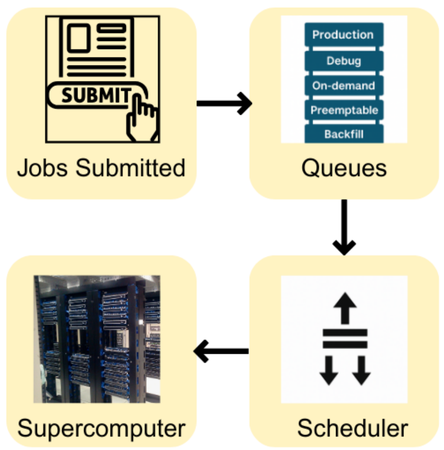
A Survey of Scheduling Policies in High-Performance Computing Systems

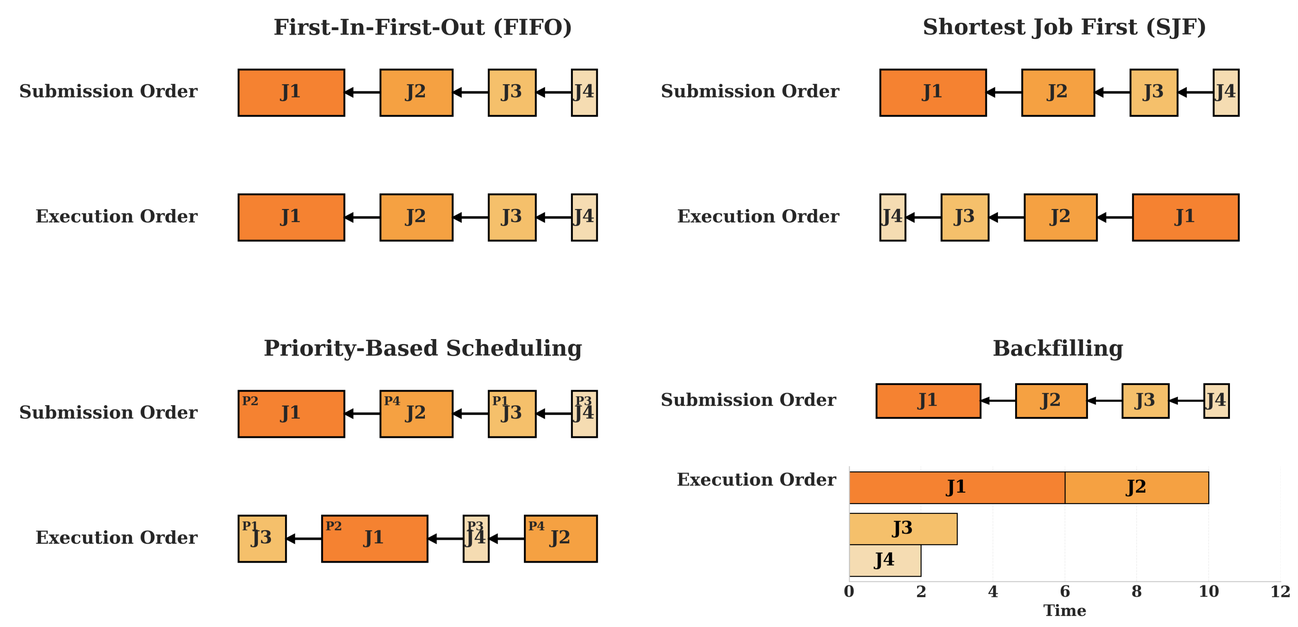
Kausalya Sankaranarayanan, Northeastern University

HPC Scheduling Overview



High-Performance Computing (HPC) Systems: Runs diverse, large-scale scientific workloads.
Efficient Job Scheduling: Critical to maximize utilization, reduce wait times, and ensure fairness. Scheduling policies directly affect system throughput and user satisfaction.
Traditional: First-In-First-Out (FIFO), Shortest Job First (SJF), Priority-based, Backfilling.
In-production: SLURM, Cobalt, PBS Pro.
Modern/Future: ML-driven, Energy-aware, Heterogeneity-optimized.

Traditional Scheduling Policies



FIFO (First-In-First-Out): Runs jobs strictly in arrival order.
SJF (Shortest Job First): Chooses the runnable job with the smallest estimated runtime to minimize average wait/turnaround.
Priority-based: Orders jobs by admin/user-assigned priority (queues/QoS/weights), often with fair-share/aging and optional preemption.
Backfilling: Reserves a start for the head-of-queue job, then fills idle gaps with smaller jobs that won't delay that reservation.

