Token Bucket Filter Policy

1. Introduction

The Lustre Network Request Scheduler (NRS) enables services to schedule RPCs in various ways. Using this new framework, various scheduling policies have been implemented. Most of these policies are aimed at improving I/O throughput, but the Token Bucket Filter (TBF) policy is aimed at a very different purpose, namely (client) I/O Quality of Service QoS). It tries to control the RPC rate limit on clients/jobs for QoS (Quality of Service).

TBF policies are most widely used as a rate control mechanism, especially in packet-switched computer networks and telecommunications. Their goal is to ensure that data transmission rates conform to given limits for bandwidth and burstiness. A simple TBF algorithm shown in Animation 1 can be conceptually understood as follows:

Animation1. Token bucket filter algorithm

- A token is added to the bucket every 1/r seconds.
- The bucket can hold at the most b tokens. If a token arrives when the bucket is full, it is discarded.
- When a packet of n bytes arrives, n tokens are removed from the bucket, and the packet is sent to the network.
- If fewer than n tokens are available, no tokens are removed from the bucket, and the packet is considered to be non-conformant.

Considering the Lustre file system, the TBF policy has been implemented as follows:

- Non-conformant packets can be treated in various ways in the TBF algorithms. Network traffic control systems normally drop non-conformant packets. But our NRS TBF policy enqueues non-conformant RPC requests, rather than simply drop them.
- Network traffic control systems are able to calculate the size of packets and thus limit the throughput rates precisely in bytes. The NRS TBF policy is facing a different situation. There exist many different kinds of Lustre RPC, and developing a straightforward method to measure the number of tokens that different RPCs consume appears tricky. Thus, the current TBF policy only supports a RPC rate limit, i.e. one RPC always consumes one token.
- For system administrators, being able to set various types of limits, and for different dimensions, would be highly desirable. For example, we might not only want to limit the RPC rate of each client, but also want to limit the RPC rate of each user. A powerful QoS mechanism should be flexible enough for this kind of requirements. Network traffic control systems basically stack multiple TBFs (or other kind of traffic control algorithms) layers in order. Today, our TBF policy only supports NID based limits, but we are working on a common layer within the TBF policy that would enable the support other types of RPC limitations, such as limit based on UIDs/GIDs even Job ID.) Further, as soon as the NRS framework supports layered polices, we will be able to define more complex policy within TBF.

2. Design

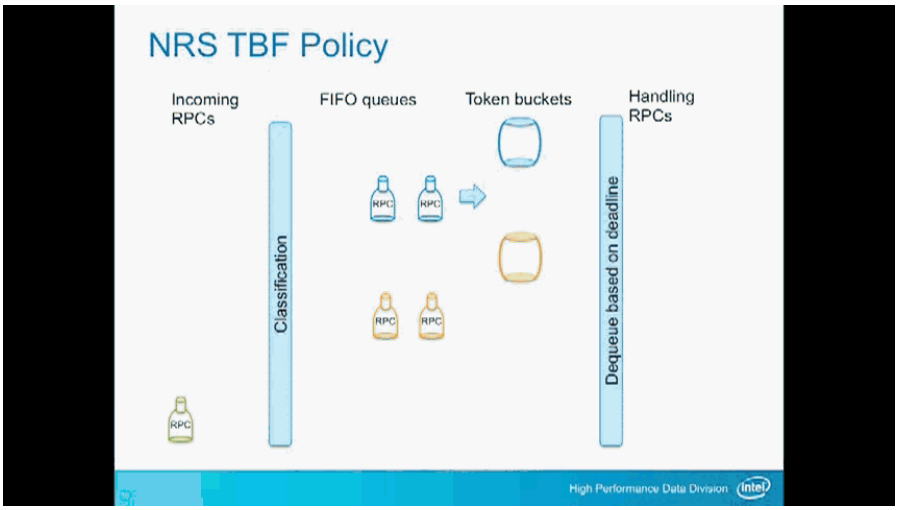
The internal structure of TBF policy is illustrated in Figure1 as follows:

Figure1. Internal structure of TBF policy

When a RPC request arrives, TBF policy puts it to a waiting queue according to its classification. The classification of RPC requests is based on either NID or JobID of theRPC according to the configure of TBF. TBF policy maintains multiple queues in the system, one queue for each category in the classification of RPC requests. If there is no corresponding queue for an incoming RPC request, a new queue will be created immediately. Queued requests wait for tokens to be assigned in a FIFO manner. A request will never be dequeued unless it is the first one in the FIFO queue and obtains enough tokens. Every queue has a token bucket to manage tokens. The requests wait for tokens in the FIFO queue before they have been handled so as to keep the RPC rates under the limits.

To manage the RPC rate of queues, we don't need to set the rate of each queue manually. Instead, we define rules which TBF policy matches to determine RPC rate limits. All of the defined rules are organized as an ordered list. Whenever a queue is newly created, it goes though the rule list and takes the first matched rule as its rule, so that the queue knows its RPC token rate. A rule can be added to or removed from the list at runtime. Whenever the list of rules is changed, the queues will update their matched rules. The queue deadline is the time point when the queue will have accumulated enough tokens to dequeue the RPC request. Queues are sorted in a binary heap according to the deadlines of their first RPC requests. The binary heap waits until it reaches the nearest deadline. Then the RPC request with the smallest deadline will be dropped from the heap and handled by an idle service. Sometimes, the service is just too busy to handle all of the requests in time, thus deadlines are missed.

