Managing batch jobs during peak periods like month-end processing can be a real challenge in cloud environments—especially when running on shared CPU instances. These jobs often require sudden bursts of compute capacity, and if you don't plan carefully, you could be paying for idle resources most of the month or suffering severe performance degradation during critical processing windows.

This post digs into how to approach **month-end activity** and **scheduled jobs** on *shared CPU* clouds with a practical engineering mindset. I'll draw from tools like AWS Compute Optimizer and Azure Advisor to help guide capacity planning without falling into common pitfalls.

Why Month-End Batch Jobs Are Different

When you're running infrastructure that supports all sorts of daily traffic, your operational chatter is steady—maybe even predictable. But month-end jobs slam the throttle wide open in bursts that can be multiples of your baseline usage.

Common issues I've seen:

- **Always-on small services** masking real cost: Someone forgets to turn off idle test environments or small APIs that run 24/7, quietly hoarding resources while your big jobs crawl.
- **Confusing shared CPU definitions**: Different cloud providers define "shared CPU" very differently, which impacts both performance and pricing.
- **Capacity planning** based entirely on averages: Average CPU utilization can be misleading and cause widespread under-provisioning.

Understanding Shared CPU Models in AWS and Azure

Before you optimize, you have to know what you're optimizing on. The term "shared CPU" means different things depending on your cloud provider, and this impacts how you measure performance or estimate cost.

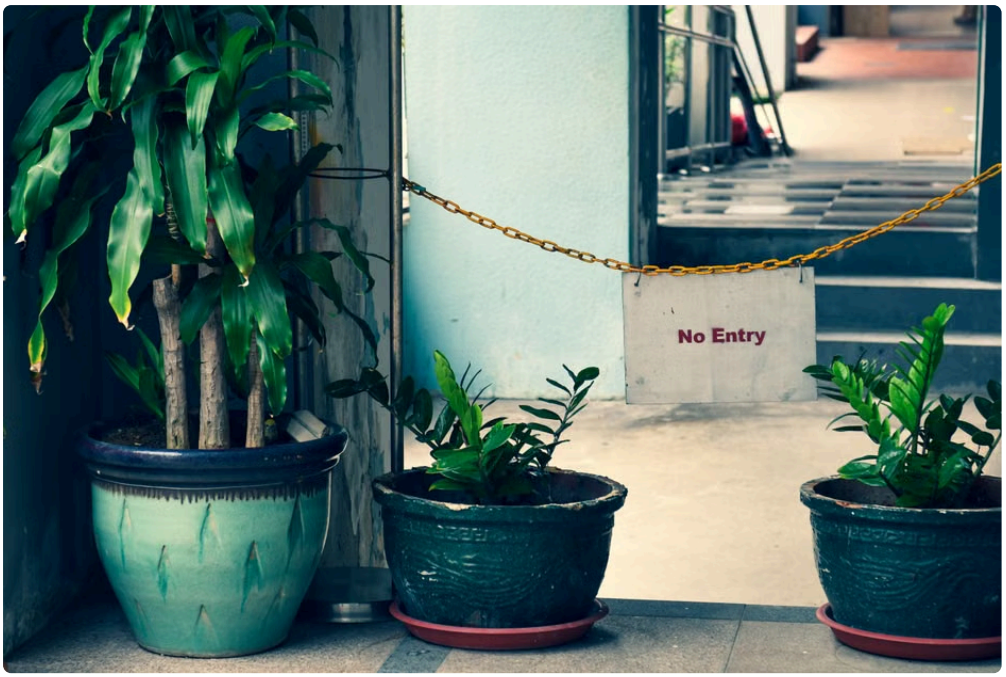
AWS Shared CPU Instances

AWS offers T-series burstable instance types (like t3, t4g) and others with a *CPU credit* model.

- Instances accumulate CPU credits when they're idle, which can be 'spent' in bursts.
- If you use up credits, performance throttles, with CPU capped and possibly impacting job completion times.
- Crucially, your workload needs to be intermittent or you risk throttling during critical batch jobs.

Azure Shared CPU VMs

Azure's B-series is the closest analog to AWS's burstable instances.



- Like AWS, Azure meals out *credits* when underutilized and charges bursting against those credits.
- Details on credit accrual and spending differ slightly in cadence and limits.
- Not all Azure VM families support CPU bursting.

What this means in practice: *Do not assume shared CPU means shared performance!* When you see a VM with 2 vCPUs in either provider's burstable family, this does NOT mean you have the equivalent of 2 always-on physical cores. This gets messy quickly when you're trying to deliver guaranteed job performance during month-end processing.

Measure Peaks Accurately: Use the Right Observation Window

One of my biggest annoyances is when teams plan based on a daily or hourly average CPU. This approach misses the spikes that matter, because the *batch job's performance depends on peak capacity* rather than average utilization.

