IBM AIX MPIO: Best practices and considerations

Article Number: 87 | Rating: Unrated | Last Updated: Sun, Nov 22, 2020 12:14 AM

IBM AIX MPIO: Best practices and considerations

Shannon Moore and Gary Domrow
Published on January 20, 2014

15

IBM Power Systems™ servers are designed to offer very high stand-alone availability in the industry. Enterprises must occasionally restructure their infrastructure to meet new IT requirements and handle scheduled outages (such as scheduled system maintenance).

MPIO best practices have never been officially documented. There have been some documents and IBM Redbooks® that have briefly mentioned certain MPIO aspects for specific scenarios and environments, but recommendations pertaining to MPIO configurations, in general, have been lacking.

System reliability and availability are increased by a careful consideration of the user-modifiable options in each system configuration. This article outlines the best practice configuration considerations that pertain to MPIO on AIX.

Some of the features described in this article are specific to particular technology levels of AIX, or are specific to the path control module (PCM) supplied with AIX. If using *Subsystem Device Driver Path Control Module* (SDDPCM) or a vendor-supplied Object Data Manager (ODM) package (often referred to as a *host attachment kit*, or something similar), then some of these options might be unavailable, and other options can be added.

The AIX MPIO infrastructure allows IBM or third-party storage vendors to supply ODM definitions, which have unique default values for the important disk attributes. Thus, for example, the default value for attributes on an hdisk representing a logical unit number (LUN) from an IBM System Storage® SAN Volume Controller (SVC) might be different from the default values for an hdisk representing a LUN on an IBM System Storage DS8000® system. As a result, the default values for the attributes are appropriate for most situations. Generally, the hdisk attributes should be left at their default values, especially the attributes that are not mentioned in this article.

The disk attributes described in the following sections can be displayed by using the lsattr command, and can be changed with the chdev command. The path attributes, such as path_priority, can be displayed or set by using the lspath and chpath commands. Refer to the AIX publications or AIX man pages for details on those commands.

This article does not address the attributes associated with the adapters being used to attach to the MPIO devices. Some of those attributes might also affect error detection and recovery times. In particular, the fc_err_recov attribute for Fiber Channel adapters is an important one to consider.

Consideration 1: MPIO algorithm and path_priority

The MPIO algorithm setting determines whether:

- The PCM can attempt to distribute I/O across all available paths to a given LUN
- The I/O will be active only on one path at a time
- The I/O flow will be weighted based on a combination of the algorithm setting and the path_priority settings per disk

A device that has multiple controllers can designate one controller as the active or preferred controller. For such a device, the PCM uses just the paths to the active or preferred controller as long as there is at least one such path that is enabled and not failed. Thus, algorithms that use all available paths might only use a subset of those paths at one time for such devices.

```
algorithm = fail_over
```

This is the default algorithm for most disks using the ODM definitions included with AIX. Some third-party ODMs use a different default value.

With this algorithm, I/O can only be routed down one path at a time. With algorithm=fail_over, the PCM keeps track of all the enabled paths (per disk) in an ordered list. If the path being used to send I/O fails or is disabled, the next enabled path in the list is selected and I/O is routed to that path. The sequence for path selection within the list is customizable by modifying the path priority attribute on each path, which will then sort the list by the ascending path priority value.

The fail_over algorithm is always used for virtual SCSI (VSCSI) disks on a Virtual I/O Server (VIOS) client, although the backing devices on the VIOS instance might still use round_robin, if required.

