SEE TEXT ONLY

# A REVIEW OF STORAGE Multipathing

BY ALEXANDER MOTIN

**Storage multipathing is a technique designed to improve storage reliability by eliminating a single point of failure, and/or improving its performance by distributing workload between multiple physical connections. Multipathing can be implemented on several different layers: the transport layer (like iSCSI), the peripheral device driver layer (like the SCSI block device driver), or the block storage transformation layer (like FreeBSD GEOM). Each implementation has its own benefits and downsides.**

Transport layer multipathing allows multiple physical connections to be exposed as one transport protocol connection. For example, the iSCSI protocol defines optional support for multiple connections per session (MC/S), when SCSI requests and responses are distributed between several TCP connections and are visible to upper SCSI layers as a single initiator to the target connection. This technique can be useful if initiator and/or target storage stacks do not natively support application layer multipathing. For example, Microsoft supports SCSI layer multipathing (MPIO) only in the server version of Windows, while iSCSI MC/S is fully supported even for desktops. The biggest downside of this technique is that only paths using the same capable transport protocol (like iSCSI) may participate—you cannot back up Fibre Channel fabric with iSCSI this way. Also, the requirement to provide semantics of a single SCSI initiator-target connection (request ordering, error recovery, etc.) significantly complicates the transport protocol implementation. For example, if one of the connections experiences delays or packet losses, other connections have to delay their otherwise already delivered requests while that connection recovers, or they must implement mechanisms for quick problem detection with the affected requests being rerouted via different connections. For these reasons, the new FreeBSD kernel space, the iSCSI initiator and target implementations do not support MC/S. The old user space iSCSI target implementation, istgt, declared MC/S support, but implemented it in a minimalistic way without proper error recovery, and that sometimes improved storage performance, but could also cause suffering on the reliability side.

Peripheral device driver-layer multipathing allows the execution of upper-layer requests (such as block read, write, delete, etc.) via multiple, transport-layer connections (such as iSCSI, Fibre Channel, SAS, etc.). This technique permits simultaneous use of different transport protocols only limiting to a single application protocol (such as SCSI Block Commands). You cannot backup an NVMe-over-Fabric connection with iSCSI using multipathing at this layer unless you translate one command set (NVMe) into another (SCSI). Multipathing at this layer assumes a completely different view of request execu-

tion ordering. Since the target does not know which of the multiple initiators represent the same peripheral device driver (as the source of properly ordered requests), it can not guarantee ordering for requests coming via different paths. This significantly simplifies multipathing-capable target implementation, but requires a special compilation of request distribution logic between the paths in the peripheral driver, which should prevent, or at least minimize, unwanted request reordering, probably at the cost of lower path utilization. Fortunately, the peripheral driver knows more about the nature and origination of requests, and can do it in a more clever way than is possible on transport layers.

For people familiar with network protocols, an analogy can be drawn: transport-layer multipathing is akin to Multilink PPP, which guarantees full, original ordering of all packets as required by upper layers of the PPP stack, and distributes packet fragments between the available links without looking inside; same-time, peripheral device driver-layer multipathing can be compared to LACP, which identifies individual data streams, looking at a packet's MAC and IP addresses, TCP/UDP ports, etc., to distribute them between the links in a way that would guarantee ordering, only separately within each stream. While the first approach may theoretically provide much better throughput for a low number of data streams, the second provides a less complicated stateless implementation, lower average latency, and better resiliency to individual link problems.

As an example of minimal, peripheral driver-layer multipathing implementation, we may look at SAS HDDs. Each SAS HDD typically has two SAS ports, each of which can be connected to a separate SAS expander of the backplane, and then to a separate HBA of the host, providing the host two separate SCSI `I_T_L` (Initiator-Target-Logical Unit) nexuses. A multipathing-capable SCSI peripheral driver can identify those two `I_T_L` nexuses as paths to the same Logical Unit by fetching and comparing the globally unique Logical Unit Name (can be NAA or EUI IDs, text or binary strings, UUIDs, etc.). After that, the peripheral driver can report a single block storage device to upper layers and choose a strategy to distribute workload between those two paths. Since, for a typical SAS HDD interface, throughput is much higher than media throughput, and sequential access is much faster than random, simple failover configuration, using only one of the paths at a time can be perfect. The same time for high-performance SSDs throughput of a single SAS link may not be enough to saturate the backing storage, requiring both paths to be utilized simultaneously to reach full performance.