Here's what I recommend practically:

1. **Capture high-resolution metrics:** Use CloudWatch (AWS), Azure Monitor, or your metrics system to record CPU utilization at 1-minute (or better) granularity.
2. **Pick your observation window wisely:** For month-end batch jobs, the right window is often the full duration of the job plus some buffer. For example, if jobs run for 4 hours, focus on that 4-5 hour window each month rather than whole-day or week averages.

3. **Calculate percentiles over that window:** Your goal is to understand P95 and P99 CPU utilization, i.e., the values below which 95% or 99% of the CPU measurements fall. These reveal spikes, not smoothed averages.
4. **Look at spike duration:** Are you seeing short bursts of 30 seconds or prolonged load over 30 minutes? This impacts whether temporary bursting suffices or if sustained capacity planning is needed.

Example: Using AWS Compute Optimizer to Assess Burstable Instances

AWS Compute Optimizer can analyze your EC2 instances over the past 14 days, highlighting opportunities to optimize based on utilization patterns.

Metric Observation Action Average CPU utilization Shows 20% over full day Misleading. Monthly averages hide spikes. P95 CPU utilization on job run days 85% Shows real month-end peak load. CPU credit balance Shows frequent depletion on batch periods Triggering throttling, job times could increase

With these data points, you might move from a burstable instance to a small baseline performance instance or schedule jobs to spread load if possible.

Don’t Let Always-On Small Services Hide Cloud Waste

Have you looked at your staging environments or small microservices in production? These always-on systems often run on fixed-size VMs (sometimes shared CPU) but contribute to higher baseline cloud spend.

Month-end jobs inflate CPU demand in bursts, while these always-on services create a "floor" of resource use that might be avoidable or shiftable.

- **Audit these always-on services**: Are they needed all day? Could they be off when not serving active users?
- **Streamline or consolidate**
- **Tag and track costs** carefully to avoid hidden waste.

Reducing this baseline frees budget to handle those peaks more gracefully without resorting to oversized over-provisioning.

Using Azure Advisor for Similar Insights

Azure Advisor integrates with Azure Monitor and provides recommendations based on observed metrics and configuration.

- **Right-size VMs:** Advisor will warn you if you have oversized or undersized VMs based on your observed peak usage.
- **Insights on burstable instances:** If you rely on B-series VMs, Advisor may show when you run out of credits and suggest changes.
- **Cost-saving recommendations:** Often includes scheduling shutdowns for dev/test or always-on services that could be moved.

As with AWS Compute Optimizer, feed in data covering your *month-end activity window* so recommendations reflect critical-peak usage, not averages.

Capacity Planning: Percentiles and Spike Duration, Not Averages

Engineers love averages. They're simple. But they frequently cause under-provisioning or waste. Your capacity plan needs more granularity.

- **Use P95 and P99 metrics:** These show the behavior during the busiest ~5% or 1% of time periods, giving you a safety margin.
- **Factor in the duration of CPU spikes:** A single 30-second spike might be tolerable, but sustained elevation for hours needs dedicated capacity.
- **Include storage IOPS and network egress calculations:** Don't just look at CPU averages—batch jobs can cause storage bottlenecks or network costs that average CPU misses.
- **Develop rollback criteria before your pilot:** For example, if CPU throttling increases job time > 10%, rollback your instance type changes or push batch jobs to a scheduled off-peak window.

Putting It All Together: A Sample Month-End Batch Job Strategy

1. **Collect detailed metrics** across at least two full month-end periods, including CPU, storage, network.
2. **Use AWS Compute Optimizer or Azure Advisor** to identify burstable instance limits or oversizing on always-on services.
3. **Calculate P95/P99 and spike-duration** CPU usage during batch jobs.
4. **Audit always-on services** and shut down or right-size them to reduce baseline.
5. **Select instances or VM types** that can sustain your P95 load without throttling or high latency.
6. **Consider scheduling jobs staggered across different fleet segments** if peak demands exceed practical instance sizes.
7. **Define rollback criteria** to detect when batch times degrade or throttling occurs during your pilot rollout.

Summary

Month-end batch jobs on shared CPU clouds are tricky. If you rely on averages or treat vCPU counts as hard performance guarantees, you risk wasting money https://computingforgeeks.com/shared-cpu-cloud-waste-migration-guide/ or failing SLAs. Instead, dig into high-resolution metrics across the right observation windows and focus on *peak percentiles and spike durations*.

Leverage tools like AWS Compute Optimizer and Azure Advisor to get data-driven recommendations based on your actual workloads. Audit your always-on small services—sometimes those hidden costs matter more than your batch jobs.

Finally, plan your experiments with clear rollback criteria and respect the differences between cloud providers' shared CPU models. Doing this lets you optimize cost, capacity, and performance in harmony.

Further Reading

- Understanding Burstable Instance Credits on Amazon EC2
- Azure B-series Burstable VMs overview
- AWS Compute Optimizer Documentation
- Azure Advisor Documentation