Fail_over is also the only algorithm that might be used if using SCSI-2 reserves (reserve_policy=single_path).

```
1
                                          algorithm = round_robin
```

With this algorithm, I/O will be distributed and activated across all enabled paths to a disk. The percentage of I/O routed down each path can be weighted by setting the path_priority attribute on each path for each disk. If a path fails or is disabled, it is no longer used for sending I/O. The priority of the remaining paths is then recalculated to determine the percentage of I/O that should be sent down each path. If all paths have the same path_priority value, the PCM ***attempts*** to equally distribute I/O across all enabled paths. Optimal performance in a failed path scenario is to ensure that the ordered path list alternate paths between separate fabrics.

```
1
                                          algorithm = shortest_queue
```

This algorithm is available in the latest technology levels of AIX for some devices. The algorithm behaves very similar to round_robin when the load is light. When the load increases, this algorithm favors the path that has the fewest active I/O operations. Thus, if one path is slow due to congestion in the storage area network (SAN), the other less-congested paths are used for more of the I/O operations. ***The path priority values are ignored by this algorithm.***

Recommendation: If using SCSI-2 reserves or vSCSI disks, then fail_over must be used. For other situations, shortest_queue (if available) or round_robin enable maximum use of the SAN resources.

Consideration 2: Path health check settings

Path health check mode (hcheck_mode)

The path health check mode determines the paths that the MPIO's path health checker will probe for path availability during normal business operations. The health checker never probes paths that are in a **Disabled** or **Missing** state. Paths in those two states must be recovered manually with chpath (for Disabled paths) or with cfgmgr (for Missing paths). *If a disk is not open and in use as is the case, for instance, when its volume group is varied off, no path health checks will take place down any path for that disk.*

There are three possible modes for the MPIO path health checker.

hcheck_mode = nonactive: In this mode, the PCM sends health check commands down paths which have no active I/O. That includes paths with a state of *failed*. If the algorithm selected is fail_over, then the health check command is also sent on each of the paths that have a state of enabled but have no active I/O. If the algorithm selected is round_robin or shortest_queue, then the health check command is only sent on paths with a state of *failed*, because the round_robin and shortest_queue algorithms both keep all enabled paths active with I/O when the disk is in use. If the disk is idle, the health check command is sent on any paths that do not have a pending I/O at the expiration of the health check interval.

hcheck_mode = enabled: In this mode, the PCM sends health check commands down all enabled paths, even paths that have other active I/O at the time of the health check.

hcheck_mode = failed: In this mode, the PCM only sends path health checks down paths that are marked as *failed*.

Recommendation: The default value for all devices is nonactive, and there is little reason to change this value unless business or application requirements dictate otherwise.

Path health check interval (hheck_interval)

The path health check interval is the interval, in seconds, at which MPIO path health checks will probe and check path availability of open disks, based on the hcheck_mode setting.

A hcheck_interval = 0 setting disables MPIO's path health check mechanism, which means any failed paths require manual intervention to recover or re-enable.

Recommendation: The best practice guideline for hcheck_interval is that it should be greater than or equal to the rw_timeout (read/write timeout) value on the disks. Also note that it is **not** a good idea to lower the rw_timeout value in order to set a lower health check interval. The default rw_timeout values set in ODM are based on the recommendations of the device manufacturers for each device type. The following section provides technical details regarding this best practice recommendation.

It might be tempting to think that a smaller health check interval is preferable as it might lead to faster detection or recovery of failed paths. However, the cost of setting a lower health check interval far outweighs the benefits. There are several reasons for this.

- Because the health check commands can be sent on every path of every open disk (depending on hcheck_mode) at the expiration of the health check interval, a small health check interval can quickly use up a lot of bandwidth on the SAN if there are a large number of disks.

- The health check commands count against the disk's queue_depth (only to be changed upon recommendation from the storage vendor), and they receive a higher priority for processing than normal user I/O. Because error scenarios typically take longer than *good path* scenarios, a small health check interval can negatively impact the user I/O on good paths when there are one or more failing paths. Note that because queue_depth is a function of the disk driver, queue_depth is on a per-LUN basis rather than a per-path basis. For example, assume that a device has a queue_depth of 8, with eight paths. If four of those paths have failed, the health check commands on those paths might take anywhere from a few seconds up to rw_timeout to fail. During that time, at least four of the eight commands in the queue_depth will be consumed by the health check commands, leaving an effective queue_depth of only four commands for the good paths and regular I/O for that disk.