If peripheral device-driver multipathing is not supported or cannot be used, this functionality can be implemented on a higher, block-storage transformation layer, such as GEOM on FreeBSD or Device Mapper on Linux. This layer operates in terms of abstract block read, write, delete, etc., requests, and is not related to the specifics of any transport or application protocol. It may use some out-of-band information from the lower layers or work independently. Better integration makes it possible to automate its operation, and, thus, makes it more reliable, but it also makes the storage stack more complicated. The FreeBSD GEOM MULTIPATH class, by default, operates without using any out-of-band information, relying only on its own metadata, stored at the last sector of the device, and the configuration provided by administrator. But it also has a mode of operation without using the on-disk metadata, relying on an external government of some third-party code to manage path detection, activation, prioritization, failing, etc., and by relying on the Logical Unit Names and other information.

Things become much more interesting when target ports are not equal. SCSI specifications call it Asymmetric Logical Unit Access (ALUA). ALUA can be caused by different things, but typically it is caused by the presence of several storage controllers within the target device that handle requests from different target ports, and which, for some reason, cannot work simultaneously either at all, or for some Logical Units. The degree of the asymmetricity may vary. In the worst case, some ports/controllers may be unable to even report any identification information for the Logical Unit. ALUA calls this state «`Unavailable`». This is not a very useful state, since it does not provide sufficient information even for automatically setting up multipathing. The next, better state is called «`Standby`». In this state, the port is able to report full identification of the Logical Unit, handle SCSI reservation and some management requests, but is unable to access the media. This state allows automatic multipathing configuration, but also demands full ALUA multipathing support from the initiator OS, since data access requests sent via the wrong path will fail. The next state is called «`Active/Non-optimized`». This state allows full Logical Unit access via the

path, but the performance characteristics of such access may be more or less suboptimal, for example, due to the need for additional synchronization or command/data transfer between the storage controllers. This state allows the Logical Unit to be used, even with a peripheral driver without ALUA support, but such use may be inefficient if the wrong path is selected. And finally, the best state the port may have is «`Active/Optimized`». This state means that the Logical Unit is fully accessible via this port with maximal performance. An additional ALUA state is called «`Transitioning`» and covers situations when the port or the Logical Unit is changing its state and is temporarily not accessible. The duty of the ALUA-capable initiator is to continuously monitor the state of each Logical Unit and choose the optimal path through which to send requests. In some cases, the ALUA-capable initiator may explicitly or inexplicitly request port state transitions, but the logic of those requests is outside the scope of SCSI specifications and is vendor-specific.

The multipathing operation with ALUA can be exemplified with the TrueNAS storage appliance from iXsystems Inc., which uses the High-Availability functionality of the FreeBSD CTL subsystem to implement the ALUA-capable multipath SCSI target. The TrueNAS High-Availability appliance includes two storage controllers, each running its own copy of FreeBSD 11, connected via either Ethernet or by a PCIe Non-Transparent Bridge internal link, and having access to a shared array of SAS disks. The appliance uses ZFS pools to store the user data, but since ZFS is not a clustered filesystem, only one of the controllers can access a specific pool at any point in time. SCSI requests received via iSCSI or the Fibre Channel links with the controller having no access to the required pool are proxied by CTL to the other controller via the internal link. Providing multipathing functionality without ALUA in such a situation would cause either additional latency or severe bandwidth limitations, depending on the type of internal link used. The use of ALUA allows a capable multipathing client to know the present situation and always submit requests via the most efficient paths. Let's look at some practical examples:

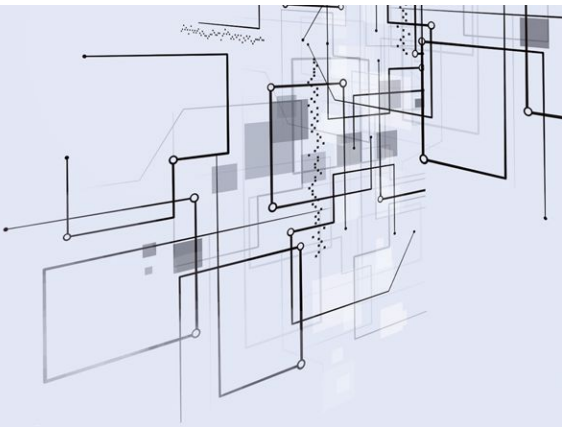
A storage system with two controllers was configured to provide one iSCSI target Logical Unit in ALUA mode. Multipath-related parts of the CTL daemon configuration on one of nodes look like this:

```
portal-group pg1A {
        tag 0x0001
        listen 10.20.20.234:3260
        foreign
}
portal-group pg1B {
        tag 0x8001
        listen 10.20.20.235:3260
}

lun "aluademo" {
        ctl-lun 0
        serial "ac1f6b0c248600"
        device-id "iSCSI Disk      ac1f6b0c248600                  "
        option vendor "TrueNAS"
        option product "iSCSI Disk"
        option revision "0123"
        option naa 0x6589cfc000000d7a21ae0e095faf3cea
}

target iqn.2005-08.com.ixsystems:aluademo {
        alias "aluademo"
        portal-group pg1A no-authentication
        portal-group pg1B no-authentication

        lun 0 "aluademo"
}
```



You may see two iSCSI portals configured with IPs of different controllers. Different portal group tags mean those portal groups represent separate SCSI ports and cannot share MC/S session connections. You may see a single SCSI target associated with those portal groups. And you may also see a single Logical Unit associated with that target, including a set of different unique IDs required for multipath operation.

Now we connect to this target using the FreeBSD iSCSI initiator:

```
# iscsictl -A -d 10.20.20.234
# iscsictl -A -d 10.20.20.235
# iscsictl
Target name                                Target portal     State
iqn.2005-08.com.ixsystems:aluademo         10.20.20.234      Connected: da0
iqn.2005-08.com.ixsystems:aluademo         10.20.20.235      Connected: da1
```

Since the FreeBSD initiator does not support peripheral driver-level multipathing, the two paths to one Logical Unit were detected as two separate block devices, `da0` and `da1`. Looking now at the storage side, we see lists of initiator and target ports inside CTL:

```
# ctladm portlist -i
Port Online Frontend Name      pp vp
0    NO     ha        1:camsim 0  0  naa.5000000341ab1b01
  Target: naa.5000000341ab1b00
1    YES    ha        1:ioctl  0  0
2    YES    ha        1:tpc    0  0
3    YES    ha        1:iscsi  1  1  iqn.2005-08.com.ixsystems:aluademo,t,0x0001
  Target: iqn.2005-08.com.ixsystems:aluademo
  Initiator 0: iqn.1994-09.org.freebsd:mini.ixsystems.com,i,0x8047c8217cb4
128  NO     camsim   camsim   0  0  naa.50000003530efb81
  Target: naa.50000003530efb00
129  YES    ioctl    ioctl    0  0
130  YES    tpc      tpc      0  0
131  YES    iscsi    iscsi    32769 1  iqn.2005-08.com.ixsystems:aluademo,t,0x8001
  Target: iqn.2005-08.com.ixsystems:aluademo
  Initiator 0: iqn.1994-09.org.freebsd:mini.ixsystems.com,i,0x808be26435c8
```

Here we see two groups of target ports, serviced by different storage controllers: ports 0-127 belonging to controller «a», ports 128-255 belonging to controller «b». Controller «b» is the one with ZFS pool access at this point. You see each controller with its own iSCSI target port with unique name. Each of the iSCSI target ports has one connected initiator port, belonging to the same initiator system, but different session IDs (not part of MC/S or a reconnection attempt).

