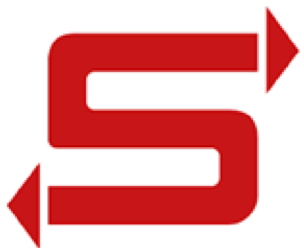
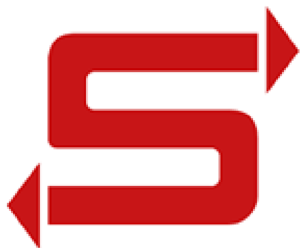
Introduction



Taming Async I/O in Samba: Inside vfs_aio_ratelimit

Avan Thakkar
@IBM



Avan - avanthakkar

- Software Engineer at IBM
- 5+ years of experience working with Ceph Storage, currently working with Ceph-SMB team.
- Like to watch different Sports, reading HackerNoon articles in weekend afternoons.

Agenda

- In this session we'll cover:
    - Why rate limiting is needed in SMB(especially CephFS) deployments
    - Design of `vfs_aio_ratelimit`
    - How we enforce limits in async I/O path
    - Evolution: from basic limits → burst-aware model
    - Challenges in scaling to cluster-wide solution
    - Integration with Ceph SMB QoS
    - Live demo of rate limiting in action

The Problem and Motivation

- Growing use of **CephFS-backed SMB deployments**
- Multi-tenant environments → shared resources
- “Noisy neighbor” problem
- Lack of **per-share I/O control**
- Need for predictable performance

What We Needed: Design Requirements

- Per-share granularity
- Async I/O compatible
- Stackable VFS module
- Minimal performance overhead
- IOPS and bandwidth controls
- Configurable via smb.conf

Architecture: Stackable VFS Module

- Hooks into async read/write paths (`pread`, `pwrite`)
- Checks available tokens (IOPS / bandwidth)
- Calculates delay based on token state
- Injects delay via tevent timer
- Forwards request to next VFS module
- **Configuration:** vfs objects = aio_ratelimit ceph_new

How It Works: Token Bucket Algorithm

Core Idea

- Tokens = allowed I/O capacity
- Bucket has maximum capacity
- Tokens refill at configured rate
- Each I/O consumes tokens
- If tokens go negative → delay the request

What Happens Per Request

- **Refill:** Add tokens based on elapsed time. If idle long enough, return full capacity.
- **Consume:** Subtract tokens for this I/O
- **Check:** Token balance negative?
    - Yes → Calculate delay, inject timer
    - No → Proceed immediately

Basic Configuration & Operation

Configuration example:

```
[global]
        clustering = Yes
        registry shares = Yes
        smbd profiling level = on
        ctdb:registry.tdb = yes
[share1]
        path = /volumes/group1/subvol1/9eade4dd-e709-4668-88ab-d695472583ed/
        read only = No
        smbd profiling share = Yes
        vfs objects = acl_xattr ceph_snapshots aio_ratelimit ceph_new
        aio_ratelimit:write_delay_max = 5
        aio_ratelimit:write_bw_limit = 2097152
        aio_ratelimit:write_iops_limit = 200
        aio_ratelimit:read_delay_max = 5
        aio_ratelimit:read_bw_limit = 1048576
        aio_ratelimit:read_iops_limit = 100
```

Initial Implementation – What Worked & What Didn't

What Worked Well

- Per-Process Rate Limiting
  - Each smbd process enforced limits independently
- **Dual Control: IOPS and Bandwidth**
  - Separate limits for operations and throughput
  - Independent read and write controls
- **Non-Blocking Implementation**
  - Proper async I/O handling
  - No event loop blocking

Where It Fell Short

- Per-Process, Not Cluster-Wide
- No Burst Handling
- No State Persistence
- Raw Byte Configuration: write_bw_limit = 104857600

Enhancement #1 – The Burst Challenge

Real Workloads Are Bursty

- Periods of idle/low activity
- Sudden bursts of activity
- Not constant steady stream

What Goes Wrong with Hard Limits

- Bursts get throttled immediately. no buffer for short-term bursts
- Causes stuttered performance
- Long-term average could be well below limit!

Solution: Burst Multiplier

- Allow short-term bursts above steady-state rate while still enforcing long-term ceiling.

