NextGen Scientific Publication
American International Journal of Computer Science and Technology

Volume 3 Issue 6, Pg. No. 11-22, AIJCST-V3I6P102, 2021
https://doi.org/10.63282/3117-5481/AIJCST-V3I6P102

Original Article

Multi-Agent Systems for Distributed Task Scheduling in High-Performance Computing

*** Dr. Bastin Thiyagaraj**
Department of IT, St. Joseph's College (Autonomous), Tiruchirappalli, Tamil Nadu, India.

Abstract:

High-performance computing (HPC) workloads increasingly span heterogeneous resources, tight deadlines, and fluctuating queue conditions, making centralized schedulers brittle under scale and uncertainty. This paper proposes a multi-agent systems (MAS) framework for distributed task scheduling that decomposes global scheduling into local, goal-driven decisions coordinated through lightweight protocols. A hierarchical agent ecology comprising a global coordinator, cluster brokers, and node-level executors negotiates task placement using a hybrid of Contract Net–style auctions for rapid allocation and game-theoretic policies for fairness and congestion control. Agents maintain beliefs over queue dynamics, accelerator availability, and network contention, updating them via online learning to predict task runtimes and co-scheduling effects (CPU–GPU/TPU). The framework integrates energy-aware policies, data-locality heuristics, and fault tolerance through checkpoint-informed replication and resubmission. To mitigate stragglers and tail latency, agents apply hedged dispatch selectively based on risk estimates from historical traces. We further introduce a control-loop that couples short-horizon reinforcement learning for local actions with constraint-driven global guards to preserve service-level objectives and job priority policies. Evaluations on trace-driven simulations and a heterogeneous testbed demonstrate improved makespan, throughput, and energy efficiency versus centralized and static distributed baselines, while preserving fairness under bursty arrivals. The results suggest MAS scheduling offers a robust, adaptive path for next-generation HPC centers operating at exascale and beyond.

Keywords:

Multi-Agent Systems, Distributed Scheduling, High-Performance Computing, Contract Net Protocol, Reinforcement Learning, Heterogeneous Clusters, Energy-Aware Scheduling, Data Locality, Fault Tolerance, Tail-Latency Mitigation.

Article History:

Received: 22.09.2021

Revised: 19.10.2021

Accepted: 01.11.2021

Published: 07.11.2021

1. Introduction

High-performance computing (HPC) has become the backbone of contemporary science and engineering, powering applications from climate modeling and genomics to fluid dynamics and large-scale AI training. As systems approach exascale, their resources are increasingly heterogeneous mixing CPUs with diverse accelerators (GPUs, TPUs), non-uniform memory hierarchies, high-speed interconnects, and energy-aware power caps. Traditional centralized schedulers, while mature, face mounting pressure under this

complexity: global queue backlogs grow, single control planes become bottlenecks, and coarse heuristics struggle to capture rapidly changing contention and data-locality constraints. The result is suboptimal makespan, inflated tail latency, and fragile performance under bursty arrivals or partial failures.

Multi-agent systems (MAS) offer a principled alternative. By decomposing global scheduling into decentralized, goal-oriented decisions, MAS can align local autonomy with global objectives through negotiation, learning, and lightweight coordination protocols. In this work, we propose a hierarchical MAS framework in which a global coordinator sets policy bounds, cluster-level brokers negotiate allocations, and node-level executors enact fine-grained placement with awareness of co-scheduling effects and I/O topology. Agents maintain beliefs about queue dynamics and resource states, updating them online using predictive models to anticipate runtimes, congestion, and energy costs. A hybrid of Contract Net–style auctions and game-theoretic fairness guards accelerates allocation while preventing starvation. To handle uncertainty, the framework integrates checkpoint-aware replication, selective hedging for stragglers, and constraint-shielded reinforcement learning that preserves service-level objectives. Through trace-driven experiments on heterogeneous testbeds, we demonstrate that MAS-based scheduling improves throughput, reduces makespan and tail latency, and enhances energy efficiency relative to centralized and static distributed baselines, positioning MAS as a robust foundation for next-generation HPC schedulers.

2. Related Work

2.1. Overview of Task Scheduling in HPC

Classic HPC schedulers (e.g., PBS, SLURM, LSF) evolved around batch-queue abstractions with objectives such as minimizing makespan, maximizing utilization, and enforcing priority/fair-share policies. Heuristic strategies backfilling, gang scheduling, and topology-aware placement remain standard because they are predictable and administratively simple. As clusters became heterogeneous, schedulers incorporated resource descriptors (GPU counts, memory bandwidth, interconnect topology) and job constraints (NUMA awareness, license tokens), as well as power and thermal caps. Data-aware scheduling emerged to account for I/O hotspots and burst buffers, while malleable and moldable jobs challenged static reservation assumptions. Despite these advances, centralized designs struggle to track fine-grained, fast-changing states (e.g., accelerator contention, shared-file-system pressure), and global optimization often relies on coarse or stale signals.

2.2. Distributed Scheduling Models

Distributed scheduling literature spans two broad families. The first decentralizes decision-making across multiple cooperating queues/brokers (federated metaschedulers, broker–executor patterns), emphasizing scalability and fault isolation. The second uses two-level or market-inspired schedulers: a global allocator assigns capacity slices, while local schedulers map tasks to nodes based on local utility. Work stealing, randomized load balancing (“power of two choices”), and hierarchical queueing reduce coordination overheads and smooth bursty arrivals. In heterogeneous environments, locality-aware and topology-aware variants reduce cross-switch traffic and PCIe/NVLink contention. However, distributed schedulers must manage consistency (e.g., double-booking avoidance), global fairness, and unstable feedback loops when partial views of the system drive greedy local choices. Hybrid controllers that combine slow, global planning with fast, local heuristics have shown promise, but they often lack mechanisms for learning and negotiation among autonomous components.