This is where the multipath client operation starts. Using the `sg_inq` tool from the `sg3_utils` package, we fetch identification information for both block devices `da0` and `da1` reported by FreeBSD:

```
# sg_inq -p di da0
VPD INQUIRY: Device Identification page
  Designation descriptor number 1, descriptor length: 59
    designator_type: T10 vendor identification,  code_set: ASCII
    associated with the Addressed logical unit
      vendor id: TrueNAS
      vendor specific: iSCSI Disk      ac1f6b0c248600
  Designation descriptor number 2, descriptor length: 20
    designator_type: NAA,  code_set: Binary
```

*list continues*

*continued from page 7*

```
    associated with the Addressed logical unit
      NAA 6, IEEE Company_id: 0x589cfc
      Vendor Specific Identifier: 0xd7a
      Vendor Specific Identifier Extension: 0x21ae0e095faf3cea
      [0x6589cfc000000d7a21ae0e095faf3cea]
  Designation descriptor number 3, descriptor length: 48
    transport: Internet SCSI (iSCSI)
    designator_type: SCSI name string,  code_set: UTF-8
    associated with the Target port
      SCSI name string:
      iqn.2005-08.com.ixsystems:aluademo,t,0x0001
  Designation descriptor number 4, descriptor length: 8
    transport: Internet SCSI (iSCSI)
    designator_type: Relative target port,  code_set: Binary
    associated with the Target port
      Relative target port: 0x3
  Designation descriptor number 5, descriptor length: 8
    transport: Internet SCSI (iSCSI)
    designator_type: Target port group,  code_set: Binary
    associated with the Target port
      Target port group: 0x2
  Designation descriptor number 6, descriptor length: 40
    transport: Internet SCSI (iSCSI)
    designator_type: SCSI name string,  code_set: UTF-8
    associated with the Target device that contains addressed lu
      SCSI name string:
      iqn.2005-08.com.ixsystems:aluademo

# sg_inq -p di da1
VPD INQUIRY: Device Identification page
  Designation descriptor number 1, descriptor length: 59
    designator_type: T10 vendor identification,  code_set: ASCII
    associated with the Addressed logical unit
      vendor id: TrueNAS
      vendor specific: iSCSI Disk      ac1f6b0c248600
  Designation descriptor number 2, descriptor length: 20
    designator_type: NAA,  code_set: Binary
    associated with the Addressed logical unit
      NAA 6, IEEE Company_id: 0x589cfc
      Vendor Specific Identifier: 0xd7a
      Vendor Specific Identifier Extension: 0x21ae0e095faf3cea
      [0x6589cfc000000d7a21ae0e095faf3cea]
  Designation descriptor number 3, descriptor length: 48
    transport: Internet SCSI (iSCSI)
    designator_type: SCSI name string,  code_set: UTF-8
    associated with the Target port
      SCSI name string:
      iqn.2005-08.com.ixsystems:aluademo,t,0x8001
  Designation descriptor number 4, descriptor length: 8
    transport: Internet SCSI (iSCSI)
    designator_type: Relative target port,  code_set: Binary
    associated with the Target port
      Relative target port: 0x83
  Designation descriptor number 5, descriptor length: 8
    transport: Internet SCSI (iSCSI)
```

*continued from page 8*

```
    designator_type: Target port group,  code_set: Binary
    associated with the Target port
      Target port group: 0x3
  Designation descriptor number 6, descriptor length: 40
    transport: Internet SCSI (iSCSI)
    designator_type: SCSI name string,  code_set: UTF-8
    associated with the Target device that contains addressed lu
      SCSI name string:
      iqn.2005-08.com.ixsystems:aluademo
```

Comparing the outputs, we see that all reported descriptors associated with the addressed Logical Unit are identical. It means those two devices, indeed, represent the same logical unit. Descriptors associated with the target port are different and provide full information to identify them within the target device. You'll see the numbers there match the information reported by the `ctladm portlist` command.

And now, ALUA appears. The primary SCSI command for fetching ports statuses is `REPORT TARGET PORT GROUPS:`:

```
# sg_rtpg da0
Report target port groups:
  target port group id : 0x2 , Pref=0, Rtpg_fmt=0
    target port group asymmetric access state : 0x01
    T_SUP : 1, O_SUP : 0, LBD_SUP : 0, U_SUP : 1, S_SUP : 1, AN_SUP : 1, AO_SUP : 1
    status code : 0x02
    vendor unique status : 0x00
    target port count : 03
    Relative target port ids:
      0x01
      0x02
      0x03
  target port group id : 0x3 , Pref=0, Rtpg_fmt=0
    target port group asymmetric access state : 0x00
    T_SUP : 1, O_SUP : 0, LBD_SUP : 0, U_SUP : 1, S_SUP : 1, AN_SUP : 1, AO_SUP : 1
    status code : 0x02
    vendor unique status : 0x00
    target port count : 03
    Relative target port ids:
      0x81
      0x82
      0x83
```