- It is not always desirable to recover a path quickly. In a situation where a link is suffering from repeated, intermittent failures, the more quickly the link is recovered by a health check command, the more likely it is that a user I/O will be sent on that link only to fail due to the intermittent errors. A longer health check interval reduces the use of links with frequent but intermittent failures.

- AIX implements an emergency *last gasp* health check to recover paths when needed. If a device has only one non-failed path and an error is detected on that last path, AIX sends a health check command on all of the other failed paths before retrying the I/O, regardless of the health check interval setting. This eliminates the need for a small health check interval to recover paths quickly. If there is at least one good path, AIX discovers it and uses it before failing user I/O, regardless of the health check interval setting.

Recent technology levels of AIX also make use of asynchronous events from the Fibre Channel (FC) device drivers to manipulate path states. This makes AIX less dependent on the health check commands to detect path failures or to recover paths when using Fibre Channel.

For most cases, the default value of hcheck_interval is appropriate. There have been some storage vendors who, in older versions of their ODM definitions, had set hcheck_interval to a value smaller than the rw_timeout value. The previous recommendation from AIX development stands in those cases: Increase hcheck_interval such that it is greater than or equal to rw_timeout value. It is much more likely to be a good idea to increase the health check interval than to decrease it. Better performance is achieved when hcheck_interval is slightly greater than the rw_timeout value on the disks.

Extreme cases of the problems described in bullets 2 and 3 above can cause severe degradation of I/O performance if the health check interval is set to a small value.

Consideration 3: Time out policy

Recent technology levels of AIX include a timeout_policy attribute for some devices. This attribute indicates the action that the PCM should take when a command timeout occurs. A command timeout occurs when an I/O operation fails to complete within the rw_timeout value on the disk. There are three possible values for timeout_policy.

timeout_policy = retry_path: This represents the legacy behavior, where a command may be retried on the same path that just experienced a command timeout. This is likely to lead to delays in the I/O recovery, as it is likely that the command will continue to fail on this path. Only after several consecutive failures, will AIX fail the path and try the I/O on an alternate path.

timeout_policy = fail_path: This setting causes AIX to fail the path after a single command timeout, assuming that the device has at least one other path that is not in the *failed* state. Failing the path forces the I/O to be retried on a different path. This can lead to much quicker recovery from a command time out and also much quicker detection of situations where all paths to a device have failed. A path that is failed due to timeout policy can later be recovered by the AIX health check commands. However, AIX avoids using the path for user I/O for a period of time after it recovers to help ensure that the path is not experiencing repeated failures. (Other PCMs might not implement this grace period.)

timeout_policy = disable_path: This setting causes the path to be disabled. A disabled path is only recovered by manual user intervention using the chpath command to re-enable the path.

Recommendation: If this attribute is available on the device, a value of fail_path is the recommended setting.

Consideration 4: How many paths to configure for AIX MPIO

In an MPIO configuration, *more* is not necessarily *better*. In fact, an excessive number of paths in an MPIO configuration can actually contribute to system and application performance degradation in the event of SAN, storage, or Fibre Channel fabric issues or failures.

The general recommendation for the number of paths to configure in an MPIO environment is 4 to 8 per LUN, with 16 paths being recommended as the maximum, to be used only in specialized situations. *It is important to note that MPIO does support many more paths than 8 or 16, but from a design and functional perspective, four to eight paths have been proven to be the most effective*.

Businesses that need to configure more than eight paths per LUN need to carefully consider the following details:

1. When an error is encountered on an MPIO disk, error recovery normally takes place down all configured paths. The most common types of disk or SAN errors that occur will also lead to multiple retry attempts ***on each path for each failed I/O.*** With "N" paths, there could easily be a situation where a disk encounters an error that would lead to five tries on each path, multiplied by the rw_timeout value on the disks. So, total recovery ***per I/O*** could potentially be:

    ```
    1
    ```

(N * rw_timeout value * 5)

If multiple disks were to encounter similar issues at the same time, the consequences for applications might be severe. For example, a marginal, constantly bouncing link in the SAN fabric might lead to this type of error recovery, resulting in extreme performance degradation.