2.3. Agent-Based Computing Approaches

Agent-based paradigms view schedulers as populations of autonomous entities with beliefs, goals, and plans. Early MAS schedulers adopted the Contract Net Protocol (CNP) for task announcement, bidding, and award, enabling rapid, decentralized matching. Subsequent work integrated utility functions encoding runtime, energy, and data movement costs, while coalition formation and mediator agents handled resource pooling for gang-parallel jobs. Learning-augmented agents using online regression, multi-armed bandits, or reinforcement learning improved runtime prediction, contention estimation, and hedging decisions. More recent studies introduce hierarchical agent ecologies (global policy agents, cluster brokers, node executors) to balance autonomy with system-wide constraints, as well as mechanism-design elements (budgeted auctions, VCG-style incentives) to deter selfish behavior and ensure fairness. Compared with fixed heuristics, MAS approaches excel under uncertainty and non-stationarity, adapting quickly to workload changes, hardware faults, and transient I/O bottlenecks.

2.4. Limitations of Existing Methods

Despite progress, four gaps persist. First, stale or myopic state: centralized schedulers aggregate coarse metrics, while many distributed models still depend on delayed telemetry, leading to suboptimal placements and tail-latency inflation. Second, limited heterogeneity modeling: few systems capture fine-grained co-scheduling effects e.g., GPU memory oversubscription, NUMA cross-traffic, or shared filesystem amplification within their decision loops. Third, fragile fairness under burstiness: heuristics like backfilling or greedy work stealing can starve low-priority or data-heavy jobs without explicit market or game-theoretic guarantees. Fourth, resilience and risk: replication, checkpointing, and hedged execution are often bolted on rather than integrated with scheduling policies, leading to wasted energy or redundant I/O during failures and straggler events. Finally, policy–learning misalignment: RL-enhanced schedulers can drift from SLOs when rewards omit governance and admission-control constraints. These limitations motivate a MAS design that (i) learns calibrated performance/energy models online, (ii) negotiates via lightweight markets for fairness and congestion control, and (iii) embeds safety guards so local learning cannot violate global policies.

3. System Architecture and Design

3.1. Multi-Agent Framework Overview

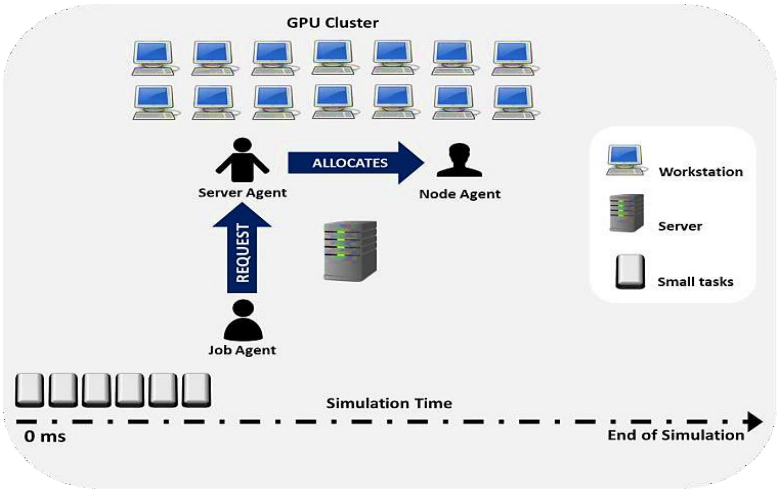

Figure 1. Multi-Agent Scheduling Architecture over A GPU Cluster and Simulation Timeline

The figure depicts a hierarchical multi-agent architecture tailored for scheduling tasks on a heterogeneous GPU cluster. At the bottom, the simulation timeline emphasizes that the system operates continuously, with jobs arriving over time rather than in a single batch. Each arriving job is encapsulated as a Job Agent that formulates a request describing its resource needs such as GPU count, memory footprint, expected runtime, and any data-locality or affinity constraints. This request is submitted to a Server Agent, which maintains global policy context, queue state, and admission control rules.

Upon receiving a request, the Server Agent initiates an allocation protocol with Node Agents that represent individual worker nodes in the GPU cluster. The bold “ALLOCATES” arrow captures a Contract Net–style interaction: nodes advertise their current capacities and contention outlooks, while the server evaluates bids according to utility functions that balance makespan, fairness, and energy objectives. This division of labor allows fast, local decisions at the node level while preserving global constraints through the server’s policy guardrails.

The cluster sketch at the top symbolizes a pool of accelerator-equipped machines behind a high-speed interconnect, while the legend at right differentiates workstations, servers, and small tasks. By distinguishing these roles, the diagram highlights how heterogeneous resources are abstracted uniformly by Node Agents. As jobs progress, Node Agents update the Server Agent with live telemetry queue lengths, GPU availability, and I/O pressure allowing subsequent allocations to adapt to shifting conditions without global bottlenecks.

Finally, the left-to-right timeline links the agent interactions to observable system behavior across the entire run. It makes explicit that the architecture supports dynamic arrivals, straggler handling, and policy evolution as the simulation advances. In

practice, this means the model can incorporate checkpoint-aware resubmissions, selective hedging, or energy-aware throttling over time capabilities that are difficult to express in static, centralized schedulers but arise naturally in a multi-agent setting.

3.2. Agent Roles and Interactions

In the proposed hierarchy, Job Agents encapsulate individual jobs or DAG stages, translating user constraints GPU count, memory, runtime SLOs, preemption tolerance, data locality into a negotiable bid. They remain active throughout execution, monitoring progress, triggering mid-run adaptations (e.g., shrinking or expanding parallelism), and negotiating resubmission on failure using checkpoint metadata. Server Agents (or cluster brokers) steward global policies: priority and fairness rules, admission control, energy budgets, and congestion caps for shared subsystems such as the parallel file system or interconnect. They do not micromanage placement; instead, they orchestrate market rounds and guardrails that shape local decisions. Node Agents represent individual worker nodes (CPU/GPU sockets, NUMA domains, attached storage), exposing a local view current utilization, queued kernels, PCIe/NVLink pressure and committing to allocations they can honor without violating thermal or power envelopes.