Here we see that our target reports two port groups (one per storage controller) and 6 ports (3 per storage controller). Groups and ports identifications here match earlier outputs of the `sg_inq` and `ctladm portlist` commands. In addition to information we already know, this command tells us that each group supports a number of ALUA states, and also that port group 0x2 (controller «a») is now in «Active/Non-optimized» state (0x01), while port group 0x3 (controller «b») is in «Active/Optimized» state (0x00). It tells us that at this moment it is preferable to send all requests via the block device `da1`, but if we lose connectity via that path, the `da0` device can also handle requests, just more slowly.

Unfortunately, the FreeBSD SCSI disk peripheral driver does not support multipathing for using that information automatically. The multipathing setup on the GEOM layer has to be done either manually by the administrator, or by some external scripts, as is done, for example, by FreeNAS software for multipath SAS disks. Here we do it manually:

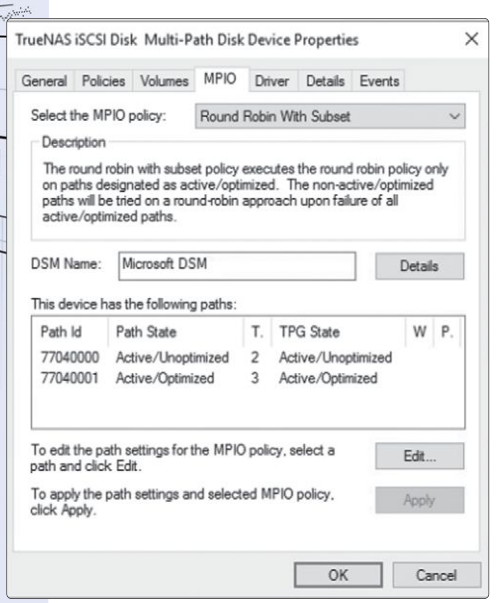
*see next page*

```
# kldload geom_multipath
# gmultipath label mp0 da0 da1
# gmultipath prefer mp0 da1
# gmultipath status
          Name    Status  Components
multipath/mp0   OPTIMAL  da0 (PASSIVE)
                         da1 (ACTIVE)
```

We see that both da0 and da1 devices do now belong as paths to the multipath/mp0 device, and the da1 device will handle the I/O until either some error occurs or the administrator commands otherwise. Some other initiators may do it completely automatically. For example, VMware vSphere automatically detects multipath devices and configures their operation using ALUA.


Here the initiator correctly identified that two paths belong to the same target, and send I/Os via the proper controller «b». If there is more than one path via that controller, it would be possible to use either the Most Recently Used (default) or the Round Robin path selection policy. In the case of the Round Robin policy, vSphere switches the active path after a certain amount of requests (1,000 by default). The fairly high default value does not allow complete throughput utilization of all paths with only one server, but, as described above, it maintains the original request ordering.

Out of the box, Windows Server only supports MC/S transport-layer multipathing. The upper-layer multipathing has to be installed as a separate, optional feature. But after the component installation and couple system reboot, if your system is still alive ;), the ALUA multipath target should be properly detected and work:




To summarize: used as a target, FreeBSD CAM Target Layer (CTL) can provide decent SCSI multipathing functionality, supporting ALUA and High-Availability clustering, and compatible with many third-party intiators. Transport layer multipathing in the case of iSCSI MC/S is not supported due to high complexity and limited scope, but it could possibly be useful in environments with many Windows desktops. Being used as the initiator, FreeBSD, at the moment, can propose only basic GEOM layer multipathing without ALUA support. To some degree, this can be compensated for by external scripting, but it is prudent to implement full SCSI layer multipathing with ALUA support right out of the box.

ALEXANDER MOTIN is Team Lead of the OS/Services team at iXsystems Inc., and has been a FreeBSD source committer since 2007.