How it works:

- Traditional Token Bucket:
  - Capacity = Rate
  - Example: 5000 IOPS limit = 5000 token capacity
- With Burst Multiplier:
  - Capacity = Rate × Burst Multiplier
  - Example: 5000 IOPS × 1.5 = 7500 token capacity
  - 1.5x is default
- Burst Configuration

```
[share]
    vfs objects = aio_ratelimit ceph_new

    # Global limits
    aio_ratelimit:read_iops_limit = 10000

    # Burst (optional)
    aio_ratelimit:read_burst_mult = 15 (default = 1.5x)
    aio_ratelimit:write_burst_mult = 15 (default = 1.5x)
```

Enhancement #2: Human-Readable BW Limit

- Earlier, bandwidth limits had to be configured in raw bytes, which isn't exactly human-friendly.

aio_ratelimit:write_bw_limit = 104857600
aio_ratelimit:read_bw_limit = 1073741824

- Users can now specify read and write bandwidth limits using standard size units (K, M, G) instead of raw bytes. Supported units:K (KiB), M (MiB), G (GiB). Raw byte values remain supported for backward compatibility.

aio_ratelimit:write_bw_limit = 100M
aio_ratelimit:read_bw_limit = 1G

Enhancement #3: State Persistence Challenge

The Issue:

Token bucket state lives in memory only:

- smbd restart (planned or crash)
- VFS module reload
- Share disconnect/reconnect
- Any process termination

Result: Token state reset to zero, full burst capacity available

Impact at Scale

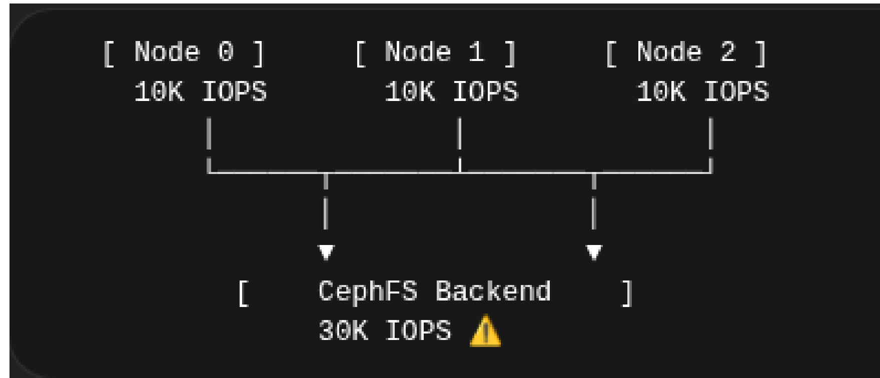
- Per-node limits → amplified at cluster level
- Short spikes can overload backend systems and further undermines reliability of QoS guarantees

The Goal: Preserve token state across all these events.

TDB State Persistence

- Persist rate limiter state using TDB
- Store per-share, per-operation state (read/write)
- Persist key fields: timestamp, IOPS tokens, bandwidth tokens, version for future schema changes
- Key format: `share/<name>/<operation>` (e.g. `share/data/read`)
- Location: /var/lib/samba/aio_ratelimit.tdb
- State is saved periodically (~30s), not on every I/O
- Small state loss window acceptable (covered by burst capacity)
- On restart or reconnect → state restored → no sudden full burst

Enhancement #4: The Cluster Problem



- Clustered deployment (CTDB) with multiple `smbd` nodes
- Each node enforces limits independently
- Configured limit applies per node, not globally
- Actual load = limit × number of nodes
- As cluster scales the problem amplifies
- Breaks QoS guarantees and overloads backend

Cluster Coordination Requirements

- Enforce global limits across cluster (not per node)
- Ensure fair distribution across active processes
- Automatically adapt as nodes/processes join or leave
- Use efficient coordination (avoid excessive messaging)
- Scale to large clusters
- Support graceful degradation (fallback to per-node limits if needed)
- Maintain fail-safe behavior (never break existing functionality)
- Minimal performance overhead
- Avoid single points of failure; handle node failures cleanly

Why Direct P2P Messaging Doesn’t Scale

- Naive approach: every process broadcasts to every other process
- Each process reports its current I/O usage periodically
- All processes aggregate global state locally
- Message complexity grows as $O(N^2)$
- High network and CPU overhead just for coordination
- Becomes impractical as cluster size increases
- Insight: process-level coordination is too expensive
- Better approach: coordinate at node level, not process level