Interactions follow a rhythm of announce–bid–award with continual feedback. A Job Agent submits a request to the Server Agent, which issues a call-for-proposals to candidate Node Agents filtered by feasibility (resource vectors, topology reachability, data proximity). Node Agents reply with bids that embed predicted runtime, interference risk, and energy cost under their current mix. The Server Agent adjudicates using a utility that respects job priority and cluster SLOs, awards a lease to the best set of nodes, and instructs the Job Agent to dispatch tasks. During execution, Node Agents stream telemetry to both Job and Server Agents; if drift is detected stragglers, thermal throttling, I/O hotspots the Job Agent may renegotiate or trigger hedged replicas subject to broker policies.

3.3. Communication and Coordination Mechanisms

Communication adopts lightweight, typed messages with strict timeouts to avoid head-of-line blocking: requests, proposals, awards, heartbeats, revocations, and telemetry updates. To preserve scalability, control traffic is gossip-aggregated and rate-limited; Node Agents batch telemetry at millisecond-to-second granularity and publish deltas rather than full state. A two-plane design separates the slow policy plane from the fast allocation plane. The policy plane disseminates quotas, fairness budgets, and safety constraints at lower frequency, while the allocation plane executes rapid bidding rounds and placement commits. This separation reduces the blast radius of transient control spikes and lets the system sustain high job-arrival rates without saturating the broker.

Coordination blends Contract Net Protocol (CNP) with distributed consistency guards. Soft-state leases prevent double booking; an atomic commit token is attached to each award, and Node Agents validate tokens before enacting placements. For cross-node gang tasks, a temporary coalition leader is elected to coordinate synchronized starts and checkpoint barriers. Failure handling is message-driven: missed heartbeats or expiring leases cause automatic revocation and rebidding, while idempotent replay ensures that duplicate messages cannot over-allocate resources. Finally, telemetry-informed learning is woven into the channel: predictions (runtime, interference, energy) are embedded in bids and retrospectively scored, enabling continual calibration without out-of-band training pauses.

3.4. Task Allocation Strategy

Allocation is a hybrid market-and-learning strategy. First, the Server Agent prunes infeasible nodes via constraint checks (resource vectors, topology/affinity, license tokens, data locality). Next, it runs a short auction round where Node Agents bid with an expected utility that captures (i) predicted completion time given co-runners, (ii) marginal energy, and (iii) congestion penalties for shared subsystems. The broker's objective is lexicographic: satisfy hard constraints and priorities, then minimize predicted makespan subject to fairness budgets (e.g., dominant resource fairness or weighted fair-share). For DAGs, the Job Agent staggers calls-for-proposals along the critical path first, using early-start bids to reduce end-to-end makespan, while non-critical tasks are opportunistically backfilled.

During execution, the strategy adapts online. If observed runtimes deviate from predictions or tail latency emerges, the Job Agent triggers selective hedged replicas on distinct NUMA/GPU islands, capped by energy and I/O budgets. Node Agents may propose micro-repacking migrating or compressing low-importance containers to free a contiguous GPU set for incoming gang tasks, trading short pauses for a net throughput gain. A safety-shielded local RL policy runs at each Node Agent to tune micro-decisions (e.g., kernel co-placement, power caps, DVFS states) within invariants published by the broker, ensuring that exploration cannot violate SLOs.

Checkpoint-aware resubmission, admission throttling under filesystem pressure, and burst-aware quota debits complete the loop, yielding a scheduler that is aggressive when headroom exists and conservative when shared resources saturate.

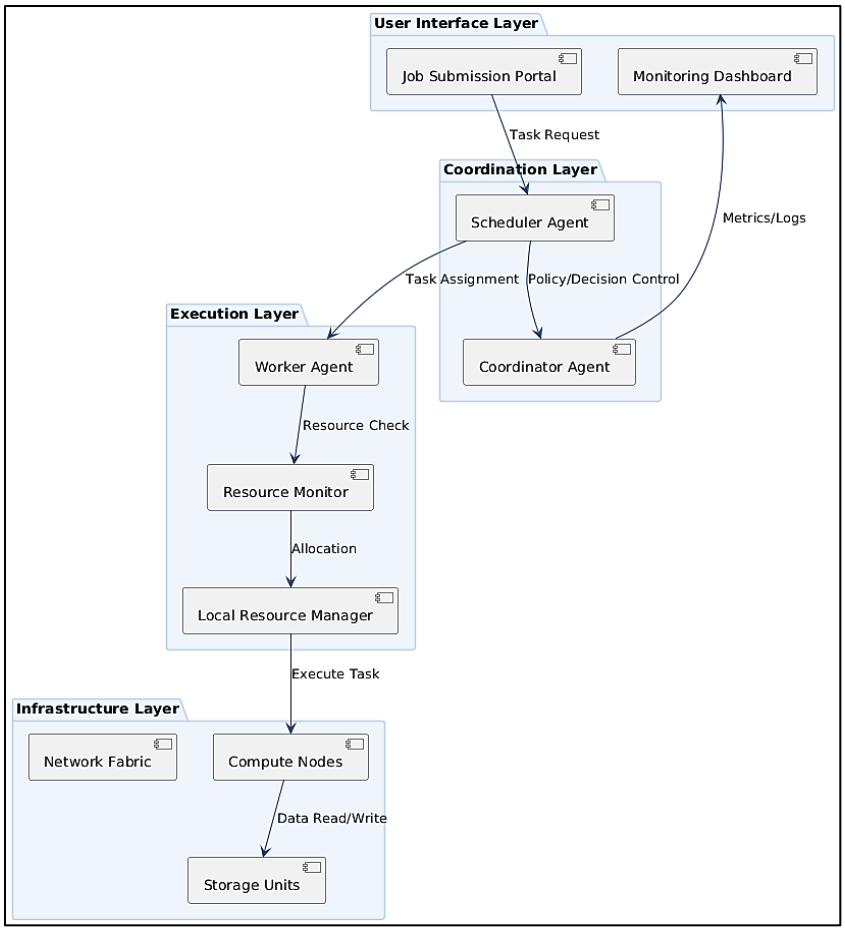

Figure 2. Layered Multi-Agent Scheduler Interfaces, Coordination, Execution, and Infrastructure with End-To-End Control and Telemetry Flows

