When architecting cloud infrastructure for always-on small services, one of the trickiest cost-performance trade-offs revolves around **shared CPU** instances. Many engineers gravitate to shared CPU models due to their lower price points and apparent flexibility to handle sporadic workloads. However, the devil is in the details: how long can your CPU bursts last—and how often should they recur—before shared CPU throttling or noisy neighbors start crippling performance?

In this deep dive, I'll walk you through what I've learned after running cost reviews and performance tuning across AWS, Azure, and Google Cloud. We'll discuss:

- Why always-on small services hide cloud waste
- The varying definitions of shared CPU across cloud providers
- The critical importance of measuring CPU peaks with the appropriate observation window
- How to use percentiles and spike duration, rather than averages, for better sizing
- Practical guidance with tools like AWS Compute Optimizer and Azure Advisor

Let's start with the basics: what counts as a CPU burst, and why does burst *duration* matter?

Understanding Burst Duration, Recurrence, and Sustained Demand

Cloud providers often offer "burstable" or shared CPU instance types to optimize for workloads with intermittent CPU needs. Instead of paying for full vCPU capacity 24/7, you get a low baseline with the ability to burst above that baseline for a limited time.

- **Burst duration:** How long a CPU can operate at high utilization above its baseline before it gets throttled.
- **Recurrence:** How often these bursts happen—once an hour, multiple times per minute, or sustained over long periods.
- **Sustained demand:** The CPU usage level the workload needs consistently over time, above which shared CPU instances become inefficient or even problematic.

The key takeaway: **burst duration and recurrence determine whether shared CPU works or becomes a bottleneck.** A system exhibiting frequent or long-lasting bursts will likely exhaust burst credits (if applicable) and face throttling, causing latency spikes and poor user experience.

Always-On Small Services: The Cloud Waste Hidden in Plain Sight

One thing I've seen repeatedly is that always-on small services tend to hide cloud waste. Why? Because they often run on under-resourced but always-on shared CPU instances to contain costs. Yet, this "optimization" can backfire.

To make matters worse, many operators measure average CPU utilization over hours or days. Averaging smooths out spikes and masks the true demand patterns. This leads to a false sense that CPU capacity is sufficient, even though the system experiences regular microbursts that lead to throttling.

Here's why this is a problem:

- **Hidden latency spikes:** Users experience poor responsiveness during bursts that exceed available CPU credits.
- **Unnecessary restart and scaling cycles:** Autoscalers or orchestration platforms react to lagging performance rather than actual sustained CPU demand.

- **Missed optimization opportunities:** Engineers keep adding tiny shared CPU instances—which appear cheap but collectively rack up bills with hidden inefficiencies.

One of my favorite "gotchas" is when teams tell me, "Our CPU average is only 10%, so using the smallest shared CPU instance is justified." I always ask: *what do your P95 and P99 CPU usage percentiles look like? And over what observation window do you measure those?* The answers almost never justify shared CPU.

How Shared CPU is Defined Differently by Cloud Providers

Shared CPU isn't a standardized concept across clouds. Here's how three big providers differ, and why knowing these differences matters when sizing for burst duration and recurrence.

Cloud Provider Shared CPU Definition CPU Credit Model Throttling Behavior Observation Window* AWS (T-series) Baseline CPU with credit accrual and bursting above baseline CPU Credits measured in credit seconds Throttles CPU once credits are exhausted 5 minutes rolling window Azure (B-series) Baseline with accrued credits and burst capability CPU Credits accrue during idle, used on burst CPU throttled on credit depletion Similar 5-minute scale Google Cloud (E2 shared-core) Processes share a physical CPU; no formal credit system No credit system; best-effort CPU time sharing Throttling based on host contention and workload priority Variable, depends on host load

*These observation windows are rough guidance and depend on provider-specific implementation.

This variation impacts how you analyze burst duration. For instance, AWS and Azure provide explicit CPU credit mechanisms with computingforgeeks.com measurable burn and recharge rates, whereas Google shared-core instances rely on best-effort sharing without strict credit accounting. Thus, burst definition and throttling aren't necessarily uniform.

Measuring CPU Burst Duration: Observation Window Matters

You cannot optimize for burst duration without measuring correctly. Many teams rely on coarse metrics averaged over hours or days—which hide burst dynamics completely.

Instead, I recommend:

- **Use fine-grained metrics:** Collect CPU usage at 1-minute or sub-1-minute granularity to observe microbursts.
- **Analyze P95 and P99 percentiles:** Focus on higher percentiles to understand peak demand rather than mean utilization.
- **Track spike duration:** Identify how long bursts last at a given percentile. For example, does the P99 usage >80% last 30 seconds or 10 minutes?
- **Record recurrence patterns:** Are bursts isolated events or a steady stream? Continuous bursts will exhaust credits faster.

Cloud monitoring tools, such as CloudWatch on AWS or Azure Monitor, allow you to configure detailed insights that can highlight these critical metrics. You want to know not just how often CPU gets hammered but for *how long* at scale.