Why Peer-to-Peer Doesn't Scale

The Naive Approach: Direct Process-to-Process Broadcasting

What Each Process Broadcasts:

```
{
  "pid": 12345,
  "recent_iops": 150,
  "inflight_ios": 5,
  "operation": "read"
}
```

Sent to: **Every other process**
Frequency: **Once per second**
Purpose: **Let others know I'm active**

How Each Process Uses This:

1: Receive messages from all others
2: Count total active processes
3: Calculate fair share:

```
my_limit = global_limit / total_active
```

4: Update token bucket with new limit
5: Repeat every second

Process 1
Process 2
Process 3
Process 4
Process 5

N × (N-1) messages per second

Processes	Msgs/sec	Status
10	90	✓
50	2,450	⚠
100	9,900	✖
500	249,500	✖
1,000	999,000	[illegible]

Message Volume Growth ($O(N^2)$)

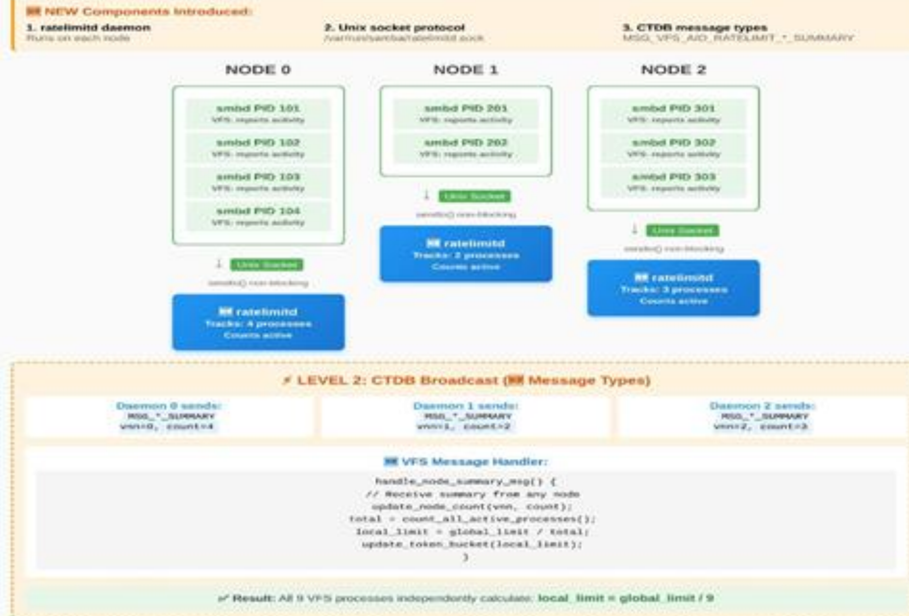
Processes

Why This Fails:

- Network floods with coordination traffic
- CPU overhead processing thousands of messages
- Scales quadratically — double processes = 4× messages
- 100 processes already problematic
- 1000+ processes completely impossible

Solution: Two-Level Coordination

- Key idea: group processes by node → introduce hierarchy
- Avoid process-level mesh; coordinate at node level
- Two communication levels: local + cluster-wide
- Local: each `smbd` reports to a node-local daemon
- Cluster: daemons exchange node-level summaries
- Uses existing messaging infrastructure
- Reduces complexity from $O(N^2)$ → $O(M^2)$, where M = no. of nodes in cluster