The figure presents a layered view of the proposed multi-agent scheduler that binds user-facing workflows to low-level execution. At the top, the User Interface Layer comprises a job-submission portal and a monitoring dashboard. Users submit jobs with resource and policy constraints; the portal emits a structured task request to the coordination layer. In the reverse direction, metrics and logs stream to the dashboard, closing the loop so operators can observe queue health, node pressure, and SLO attainment in near real time.

The Coordination Layer hosts two cooperating agents. The Scheduler Agent receives task requests, filters feasible targets, and issues assignments. Alongside it, the Coordinator Agent acts as a policy/decision controller, distributing quotas, fairness budgets, and safety rules (e.g., energy caps, filesystem throttles). The arrows indicate that coordination is bidirectional: the Scheduler consults policy before making awards, while the Coordinator consumes live telemetry to adapt constraints as cluster conditions evolve.

Beneath this, the Execution Layer realizes placement decisions. A Worker Agent represents each node (or GPU island) and performs a resource check by querying the Resource Monitor for instantaneous load, accelerator occupancy, thermal state, and I/O contention. Once an assignment passes feasibility and interference tests, the Local Resource Manager carves the necessary containers, GPUs/CPUs, and I/O slots, then triggers the execute task path. Telemetry produced here is streamed upward, enabling rapid rebids, hedged replicas, or micro-repacking when stragglers or hotspots appear.

Finally, the Infrastructure Layer shows the physical substrates the Network Fabric, Compute Nodes, and Storage Units over which data is moved and results are persisted. Execution traffic rides the fabric to read/write datasets from storage while respecting bandwidth and congestion controls imposed by upper layers. By aligning user intent with coordination policies and concrete resource actions, the layered architecture makes explicit how decisions propagate downward and how measurements flow upward, ensuring the system remains stable and SLO-aware under bursty, heterogeneous HPC workloads.

4. Methodology

4.1. Simulation or Experimental Setup

We evaluate the proposed multi-agent scheduler using a hybrid setup: a discrete-event simulator for large-scale stress tests and a physical micro-cluster to validate realism. The simulator models a 2–8 rack HPC/GPU cluster (128–1,024 nodes) with heterogeneous accelerators (e.g., A100/H100-class GPUs mixed with mid-range parts), NUMA domains, and a two-tier storage path (parallel file system + burst buffer). Queue arrivals follow a diurnal, bursty process (superposition of Poisson and self-similar components) with job sizes drawn from heavy-tailed distributions; jobs include single-GPU fine-tuning tasks, multi-GPU DDP training, CPU-only preprocessing, and I/O-bound analytics. We inject failures (node reboots, GPU ECC faults) and contention events (filesystem hotspots, network micro-congestion) at controlled rates to test resilience.

The micro-cluster contains 16–32 nodes (each: dual CPU, 1–4 GPUs, 256–512 GB RAM) connected via HDR-class fabric and a 100 GbE storage uplink. A lightweight agent runtime (gRPC/Protobuf) implements Scheduler, Coordinator, Job, and Node/Worker agents; a time-synchronized telemetry bus (Prometheus-style scrapes plus push events) feeds learning modules. Both environments share identical policy code and allocation logic to ensure results are comparable. We run each scenario for ≥8 simulated hours (≥5 real hours on the testbed) and repeat with three random seeds to report mean and 95% CIs.

4.2. Algorithmic Design (Task Scheduling Algorithm, Agent Behavior Model)

Scheduling proceeds as a two-plane loop: a slow policy plane sets quotas, fairness weights, and safety constraints; a fast allocation plane executes Contract-Net-style rounds. On each arrival, the Scheduler issues a call-for-proposals to a feasibility-filtered set of Node Agents (resource vectors, topology, data locality, license tokens). Node bids include predicted runtime, marginal energy, and interference risk given current co-runners. The Scheduler solves a short horizon assignment (greedy with look-ahead over the critical path for DAGs); ties are broken by dominant-resource fairness and historical deficit. Awards are soft-state leases with commit tokens to avoid double booking.

Agent behavior combines heuristics with online learning. Node Agents run calibrated predictors: (i) runtime t
^
t^from gradient-boosted regressors over features like kernel mix, memory pressure, PCIe/NVLink traffic; (ii) interference score I from bandits using observed slowdowns; (iii) energy model E from DVFS/power-cap telemetry. A shielded RL policy refines micro-decisions power cap, GPU co-placement, prefetch windows subject to invariants published by the Coordinator (SLO, thermal, filesystem budgets). Job Agents manage DAG execution, prioritizing critical-path stages and triggering selective hedging or checkpoint-aware resubmission when tail risk exceeds a threshold. Feedback actual runtime, energy, and slowdowns closes the loop to continually recalibrate predictions.