The basic NRS TBF policy is illustrated in Animation 2.



Animation2. NRS TBF policy

When Lustre services are too busy to handle all of the requests in time, all of the specified rates of the queues will not be satisfied. Nothing bad will happen except some of the RPC rates are slower than configured. In this case, the queue with higher rate will have an advantage over the queues with lower rates, but none of them will be starved.

3. Usage

Before enabling any NRS policies, you can run the following command to check which NRS policies are being applied in the system.

```
#lctl get_param x.x.x.nrs_policies
```

Please note that a newly started rule is prior to old rules, so the order of starting rules is critical.

3.1 Enable TBF policy

The command to enable a TBF policy is shown as follows:

```
#lctl set_param x.x.x.nrs_policies="tbf [reg|hp] <type>"
```

The argument "type" is the classification type of RPC requests. Currently, only "nid" and "jobid" are supported.

For example,

```
#lctl set_param ost.OSS.ost_io.nrs_policies="tbf nid"
#lctl set_param ost.OSS.ost_io.nrs_policies="tbf reg nid"
#lctl set_param ost.OSS.ost_io.nrs_policies="tbf hp nid"
```

3.2 Start rule

The command to start a TBF policy rule is shown as follows:

```
#lctl set_param x.x.x.nrs_policies="[reg|hp] start <rule_name> <arguments>..."
```

The argument "rule_name" argument is the TBF policy rule name. The format of the ‘arguments’ is different per the different TBF policy type.

3.2.1 NID based TBF policy

For NID based TBF policy, the format is shown as follows:

```
#lctl set_param x.x.x.nrs_policies="[reg|hp] start <rule_name> {<nid_list>} <rate>"
```

The 'nid_list' argument uses the same format used to configure an LNET route. The 'rate' argument is the RPC rate of the rule.

For example,

```
#lctl set_param ost.OSS.ost_io.nrs_tbf_rule="start loginnode {192.168.1.1@tcp} 10000"
#lctl set_param ost.OSS.ost_io.nrs_tbf_rule="reg start loginnode {192.168.1.1@tcp} 10000"
#lctl set_param ost.OSS.ost_io.nrs_tbf_rule="hp start loginnode {192.168.1.1@tcp} 10000"
#lctl set_param ost.OSS.ost_io.nrs_tbf_rule="start computenodes {192.168.1.[1-128]@tcp} 1000"
#lctl set_param ost.OSS.ost_io.nrs_tbf_rule="start other_clients {192.168.*.*@tcp} 100"
```

3.2.2 JobID base TBF policy

For JobID based TBF policy, the format is shown as follows:

```
#lctl set_param x.x.x.nrs_policies="[reg|hp] start <name> {<jobid_list>} <rate>"
```

The 'jobid_list' format is {job1.id1 job2.id2 ...}. The 'rate' argument is the RPC rate of the rule.

For example,

```
#lctl set_param ost.OSS.ost_io.nrs_tbf_rule="start user1 {iozone.500 dd.500} 100"
#lctl set_param ost.OSS.ost_io.nrs_tbf_rule="start iozone_user1 {iozone.500} 100"
```

Same as nid, could use reg and hp rules separately:

```
#lctl set_param ost.OSS.ost_io.nrs_tbf_rule="hp start iozone_user1 {iozone.500} 100"
#lctl set_param ost.OSS.ost_io.nrs_tbf_rule="reg start iozone_user1 {iozone.500} 100"
```

3.3 Change rule

The command to change a TBF policy rule is shown as follows:

#lctl set_param x.x.x.nrs_policies="[reg|hp] change <rule_name> <rate>"

For example,

```
#lctl set_param ost.OSS.ost_io.nrs_tbf_rule="change loginnode 1000"
#lctl set_param ost.OSS.ost_io.nrs_tbf_rule="reg change loginnode 1000"
#lctl set_param ost.OSS.ost_io.nrs_tbf_rule="hp change loginnode 1000"
```

3.4 Stop rule

The command to stop a TBF policy rule is shown as follows:

#lctl set_param x.x.x.nrs_policies="[reg|hp] stop <name>"

The following commands are valid:

```
#lctl set_param ost.OSS.ost_io.nrs_tbf_rule="stop loginnode"
#lctl set_param ost.OSS.ost_io.nrs_tbf_rule="reg stop loginnode"
#lctl set_param ost.OSS.ost_io.nrs_tbf_rule="hp stop loginnode"
```

Reference:

LU-3558 LU-5580 LU-6668

https://wiki.hpdd.intel.com/download/attachments/29000881/NRS-TBF.pptx?version=1&modificationDate=1435288326000&api=v2

https://jira.hpdd.intel.com/secure/attachment/13261/TBF-design-1.0.pdf