Solution: Two-Level Coordination
Daemon-Based Architecture
Key Insight: Processes are grouped by node. Use hierarchy!
LEVEL 1 Local (VFS → Daemon)
Protocol: Unix domain socket
Direction: Each smbd → LOCAL daemon only
Frequency: Once per second
Mode: Fire-and-forget (non-blocking)
Payload: Activity report (IOPS, inflight)
Purpose: Local aggregation
LEVEL 2 Cluster (Daemon ↔ Daemons)
Protocol: CTDB messaging
Direction: Daemon → ALL nodes (broadcast)
Frequency: Once per second
Mode: Reliable broadcast
Payload: Node summary (process count)
Purpose: Cluster-wide coordination
Complete Architecture Flow
NEW Components Introduced:
1. ratelimitd daemon
Runs on each node
2. Unix socket protocol
/var/run/samba/ratelimitd.sock
3. CTDB message types
MSG_VFS_AIO_RATELIMIT_*_SUMMARY
NODE 0
smbd PID 101
VFS: reports activity
smbd PID 102
VFS: reports activity
smbd PID 103
VFS: reports activity
smbd PID 104
VFS: reports activity
Unix Socket
sendto() non-blocking
ratelimitd
Tracks: 4 processes
Counts active
NODE 1
smbd PID 201
VFS: reports activity
smbd PID 202
VFS: reports activity
Unix Socket
sendto() non-blocking
ratelimitd
Tracks: 2 processes
Counts active
NODE 2
smbd PID 301
VFS: reports activity
smbd PID 302
VFS: reports activity
smbd PID 303
VFS: reports activity
Unix Socket
sendto() non-blocking
ratelimitd
Tracks: 3 processes
Counts active
LEVEL 2: CTDB Broadcast (Message Types)
Daemon 0 sends:
MSG_*_SUMMARY
vnn=0, count=4
Daemon 1 sends:
MSG_*_SUMMARY
vnn=1, count=2
Daemon 2 sends:
MSG_*_SUMMARY
vnn=2, count=3
VFS Message Handler:
handle_node_summary_msg() {
// Receive summary from any node
update_node_count(vnn, count);
total = count_all_active_processes();
local_limit = global_limit / total;
update_token_bucket(local_limit);
}
Result: All 9 VFS processes independently calculate: local_limit = global_limit / 9
Peer-to-Peer (N^2)
100 processes across 5 nodes
9,900
messages/second
Scales with processes squared
Daemon-Based (M^2)
100 processes across 5 nodes
5
messages/second
Scales with nodes (typically 3-16)
Why This Works
Local aggregation: Daemon does heavy lifting, VFS just reports
Hierarchy: Message volume scales with nodes (M), not processes (N)
Leverages CTDB: Uses existing reliable messaging infrastructure
Non-blocking: VFS reports fire-and-forget, never blocks I/O
Scalable: Works from 10 to 1000+ processes without issue

Enabling Cluster-Wide Rate Limiting

- Requires Samba built with ratelimitd support
  - *./configure <other-build-opt> --with-ratelimitd*
  - Daemon starts automatically when the requirement is met
- Given `clustering=yes`
  - *cluster_mode:*
    - yes -> uses cluster-wide logic. Default value.
    - no -> per-node only. Useful incase if there're any failures.
- If unavailable → graceful fallback to per-node limiting