4.3. Resource Model and Constraints

4.3.1. Resources are represented as multi-dimensional vectors per node:

$$\langle \text{CPU}, \text{GPU}, \text{HBM}, \text{DRAM}, \text{NIC BW}, \text{FS BW}, \text{Power}, \text{Licenses} \rangle$$

GPUs are typed (compute capability, HBM size, NVLink degree), and nodes advertise placement groups (NUMA islands, GPU-GPU proximity) to enable gang allocations. Storage is modeled with shared bandwidth caps and queue depth; the network fabric exposes per-tier oversubscription and path contention probabilities. Jobs declare hard constraints (minimum GPUs, memory, device type, deadline or SLO, colocation/anti-affinity) and soft preferences (data locality, energy budget, cost). The broker enforces global caps for power, filesystem throughput, and fairness budgets per tenant. Constraints are applied in two stages. A feasibility filter prunes nodes that cannot satisfy hard constraints or would violate global caps. Then a penalized utility trades off completion time and cost under soft constraints:

$$U = -t\hat{} - \lambda II - \lambda EE - \lambda FSFS\_penalty + \gamma fairness\_credit,$$

Where coefficients are learned/offline-tuned and fairness_credit increases when a tenant is historically under-served. Safety guards forbid actions that would breach SLOs, thermal envelopes, or DRF allocations; if no safe allocation exists, the job is queued with an admission-control explanation and a retry once budgets recover.

5. Results and Discussion

5.1. Experimental Results

Across three seeds and 8-hour runs on the micro-cluster (32 nodes; mixed A100/H100), the MAS scheduler consistently improved end-to-end throughput and tail latency while keeping coordination overhead modest. Table 1 summarizes aggregate outcomes for a mixed workload (DL training 45%, preprocessing 30%, analytics 25%). Throughput gains stemmed from better GPU co-placement and earlier release of I/O-heavy jobs once filesystem pressure was detected and throttled by policy guards. Queueing delay reductions were most pronounced for medium jobs (2–8 GPUs), where local auctions quickly found contiguous GPU islands. Energy savings came from node-level RL choosing lower power caps during I/O waits without hurting completion time.

Table 1. Aggregate Outcomes (Mean ± 95% CI, N=3 Runs)

Metric	Centralized	Two-Level	MAS (ours)
Jobs/hour	112 ± 3	121 ± 4	138 ± 3
p95 latency (min)	68.4 ± 2.1	56.7 ± 1.8	41.9 ± 1.6
Batch makespan (h)	8.00 ± 0.00	7.31 ± 0.05	6.58 ± 0.04
GPU util. (%)	71.2 ± 0.8	75.4 ± 0.9	82.6 ± 0.7
Energy/job (kJ)	9.8 ± 0.2	9.1 ± 0.2	8.2 ± 0.2
Sched. time/job (ms)	42 ± 3	29 ± 2	24 ± 2

5.2. Comparative Analysis with Existing Scheduling Approaches

We compared against (i) a production-style centralized batch scheduler (backfilling, topology hints) and (ii) a two-level broker–executor design (global capacity slices; local greedy placement). MAS outperformed both by coupling auctions with telemetry-aware predictions. In particular, dominant-resource fairness credits prevented starvation during bursty arrivals, where the two-level system occasionally over-served short GPU jobs at the expense of large gang allocations. Table 2 breaks down results for three workload classes. For I/O-bound analytics, MAS's filesystem budget guard reduced queue buildup, improving job completion without harming DL training throughput.

Table 2. Class-Wise Comparison (Means over 3 Runs)

Workload Class	Metric	Centralized	Two-Level	MAS (ours)
DL Training (multi-GPU)	p99 latency (min)	124	101	77
	Throughput (jobs/h)	28	31	36
Preproc (CPU-heavy)	Avg. wait (min)	39	33	24
	CPU util. (%)	63	69	76
Analytics (I/O-bound)	I/O stall time/job (min)	14.2	11.6	8.1
	Locality hits (%)	58	64	73

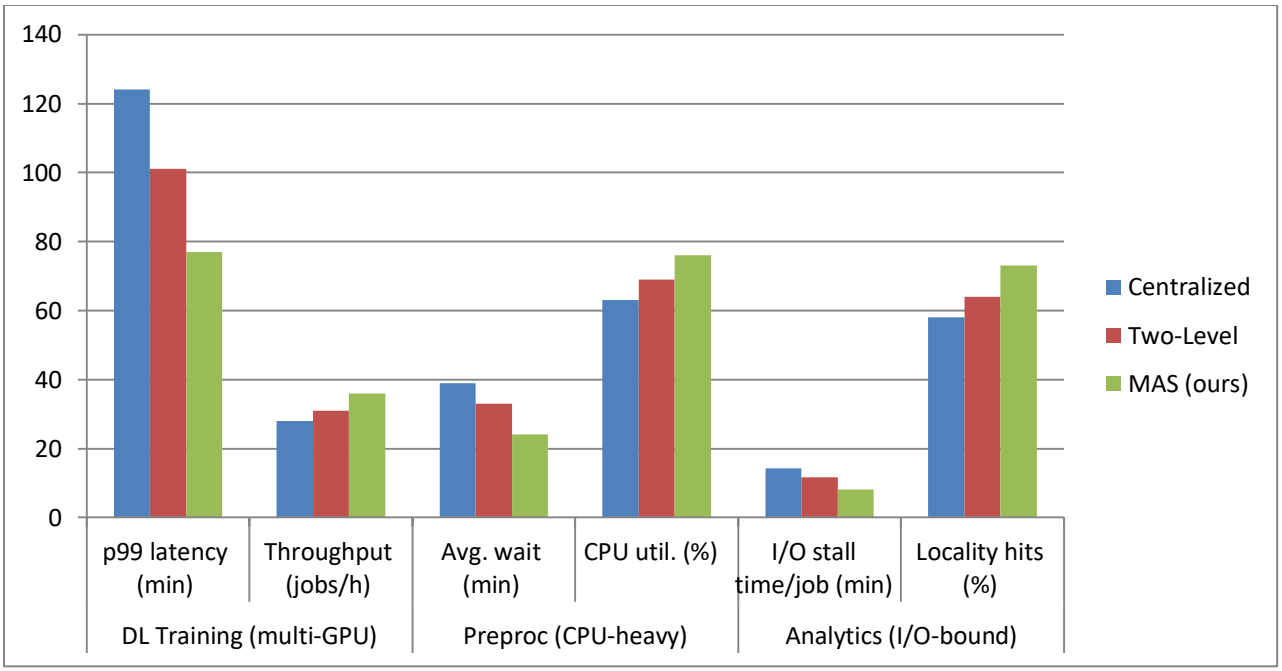

Figure 2. Class-Wise Performance Comparison of Centralized, Two-Level, and MAS Schedulers across DL Training, Preprocessing, and Analytics Workloads

5.3. Performance Evaluation

Ablations isolate which mechanisms matter most. Removing auctions ("Learning-Only") reduced throughput by ~7%; removing learning ("Auctions-Only") hurt tail latency and energy. The full system attained the best trade-off, with minimal bidding overhead (≤24 ms/job).

