the Baseline algorithm and the blue dashed line represents the median near-optimal makespan found using the SMT scheduler.

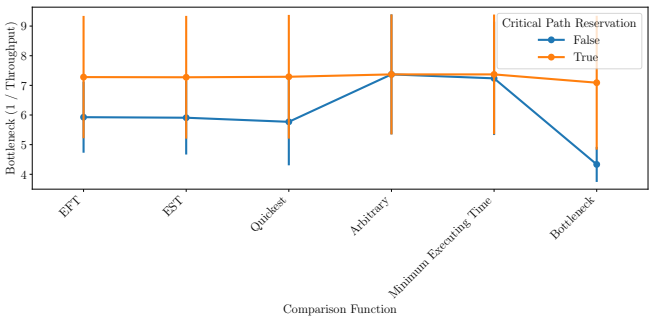

Fig. 13: Interaction between the Critical Path Reservation and Comparison Function algorithmic components on Throughput Maximization in the Hierarchy Workflow. The plot illustrates how throughput varies across different comparison functions, with and without critical path reservation.

was further enhanced when no critical path reservation was used. On the other hand, comparison functions like Quickest and Minimum Execution Time demonstrated improved performance under critical path reservation, suggesting a nuanced interplay between these components.

In the case of throughput maximization (bottleneck minimization), the Bottleneck comparison function emerged as the most effective, significantly outperforming alternatives. This success was particularly pronounced when no critical path reservation was applied. Surprisingly, the Minimum Execution Time function, despite its intuitive alignment with throughput maximization, exhibited worse or similar performance to the EFT and EST functions.

These findings demonstrate the advantages of the parametric scheduling approach in tailoring algorithms to complex scheduling problems. By allowing the systematic exploration of algorithmic components, this framework enables the identification of configurations that align well with distinct optimization objectives, such as makespan minimization or throughput maximization for application-specific scenarios.

The experimental results directly address the core technical challenges outlined in the operational background, demonstrating the applicability of algorithmic scheduling solutions to tactical environments characterized by limited network capacity, transient resource availability, and heterogeneous node capabilities. The demonstrated effectiveness of scheduling algorithms designed for less constrained environments, combined with extensions for node-level constraints, highlights their adaptability to these challenging settings.

The inefficiency of constraint satisfaction problem (CSP) solvers (e.g., the SMT Scheduler) in scenarios requiring frequent rescheduling, as evidenced by their prohibitive runtimes, emphasizes the need for lightweight algorithms. Tactical networks demand algorithms that can handle rapid changes in resource availability without significant delays. The explored configurations in this study, particularly those leveraging comparison functions like EFT and Bottleneck, provide practical alternatives that balance performance with the adaptability

required in dynamic operational contexts. These findings underscore the importance of adaptive orchestration strategies that align with the real-world constraints of federated tactical environments.

VI. Conclusion & Future Work

This paper has presented a performance assessment of various algorithms for adaptive scheduling. The focus has been the scheduling of processing tasks in disadvantaged TENs for the adaptive and federated cloud architecture developed by the NATO IST-193 RTG.

While the evaluated scheduling algorithms provide a functional basis for workflow resource (re)allocation, several areas for improvement remain and present opportunities for future research. Therefore, as follow-up work of the IST-193 RTG, various aspects of adaptivity in TENs must be further explored.

Presently, the algorithm operates as a single-shot mechanism, where allocations are reattempted only upon failure and without any optimization across workflows. This sequential approach lacks the ability to optimize resources collectively for multiple workflows or to prioritize among them. Moreover, it does not account for the costs associated with deployment, such as service installation, or the potential risks and costs tied to resource utilization, including communication links and compute resources in battery-constrained devices.

The current system also operates without memory, meaning it does not retain information about prior resource usage or the performance of previous allocations. This limitation prevents the algorithm from learning from past experiences to improve future decisions. Additionally, the absence of preemption capabilities means workflows cannot be interrupted or adjusted to accommodate higher-priority tasks that may emerge. The algorithm does not adapt to changes in resource availability if workflows remain operational, even if they do so suboptimally. Furthermore, the process of reallocation lacks a framework for evaluating the "cost of re-deployment," which could lead to inefficient adjustments when resources shift.

Addressing these shortcomings would enhance the system's robustness, efficiency, and adaptability. Incorporating mechanisms for cross-workflow optimization, cost and risk sensitivity, prioritization, and memory could significantly improve performance. Furthermore, introducing preemption and dynamic reallocation capabilities, while considering the costs of redeployment, would create a more resilient and responsive allocation system, with better overall performance. These advancements would represent a substantial step forward in the development of efficient and intelligent resource management solutions.