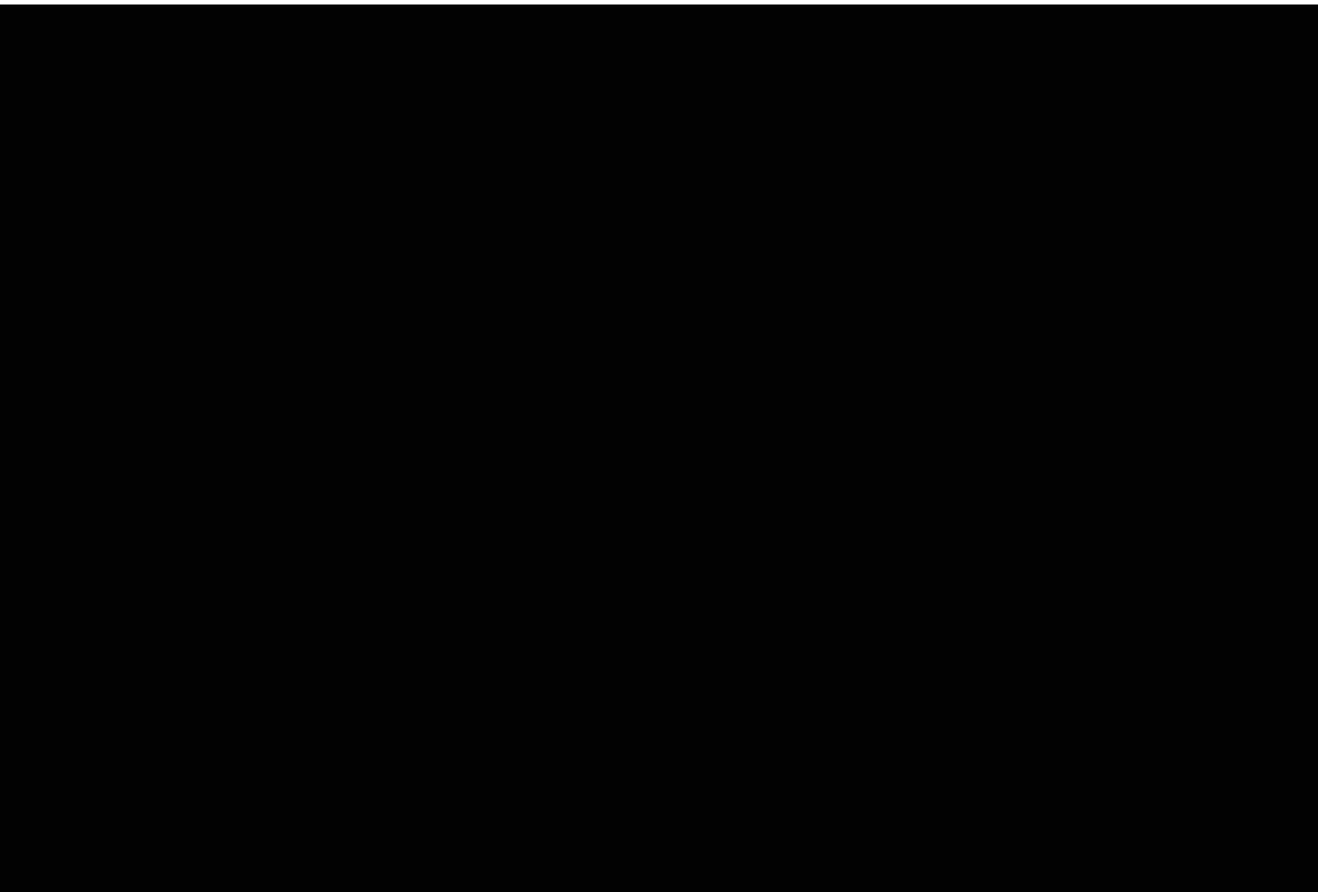
Configure smb.conf:

```
[global]
    clustering = yes

[share]
    path = /mnt/cephfs/data
    vfs objects = aio_ratelimit ceph_new

    # Global limits
    aio_ratelimit:read_iops_limit = 10000
    aio_ratelimit:write_bw_limit = 1G

    # Burst (optional)
    aio_ratelimit:read_burst_mult = 15
    aio_ratelimit:write_burst_mult = 15

    # Cluster coordination (optional)
    aio_ratelimit:cluster_mode = yes
```

Integration with Ceph SMB

- Integrated with Ceph Manager module
- Managed via `ceph smb` CLI commands
- Part of samba-container images
- All management commands and deployment setup will be shown in demo

Demo Time



What's Next

Diagnostic tooling (`samba-tool ratelimit`)

- Inspect limits, token state, cluster status
- Debug issues without restart
- Faster production troubleshooting

Per-client rate limiting

- Extend QoS beyond share → user/client-level control
- Fair usage within shared environments

References

Samba Upstream Contributions:

- Initial implementation (IOPS & bandwidth):
  https://gitlab.com/samba-team/samba/-/merge_requests/4186
- Burst support & human-readable units:
  https://gitlab.com/samba-team/samba/-/merge_requests/4396
- Cluster-wide coordination (daemon-based):
  https://gitlab.com/samba-team/samba/-/merge_requests/4436

Ceph Upstream Contributions:

- Initial QoS support in Ceph Manager:
  https://github.com/ceph/ceph/pull/64818
- Burst multiplier & cluster mode parameters:
  https://github.com/ceph/ceph/pull/67435
- Configuration enhancements & validation:
  https://github.com/ceph/ceph/pull/67684

Documentation:

- Samba vfs_aio_ratelimit man page is a part of MR mentioned before #4436 (MR yet to be merged)
- Ceph SMB QoS configuration reference:
  https://github.com/avanthakkar/ceph/blob/962f97c416e2301721ad81c3a626ad5db707cc55/doc/mgr/smb.rst#update-share-qos

Q&A

Questions?