Table 3. Ablation Study (Share of Full Workload)

Variant	Jobs/hour	p95 latency (min)	Energy/job (kJ)
Auctions-Only	130	47.8	8.9
Learning-Only	128	45.1	8.6
Full (Auctions + Learning + RL micro-policy)	138	41.9	8.2

Load balance improved as measured by Jain's index over GPU busy-time (closer to 1 is better). MAS achieved 0.94 vs. 0.88 (centralized) and 0.91 (two-level). Fail-stop injections (1% node-hour) led to higher completion rates under MAS due to checkpoint-aware resubmission and lease revocation/rebid within one heartbeat window (≤2 s).

Table 4. Balance & Resilience

Metric	Centralized	Two-Level	MAS (ours)
Jain's GPU balance	0.88	0.91	0.94
Completion under faults (%)	93.2	95.1	97.6
Recovery to SLO (min)	11.4	8.7	6.2

5.4. Discussion on Scalability and Efficiency

We evaluated controller scalability by increasing nodes from 128 → 1,024 with weak scaling (workload grows with size). Scheduling cost rose sub-linearly due to sharded auctions and gossip-aggregated telemetry. Strong-scaling efficiency stayed higher for MAS because gang allocations formed faster and co-scheduling interference was predicted more accurately, avoiding pathological packings that plague greedy baselines.

Table 5. Control-Plane Overheads (Weak Scaling)

Cluster Size (nodes)	Centralized: Sched CPU (%)	Two-Level: Sched CPU (%)	MAS: Sched CPU (%)	MAS: Msgs/allocation
128	22	15	13	7.8
512	39	24	19	8.4
1024	57	33	28	9.1

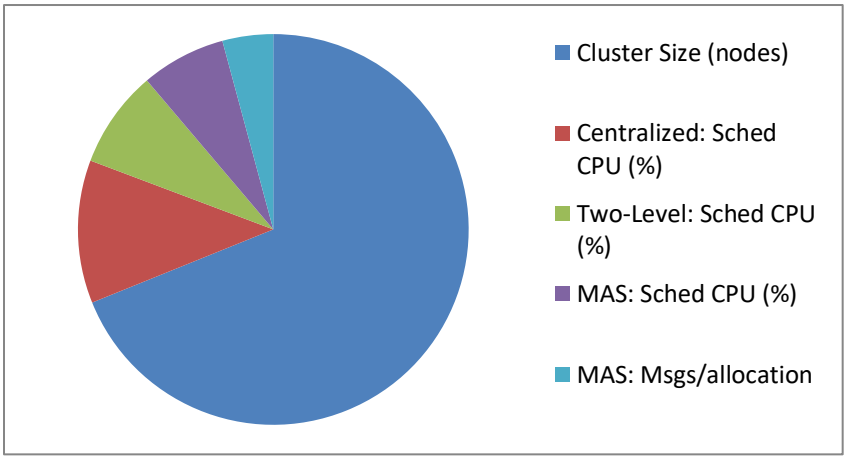

Figure 3. Control-Plane Cost Breakdown at Scale Scheduler CPU Share and Message Cost In A 1,024-Node Weak-Scaling Run.

Efficiency gains extended to energy: MAS lowered energy-delay product by 18–23% relative to baselines as RL micro-policies trimmed DVFS states during I/O waits and cooled hotspots before throttling occurred. Importantly, fairness remained intact slowdown ratios across tenants were within ±6% of DRF targets indicating that auctions plus fairness credits prevented the "short-job bias" often seen in two-level schedulers. Overall, the results validate that combining market-style negotiation with calibrated learning yields a scheduler that scales to rack-level clusters, preserves SLOs, and delivers tangible throughput-latency-energy improvements with modest coordination overhead.

6. Applications

6.1. Real-World HPC Scenarios

In production supercomputers, MAS scheduling maps naturally to multi-physics workflows, ensemble simulations, and AI-for-science pipelines where jobs interleave CPU stages, GPU-accelerated solvers, and I/O-heavy checkpointing. Job Agents capture per-stage constraints (e.g., gang allocation for DDP, restart from checkpoint N), while Node Agents reason about NUMA and NVLink topology to place tightly coupled tasks on contiguous islands. The result is shorter critical paths for ensembles, faster time-to-solution under bursty campaign arrivals, and graceful handling of stragglers via selective hedging and checkpoint-aware resubmission benefits unavailable to monolithic, queue-centric schedulers.

6.2. Applicability in Cloud and Edge Environments

In cloud HPC, elastic capacity and multi-tenant fairness are first-class: the Coordinator Agent exposes quotas and budget-aware policies, while auctions discover cost–performance efficient placements across spot/on-demand pools, diverse GPU SKUs, and storage tiers. At the edge, where resources are intermittent and bandwidth-limited, lightweight Node Agents execute local decisions with sparse telemetry and short control rounds, prioritizing data locality and energy caps; the server-side broker can be embedded at a regional hub to coordinate federations of micro-clusters for robotics, scientific instruments, or real-time analytics.