Policy	Advantages	Disadvantages
FIFO	• Easiest to implement/understand • Predictable "first come, first served" • Low scheduler overhead	• Head-of-line blocking (big job stalls the queue) • Poor average wait for short jobs • Utilization can suffer under bursty arrivals
SJF	• Minimizes average wait/turnaround (if runtimes known) • Great for interactive/short jobs • Often boosts throughput	• Needs good runtime estimates; bad estimates hurt fairness • Starvation risk for long jobs • Users may game by under-estimating wall time
Priority-based	• Aligns with organization goals (prod/debug/urgent) • Clear control knobs (queues/weights/limits) • Works with preemption/quotas	• Starvation of low-priority jobs • Priority inversion / fairness concerns • Requires ongoing tuning & governance
Backfilling	• Fills idle gaps → higher utilization • Improves short-job latency without delaying reserved head job • Better overall throughput	• Depends on accurate wall time estimates • More complex scheduler logic/overhead • Incentive to under-estimate; may reduce guarantees • Start-time predictability is harder to communicate

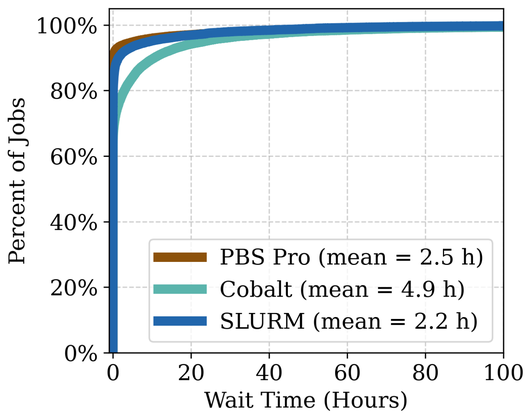
Each policy has distinct trade-offs—what improves utilization or responsiveness may hurt fairness or predictability—so there is no single "best" choice for all workloads. In practice, modern in-production schedulers (e.g., SLURM, PBS Pro, Cobalt) combine these mechanisms—priorities/fair-share with FIFO ordering, backfilling, and optional preemption—to balance cluster goals and user needs.

Current In-production Schedulers

Modern supercomputers typically use **SLURM, PBS Pro/OpenPBS, or Cobalt** schedulers. These aren't single policies but **policy frameworks**: admins tune **priorities/fair-share/aging**, enable **backfilling**, set **QoS/partitions**, and optionally use **preemption** and **reservations**. In practice, they combine FIFO ordering within priority classes with backfilling to raise utilization and cut short-job latency, while quotas/allocations and fair-share keep long-term fairness.

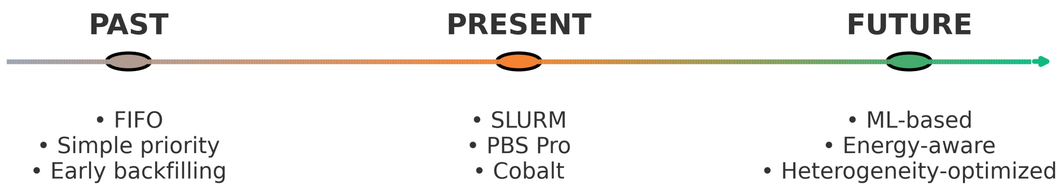
Scheduler	Fair-share	Reservation	Preemption	Backfill	Priority calc
SLURM	✓	✓	✓	✓	cycle
PBS Pro	✓	✓	✓	✓	cycle
Cobalt	✓	✓	◐	✓	accumulating

Legend: ✓ supported ◐ partial/site-config — not typical
Priority calc: cycle = recomputed each pass; accumulating = running score increases with wait



Despite widespread use of SLURM, PBS Pro, and Cobalt, large HPC systems still experience substantial queue wait times under real workloads. Minimizing queue delay remains an active research priority.

Future Work



ML-based/adaptive scheduling. Learn priorities from real traces and adapt online while honoring guardrails (fair-share, quotas, reservations). Start in "shadow mode," optimize multiple signals (median/tail wait, bounded slowdown, utilization), and require basic explainability for operator trust.
Energy/carbon-aware policies. Incorporate power caps and grid carbon-intensity forecasts; shift flexible jobs to greener windows, throttle within QoS limits, and report per-job CO_2e. Balance carbon savings against fairness and deadlines.
Heterogeneity-optimized placement. Place jobs onto the right mix of CPU/GPU/accelerators with memory, NUMA, and network topology awareness; support gang/elastic jobs and moldable requests; reduce fragmentation across partitions.
Better runtime estimates & incentives. Train per-app/user models; calibrate and auto-update after each run. Use soft/hard wall time with gentle penalties or auto-extensions to discourage under-estimation without punishing honest users.
Predictable starts & QoS. Provide ETA predictions with confidence bounds, leverage advance reservations, preemption, and elastic scaling to hit SLOs for priority workloads while maintaining long-term fairness.

Conclusion

Modern in-production schedulers (SLURM, PBS Pro, Cobalt) are policy frameworks, blending priority/fair-share with FIFO ordering, backfilling, and optional preemption/reservations. They work well across diverse workloads, yet long waits and heavy tails still appear at scale—often driven by estimation errors, heterogeneous resource constraints, and bursty demand.
The path forward is pragmatic and incremental: improve runtime estimates, make ETAs transparent, keep conservative backfilling with fair-share to protect equity, and pilot ML-assisted decisions in shadow mode before enforcement. As systems grow more heterogeneous and energy becomes a first-class constraint, schedulers should co-optimize performance, fairness, and carbon—treating the scheduler not as a single policy, but as a tunable, data-driven system that evolves with the workload.