Distributed Asynchronous Object Storage (DAOS)

DAOS Quality of Service

Liang Zhen et al.

DUG’23; Monday 13-Nov-2023; 9:00am – 12:30pm MST

Background

- Today, DAOS has no QoS framework and simply uses FIFO as I/O requests queue
  - Cannot guarantee fairness or request priority
- DAOS client has no mechanism to throttle RPC sending
  - Clients nodes can run O(100) MPI processes, each sending huge # of RPCs
- DAOS engine has no throttling on RPC receiving
  - Engine can underperform or be killed by OOM killer when not keeping up processing RPCs

High-level design of DAOS QoS includes three areas:

- QoS framework to guarantee fairness between different users and to support RPC priority.
- Server-side throttling, which should prevent a server from indefinitely receiving incoming requests and eventually losing the capability of processing requests.
- Client-side throttling, which should prevent a client from sending out an unlimited number of RPCs.

New Concept: DAOS QoS Session

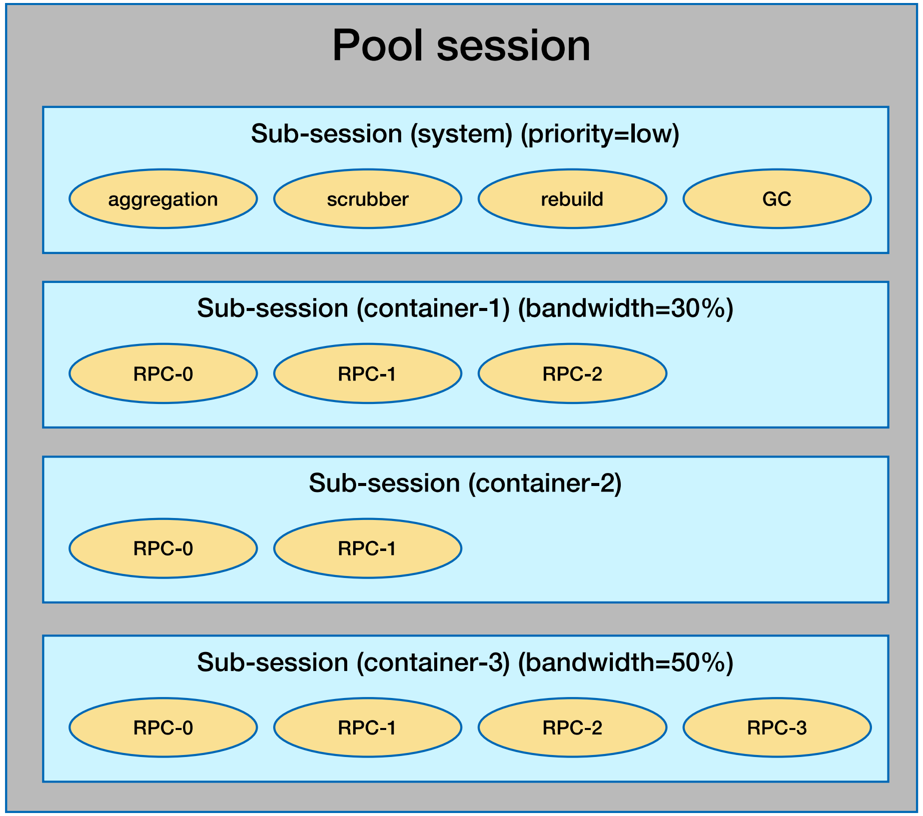
- A session includes either sub-sessions or tasks
  - Task is the execution unit
- A session describes server resources allocated for the associated dataset(s)
  - Server resource is quantified as “credits”.
  - Tasks under a session (direct and indirect) cannot consume more “credits” than the assigned quantity
- Sub-session scheduling schema
  - Example: a sub-session (scrubber) is low priority, its parent will not poll task from it unless it is idle
- Task scheduling policy
  - Default policy of a session is FIFO
  - More policy can be defined, e.g., client-based round-robin

QoS Task

- Task is execution unit
  - Attached to a session
  - Callback function, e.g., RPC handler
- Credits
  - Operation credit: an I/O request consumes one operation credit
  - Payload credit: an I/O request consumes (size/1024) payload credits
- Task types
  - User task: regular I/O request
  - System task: aggregation, checksum scrubber, rebuild, space GC…

DAOS storage model and session/task

- Session per pool
  - Optional: sub-session per pool connection
- Sub-session per container
  - Optional: sub-session per container handle
- Default schema of pool session
  - System sub-sessions
    - aggregation, rebuild, scrubber, GC…
  - User sub-session
    - client requests
  - Scheduling schema
    - For example: for an append-only pool, “aggregation=low, scrubber=low, rebuild<=30%”


Pool session
Sub-session (system) (priority=low)
aggregation
scrubber
rebuild
GC
Sub-session (container-1) (bandwidth=30%)
RPC-0
RPC-1
RPC-2
Sub-session (container-2)
RPC-0
RPC-1
Sub-session (container-3) (bandwidth=50%)
RPC-0
RPC-1
RPC-2
RPC-3

Default schema/policy

- Session per pool
  - Credits are evenly distributed to active pools
  - Poll tasks from each pool session in round-robin manner
- Pool session has one sub-session per-container
  - System sub-session (rebuild, aggregation, scrubber…) is shared
  - Poll tasks from each container in round-robin manner
- Container sub-session has a task queue
  - FIFO (advanced task selection policy in the future)

Administrator Interface

- Admin can change the overall resources assigned to a pool session
  - Assign 50% of operation & bandwidth credits to an important pool
  - Other pools share the other 50%
- Admin can change schema/policy of a pool session
  - Prioritize/deprioritize system sub-session
  - Activate/deactivate system sub-session
  - Change credits for each sub-session
    - Only allow system services to occupy 10% of the bandwidth credits
    - Only allow a low-priority container to consume 5% of the bandwidth

Administrator Interface – Custom Session

- Create a custom session
- Add pool or container to a custom session
  - Each pool or container is a sub-session
- Create a system sub-session
  - Assign dedicated rebuild, aggregation, …
    - Example: no rebuild and scrubber for scratch data
- A pool or container can only belong to one session

Server RPC throttling

- Today, server indefinitely creates tasks for incoming requests
  - Pin all the resources
- Server cannot process queued requests/tasks at full speed
  - Spend CPU cycles to select request from millions of them
  - High memory footprints
  - Not enough memory, eventually killed by OOM killer
- Solution: Reject incoming requests based on # queued requests
  - Returns a hint for retry based on historical statistics within the session

Client RPC throttling

- DAOS client is a userspace library
  - It can limit number of RPC sending by a single client process … but:
  - A node can create hundreds of processes, submit tens of thousands of RPCs
- Solution: DAOS agent creates a shared memory for node-wide coordination
  - Session ID is maintained in shared memory
  - Assign RPC credits to each session ID
  - A process should take a credit before sending RPC
    - No credit: queue the RPC locally, poll the shared memory for credit

intel®