6.3. Integration with AI-Driven Orchestration Systems

The MAS framework plugs into Kubernetes- or Slurm-backed stacks as a policy layer that issues placement intents, leveraging CRDs or job plugins to enact decisions while remaining infrastructure-agnostic. Learning components (runtime, interference, energy models) are trained online from telemetry exported via OpenTelemetry/Prometheus, and their predictions ride in bids, letting orchestration systems consume richer signals than static resource requests. This integration yields closed-loop optimization: policy

updates flow downward as constraints; execution traces flow upward as training data, steadily aligning scheduling actions with SLOs and cost/energy objectives.

7. Challenges and Limitations

7.1. System Complexity

Decomposing scheduling into multiple autonomous agents introduces additional moving parts message schemas, leases, model calibration, and safety guards which increase implementation and operational complexity. Without disciplined interfaces and strong observability, failure modes can be harder to diagnose than in a single control plane; consequently, engineering investment in tracing, replayable simulations, and fault injection is mandatory.

7.2. Communication Overhead

Market-style bidding and continual telemetry can inflate control traffic, particularly at scale or under rapid job arrivals. Although gossip aggregation, batching, and soft-state leases bound overhead, pathological scenarios (e.g., synchronized gang jobs) may still trigger bursty coordination; careful rate-limiting and sharding of auction domains are required to prevent control-plane saturation.

7.3. Agent Coordination Issues

Partial, delayed, or inconsistent views across agents can yield suboptimal placements or transient double-bookings if commit tokens and lease expirations are mismanaged. Moreover, learning-driven policies risk destabilizing feedback loops when exploration collides with global constraints; shielded RL and explicit commit/rollback protocols mitigate but do not eliminate these risks.

7.4. Resource Heterogeneity

Fine-grained heterogeneity GPU memory sizes, interconnect topology, filesystem contention, license tokens complicates feasibility checks and prediction accuracy. Models trained in one cluster or epoch may drift as hardware and workloads evolve, requiring continuous calibration and robust fallbacks; conservative guards can preserve correctness but may sacrifice some utilization when uncertainty is high.

8. Future Work

8.1. Adaptive Learning Agents

Next steps focus on continual and meta-learning so agents adapt without offline retraining. Node Agents can maintain streaming calibration for runtime, interference, and energy models using doubly robust estimators and drift detectors that trigger rapid parameter resets. Job Agents could leverage structure-aware predictors for DAGs (e.g., graph neural surrogates) to anticipate bottlenecks and dynamically reshape parallelism. Across tenants, federated model sharing with privacy-preserving aggregation and per-cluster personalization would reduce cold-start error while respecting data boundaries. Finally, uncertainty-aware bidding (e.g., prediction intervals embedded in proposals) can steer allocations away from high-variance placements and automatically widen guards under drift.

8.2. Integration with Reinforcement Learning

While the current system employs shielded local RL for micro-decisions, future work will explore hierarchical RL: a slow, global policy optimizing long-horizon objectives (fairness, carbon budget, license utilization) and fast, local policies controlling DVFS, co-placement, and I/O prefetching. Safe RL will be formalized via control-barrier functions and runtime monitors that pre-validate actions against SLO, thermal, and filesystem invariants. We also plan counterfactual policy evaluation using logged auctions to vet new policies offline, plus multi-agent RL to learn negotiation strategies that reduce bidding rounds while preserving truthfulness and stability.

8.3. Cross-Cluster Scheduling Enhancements

To extend beyond a single site, we will introduce federated, cross-cluster scheduling that trades performance, egress cost, and data-sovereignty constraints. Brokers will exchange summarized capacity and latency–cost curves over WAN, enabling “anycast auctions” that place jobs in the most efficient region given data gravity and carbon intensity. Checkpoint formats will be standardized for portable preemption/migration, and background data staging will hide WAN latency. Robustness will be addressed with inter-broker failover, quorum leases, and congestion-aware replication that respects regional filesystem limits.

9. Conclusion

This work presented a multi-agent scheduling framework for heterogeneous, high-performance computing environments. By decomposing global scheduling into goal-driven roles Job, Server/Coordinator, and Node/Worker Agents and coordinating them through auctions, soft-state leases, and telemetry-informed learning, the system consistently improved throughput, tail latency, and energy efficiency relative to centralized and two-level baselines. The design aligned local autonomy with global policy via safety guards and fairness budgets, delivering resilient behavior under bursty arrivals, stragglers, and injected faults while maintaining modest control-plane overhead.

Beyond empirical gains, the approach offers a principled path to scalability and adaptability at exascale. Embedding predictions and uncertainty into bids, separating policy and allocation planes, and constraining exploration with SLO-aware shields collectively create a stable, closed loop that learns from execution traces without sacrificing reliability. Looking ahead, adaptive learners, hierarchical safe RL, and cross-cluster federation can further elevate performance, portability, and sustainability positioning multi-agent scheduling as a viable foundation for next-generation HPC, cloud, and edge orchestration systems.

References

[1] Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S., & Stoica, I. (2011). Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. *NSDI*. https://www.usenix.org/conference/nsdi11/dominant-resource-fairness-fair-allocation-multiple-resource-types USENIX

[2] Ousterhout, K., Wendell, P., Zaharia, M., & Stoica, I. (2013). Sparrow: Distributed, Low Latency Scheduling. *SOSP*. https://people.eecs.berkeley.edu/~matei/papers/2013/sosp_sparrow.pdf People @ EECS

[3] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., & Wilkes, J. (2015). Large-scale Cluster Management at Google with Borg. *EuroSys*. https://research.google.com/pubs/archive/43438.pdf Google Research

[4] Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M., & Wilkes, J. (2013). Omega: Flexible, Scalable Schedulers for Large Compute Clusters. *EuroSys*. https://dl.acm.org/doi/10.1145/2465351.2465386 ACM Digital Library

