NRS allows the PTLRPC layer to reorder the servicing of incoming RPCs.

Predominantly server-based, although the clients could play a part in certain scenarios.

Why do this?

Different reasons:

Performance improvements:
- Increased throughput on disk drives by producing disk-friendly RPC streams.
- Increased throughput by balancing network traffic and exploiting aspects such as locality of reference on client nodes.

New functionality:
- Prioritization amongst cluster clients.
- Various forms of Quality of Service, at the filesystem level.
NRS Framework

- NRS core manages available policies.
- PTLRPC services denote what policies they support at service initialization time.
  - These are checked against the available policies from NRS core.
  - Policy assignment for each service is done from NRS core.
    - If present, first policy with Dominant flag set is chosen at service initialization time, but this scheme can be more elaborate if required.
- Each service has an Active and a Secondary policy.
  - Active policy handles all incoming RPCs.
  - Active policy can delegate unsupported RPC types to Secondary policy.
  - Default (FIFO) policy will be adequate to act as Secondary in most cases.
- Active policy changeable at run-time via lprocfs on a per-service basis.
- Policies can be disabled via lprocfs on a per-service basis.
NRS policies

- NRS can implement different policies, in order to satisfy different end goals.
  - FIFO, existing functionality wrapped in an NRS policy.
  - OBRR (Object-Based Round Robin).
    - RPCs are grouped per-object, and according to file offset.
    - Aims to provide higher throughput by reducing disk seeks.
  - CBRR (Client-Based Round Robin).
    - RPCs are grouped per client (export).
    - Aims to balance network traffic.
  - PBRR (PID-Based Round Robin).
    - RPCs are grouped according to NID::PID of the application that initiates I/O.
    - Similar to CBRR; aims to also exploit locality of reference at clients.
  - Client or User Prioritization policy.
    - Aims to offer a form of QoS by giving higher priority to more important parts of the workload.
    - Importance can be determined by different means, to achieve different goals.
  - Variable-Slice adaptations of the above algorithms can allow for further forms of control, e.g. OBVS (Object-Based Variable Slice).
NRS policy “obj_extents”

• First policy we plan to implement.
• Is an implementation of OBRR.
  • Operates on OSS nodes.
  • Handles OST_READ and OST_WRITE RPCs.
  • RPCs are grouped in number-limited or size-limited per-object groupings; this circumvents problems related to request starvation.
  • Per-object groups are sorted in either linked lists or rbtrees, tbd, depending on expected size.
  • Could benefit from using a scalable data structure like the binary heap WC are using.
• Concerns have been expressed about the effectiveness an elevator-like policy like obj_extents may have; performance measurements would be good to have in any case.
Considerations

- Scalable data structures and perhaps cache-friendly accesses are of consideration.
  - Large number of requests, many concurrent threads.
- May be beneficial to have more than one policy per-service operating concurrently.
- WC effort is taking place in parallel; seems to have a good number of things right: priority queue data structure for policies, and various other ideas.
  - Need to either merge with WC effort, or otherwise have only one ongoing effort; there is no benefit to the codebase from having two efforts in parallel.
- Need to obtain some performance measurements to ascertain the extent of the validity of NRS as a concept, and of specific policies.
Thank you!