Percentiles and Spike Duration: Why Averages Lie

To emphasize, here's why averages mislead:

Imagine your CPU spends 95% of the minute below 20% usage but spikes for 3 seconds at 100% during a critical task. The average CPU usage for that minute might be 23%, which looks low—yet the spike could cause noticeable slowdowns or throttling on shared CPU instances.

Percentiles bring context:

- **P50 (median):** The midpoint CPU usage is often very low on burstable instances.
- **P95:** Shows what your workload looks like 95% of the time, identifying sustained bursts.
- **P99:** Captures the extreme tail of usage—a great indicator for rare but impactful burst duration.

By combining percentile data with spike duration analysis (how long these high CPU usages endure), you get a clearer picture of whether burst durations are short enough to rely on shared CPU or long enough to require dedicated cores.

How AWS Compute Optimizer and Azure Advisor Help Identify Burst Duration Issues

In real-world migrations and fleet optimizations, I've found two cloud-native tools particularly valuable:

1. **AWS Compute Optimizer:** It provides recommendations based on historical utilization patterns across CPU, memory, and network. Crucially, it integrates P95 usage metrics to help identify workloads exceeding burst limits. If you see consistent throttling warnings and recommendations to switch to larger or dedicated instances, it's a sign your burst durations are too long.
2. **Azure Advisor:** Incorporates VM performance and usage telemetry to suggest resizing opportunities. Azure Advisor's recommendations often cite high CPU credit consumption for B-series VMs, highlighting when the workloads exceed burst capabilities and require dedicated vCPUs.

Both tools are invaluable for baseline tuning and identifying the recurring burst patterns that cause live production issues.

Practical Guidelines: When Shared CPU Is and Isn't Appropriate

Based on years of running cost reviews and migrations, here's what has actually worked:

- **Shared CPU is great if:**
  - CPU bursts are *very short* (e.g., under 5 seconds consistently).
  - Burst recurrence is *low frequency* (minutes apart).
  - The workload has *low sustained CPU demand* (~10-20% baseline or lower).
  - Latency spikes during bursts are *non-critical* and can tolerate throttling.
- **Shared CPU is problematic if:**
  - Burst duration frequently exceeds credit replenishment intervals (e.g., >5 minutes on AWS/Azure).
  - Recurrence is high—bursts happen multiple times per minute or are effectively constant.
  - Sustained CPU demand on average exceeds baseline plus credits.
  - Service has strict latency SLAs requiring consistent CPU availability.

Example: Measuring Bursts for a Worker Queue Service

One internal tool I reviewed ran on AWS t3.small instances with 2 vCPUs and a baseline of 20% CPU utilization. Using CloudWatch metrics sampled at 1-minute intervals, the analysis revealed:

- P95 CPU usage was ~65%
- P99 CPU usage was ~85%
- Spike durations at >80% CPU lasted 4-7 minutes, multiple times per hour
- Idle periods were too short to replenish CPU credits

Despite the average CPU hovering around 15%, the sustained bursts caused credit exhaustion and throttling during peak processing.

Outcome: Migrating to a modestly larger m5.large instance with dedicated vCPUs eliminated throttling, improved latency, and simplified autoscaling logic—while only increasing costs by 20%, which was easily justified by smoother performance.

Rollback Criteria: Before You Pilot a Switch

One quirk I always observe in migrations is failing to define rollback criteria that identify if a move to shared or dedicated CPUs worsens performance or increases costs unmanaged. For example:

- Monitor CPU throttling metrics (AWS CloudWatch metric: CPUCreditBalance, Azure analogous metrics)
- Track latency percentiles (P95/P99 response time) closely for 24-48 hours after migration
- Verify cost impact, including storage and egress, not just CPU instance price

Having these criteria upfront saved multiple projects from getting stuck on "paper" optimizations that didn't hold up under load.

Summary

Deciding how long CPU bursts can last on shared CPU setups without becoming problematic boils down to a detailed understanding of your workload's:

- **Burst duration and recurrence**—short, infrequent bursts work well on shared CPU; sustained or frequent bursts do not
- **Sustained demand**—if your baseline CPU usage plus spikes exceed credit replenishment rates, expect throttling
- **Cloud provider-specific throttling and credit models**—these must shape your observation window and capacity planning
- **Use of percentiles and fine-grained data**—P95/P99 and spike duration > averages are key metrics

Tools like AWS Compute Optimizer and Azure Advisor provide actionable insights based on these metrics and can guide informed resizing decisions.

Never rely on averages or vague cost estimates when sizing small, always-on services. Instead, dig into the P99 spikes, duration of bursts, and how frequently they recur—this will help you allocate the right CPU resources, control costs, and meet your SLAs.

Further Reading and Tools

- AWS Compute Optimizer documentation

- Azure Advisor overview
- AWS CPU Credit and Burstable Instance docs
- Azure B-series VM CPU credit docs