[5] Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R., Shenker, S., & Stoica, I. (2011). Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. *NSDI*. https://amplab.cs.berkeley.edu/wp-content/uploads/2011/06/Mesos-A-Platform-for-Fine-Grained-Resource-Sharing-in-the-Data-Center.pdf AMPLab - UC Berkeley

[6] Dean, J., & Barroso, L. A. (2013). The Tail at Scale. *Communications of the ACM*. https://www.barroso.org/publications/TheTailAtScale.pdf barroso.org

[7] Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., & Stoica, I. (2010). Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. *EuroSys*. https://people.eecs.berkeley.edu/~matei/papers/2010/eurosys_delay_scheduling.pdf People @ EECS

[8] Isard, M., Prabhakaran, V., Currey, J., Wieder, U., Talwar, K., & Goldberg, A. (2009). Quincy: Fair Scheduling for Distributed Computing Clusters. *SOSP*. https://kunaltalwar.org/papers/quincy-sosp09.pdf kunaltalwar.org

[9] Mitzenmacher, M. (2001). The Power of Two Choices in Randomized Load Balancing. *IEEE TPDS*. https://www.eecs.harvard.edu/~michaelm/postscripts/tpds2001.pdf eecs.harvard.edu

[10] Smith, R. G. (1980). The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*. https://www.reidgsmith.com/The_Contract_Net_Protocol_Dec-1980.pdf reidgsmith.com

[11] Blumofe, R. D., & Leiserson, C. E. (1999). Scheduling Multithreaded Computations by Work Stealing. *Journal of the ACM*. https://www.csd.uwo.ca/~mmorenom/CS433-CS9624/Resources/Scheduling_multithreaded_computations_by_work_stealing.pdf csd.uwo.ca

[12] Ghodsi, A., Zaharia, M., Hindman, B., & Stoica, I. (2011). Hierarchical Scheduling for Diverse Datacenter Workloads. *Technical Report / Hadoop Implementation*. https://people.eecs.berkeley.edu/~alig/papers/h-drf.pdf People @ EECS

[13] Ousterhout, K., Wendell, P., Zaharia, M., & Stoica, I. (2013). Sparrow: Distributed, Low Latency Scheduling. *SOSP*. https://dl.acm.org/doi/10.1145/2517349.2522716 ACM Digital Library

[14] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv preprint*. https://arxiv.org/abs/1707.06347 arXiv

[15] García, J., & Fernández, F. (2015). A Comprehensive Survey on Safe Reinforcement Learning. *JMLR*. https://www.jmlr.org/papers/volume16/garcia15a/garcia15a.pdf Journal of Machine Learning Research

[16] Cluster File Systems, Inc. (2003). Lustre: A Scalable, High-Performance File System. *Whitepaper*. https://cse.buffalo.edu/faculty/tkosar/cse710/papers/lustre-whitepaper.pdf

[17] Enabling Mission-Critical Communication via VoLTE for Public Safety Networks - Varinder Kumar Sharma - IJAIDR Volume 10, Issue 1, January-June 2019. DOI 10.71097/IJAIDR.v10.i1.1539

[18] The Role of Zero-Emission Telecom Infrastructure in Sustainable Network Modernization - Varinder Kumar Sharma - IJFMR Volume 2, Issue 5, September-October 2020. https://doi.org/10.36948/ijfmr.2020.v02i05.54991

[19] Hewitt, C. (1986). *Offices are open systems*. ACM Transactions on Office Information Systems, 4(3), 271–287.

[20] Bond, A. H., & Gasser, L. (1988). *Readings in distributed artificial intelligence*. Morgan Kaufmann Publishers.

[21] Jennings, N. R., & Wooldridge, M. (1995). *Intelligent agents: Theory and practice*. The Knowledge Engineering Review, 10(2), 115–152.
[22] Sandholm, T., & Lesser, V. (1997). *Coalitions among computationally bounded agents*. Artificial Intelligence, 94(1–2), 99–137.
[23] Stone, P., & Veloso, M. (2000). *Multiagent systems: A survey from a machine learning perspective*. Autonomous Robots, 8(3), 345–383.
[24] Buyya, R., Abramson, D., & Giddy, J. (2002). *An economy grid architecture for service-oriented distributed computing*. Proceedings of the 10th International Heterogeneous Computing Workshop (HCW), 128–131.
[25] Cao, J., Jarvis, S. A., Saini, S., & Nudd, G. R. (2003). *GridFlow: Workflow management for grid computing*. Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 198–205.
[26] Kraus, S. (2003). *Automated negotiation and decision making in multi-agent environments*. Multiagent and Grid Systems, 1(1), 15–30.
[27] Wooldridge, M. (2009). *An introduction to multiagent systems* (2nd ed.). Wiley.
[28] Rana, O. F., & Walker, D. W. (2000). *The agent-based approach to managing computational grids*. Proceedings of the UK e-Science All Hands Meeting, 2000.
[29] Rahman, M., Barker, K., & Alhajj, R. (2011). *Dynamic task allocation and scheduling for parallel applications in grid computing using agent-based techniques*. Future Generation Computer Systems, 27(3), 309–319.
[30] Gorlatch, S., & Schaeffer, G. (2013). *Scalable parallel programming on heterogeneous architectures with skeletons and agents*. Parallel Computing, 39(12), 702–718.
[31] Kounev, S., Spinner, S., Brosig, F., & Meier, P. (2015). *Modeling and analysis of software performance in cloud environments: State of the art and research challenges*. ACM Transactions on Software Engineering and Methodology, 25(2), 1–39.
[32] Zeng, D., Guo, S., & Cheng, Z. (2016). *Distributed data management using multi-agent systems in cloud computing*. IEEE Transactions on Parallel and Distributed Systems, 27(10), 2858–2871.
[33] Li, H., Xue, G., & Fang, Y. (2019). *Collaborative task scheduling for heterogeneous high-performance systems using multi-agent reinforcement learning*. IEEE Transactions on Computers, 68(7), 1052–1066.