Acknowledgment

The work presented in this article is being done as part of the IST-193 RTG "Edge Computing at the Tactical Edge". We would like to thank the NATO IST-Panel for providing us the opportunity to do this highly relevant and interesting research and the participating partners for providing valuable inputs within a stimulating and cooperative setting.

This work was supported in part by Army Research Laboratory under Cooperative Agreement W911NF-17-2-0196.

References

[1] H. Bastiaansen, J. v. d. Geest, C. v. d. Broek, T. Kudla, A. Isenor, S. Webb, N. Suri, M. Fogli, B. Canessa, A. Masini, R. Goniacz, and J. Sliwa, "Federated Control of Distributed Multi-Partner Cloud Resources for Adaptive C2 in Disadvantaged Networks," *IEEE Communications Magazine*, vol. 58, no. 8, pp. 21–27, 2020.

[2] T. Kudla, M. Fogli, S. Webb, G. Pingen, N. Suri, and H. Bastiaansen, "Quantifying the Performance of Cloud-Oriented Container Orchestrators on Emulated Tactical Networks," *IEEE Communications Magazine*, vol. 60, no. 5, pp. 74–80, 2022.

[3] A. Amato, H. Bastiaansen, W. Datema, M. Fogli, R. Fronteddu, R. Galliera, J. van der Geest, T. Kudla, P. Sanchez, N. Suri, and S. Watson, "A Performance Cost/Benefit Analysis of Adaptive Computing in the Tactical Edge," in *2024 International Conference on Military Communication and Information Systems (ICMCIS)*, 2024, pp. 1–8.

[4] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli, "Satisfiability Modulo Theories," in *Handbook of Satisfiability - Second Edition*, ser. Frontiers in Artificial Intelligence and Applications, A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds. IOS Press, 2021, vol. 336, pp. 1267–1329. [Online]. Available: https://doi.org/10.3233/FAIA201017

[5] J. Coleman, B. Krishnamachari, S. Adapala, R. Agrawal, and E. Hirani, "Scheduling Algorithms Gathered," Github, 2023. [Online]. Available: https://github.com/ANRGUSC/saga

[6] G. Bartolomeo, M. Yosofie, S. Bäurle, O. Haluszczynski, N. Mohan, and J. Ott, "Oakestra: A Lightweight Hierarchical Orchestration Framework for Edge Computing," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. Boston, MA: USENIX Association, Jul. 2023, pp. 215–231. [Online]. Available: https://www.usenix.org/conference/atc23/presentation/bartolomeo

[7] J. Edinger, M. Breitbach, N. Gabrisch, D. Schäfer, C. Becker, and A. Rizk, "Decentralized Low-Latency Task Scheduling for Ad-Hoc Computing," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2021, pp. 776–785.

[8] A. Amato, A. Morelli, M. Fogli, R. Galliera, and N. Suri, "Multi-Agent Reinforcement Learning for Distributed Workflow Orchestration at the Tactical Edge," in *MILCOM 2024 - 2024 IEEE Military Communications Conference (MILCOM)*, 2024, pp. 64–69.

[9] M. Fogli, T. Kudla, B. Musters, G. Pingen, C. Van den Broek, H. Bastiaansen, N. Suri, and S. Webb, "Performance Evaluation of Kubernetes Distributions (K8s, K3s, KubeEdge) in an Adaptive and Federated Cloud Infrastructure for Disadvantaged Tactical Networks," in *2021 International Conference on Military Communication and Information Systems (ICMCIS)*, 2021, pp. 1–7.

[10] Wikipedia, "Directed acyclic graph — Wikipedia, the free encyclopedia," http://en.wikipedia.org/w/index.php?title=Directed\%20acyclic\%20graph&oldid=1261432631, 2024, [Online; accessed 23-December-2024].

[11] J. R. Coleman, R. V. Agrawal, E. Hirani, and B. Krishnamachari, "Parameterized Task Graph Scheduling Algorithm for Comparing Algorithmic Components," *CoRR*, vol. abs/2403.07112, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2403.07112

[12] H. Topcuoglu, S. Hariri, and M. Wu, "Task Scheduling Algorithms for Heterogeneous Processors," in *8th Heterogeneous Computing Workshop, HCW 1999, San Juan, Puerto Rico, April12, 1999*. IEEE Computer Society, 1999, pp. 3–14.

[13] T. N'Takpé and F. Suter, "Critical Path and Area Based Scheduling of Parallel Task Graphs on Heterogeneous Platforms," in *12th International Conference on Parallel and Distributed Systems, ICPADS 2006, Minneapolis, Minnesota, USA, July 12-15, 2006*. IEEE Computer Society, 2006, pp. 3–10.

[14] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.