This situation is somewhat ameliorated by setting the timeout_policy attribute to fail_path, if that attribute is available with the device type that is being used. However, the timeout policy attribute cannot account for all possible error scenarios.

2. With the round_robin algorithm, having too many paths results in overhead as the PCM attempts to load balance I/O among the many paths.
3. With the fail_over algorithm, the PCM encounters additional overhead in determining the paths to use for failover in a failed path scenario.
4. Each configured path requires additional memory in AIX, as each path is represented by data structures in the MPIO device drivers. Having too many paths to a large number of disks can reduce the amount of memory available to the rest of the system for running applications.
5. As noted above, the health check commands count against the queue depth for the device. So, health check processing has a greater effect on devices with a large number of paths, especially with devices that have smaller queue depths, and especially when there are paths in the *failed* state.

The optimal configuration for a device having four paths on AIX is to use four physical paths to the storage subsystem with a 1:1 relationship between the host-side host bus adapter (HBA) port and the remote storage ports. If using multiport adapters on the AIX host, split at least half the paths among separate physical adapters for optimum redundancy. The AIX and device ports can be connected to the same FC switch or to two different switches in the same fabric. If using two switches, there is no single point of failure. However, certain switch or port failures might affect an entire SAN, thus impacting all four paths.

One possible eight-path configuration that provides full redundancy uses two distinct SAN fabrics. The AIX node and the storage device each have two ports connected to each of the two SAN fabrics, using a total of four ports on AIX and four ports on the storage device. There are four paths between AIX and the storage device for each of the two distinct SAN fabrics, for a total of eight paths. Thus, there is no single point of failure for either SAN fabric, and there are redundant SAN fabrics. (Note: This is just an example. It is completely possible to have full redundancy with four paths per LUN using dual fabrics.)

The only case for more than eight paths is for specialized storage devices that configure a cluster of controllers, or for devices using Peer-to-Peer Remote Copy (PPRC). For example, an hdisk representing an IBM HyperSwap® pair of LUNs on two DS8000 devices could have 16 paths if each of the DS8000 systems used to form the HyperSwap pair are configured in the 8-path configuration described above. After the two 8-path hdisks are configured as a single HyperSwap enabled hdisk, it will have 16 paths.

There are other possible configurations beyond what is described here that can be considered. However, as noted above, *going beyond eight paths can be more problematic than helpful, and should be carefully considered.*

Recommendation: Configure 4 or 8 paths per disk, or up to 16 paths for rare situations. Carefully consider the impacts of extra, unnecessary redundancy before using more paths.

Consideration 5: Operational considerations

Scheduled maintenance: AIX MPIO is capable of robust error detection and recovery. However this error detection and recovery might take some time, and that delay might impact applications. If scheduled maintenance is planned for a SAN or for a storage device, it is best to identify the disk paths that will be impacted by that maintenance and use the rmpath command to manually disable those paths before starting the maintenance. AIX MPIO stops using any disabled or **Defined** paths, and therefore, no error detection or recovery will be done as a result of the scheduled maintenance. This ensures that the AIX host does not go into extended error recovery for a scheduled maintenance activity. After the maintenance is complete, the paths can be re-enabled with cfgmgr. (Note: When disabling multiple paths for multiple LUNs, rmpath is simpler than chpath, as it does not have to be run on a per-disk basis.)

The lspath command (or in newer technology levels, the lsmpio command) can be used to determine the MPIO paths that are associated with specific SAN ports.

Changing attributes: For most attributes and most levels of AIX, attributes could historically only be changed on devices that were not in use. For disks, this meant that the disk must be closed (for example, volume group varied off) in order to change attributes. If the disk could not be closed, such as the disks containing rootvg, then the user had to include the -P flag in the chdev command to write the attribute change to ODM and then restart AIX in order for the attribute to take effect.

For the newest technology levels of AIX (at the time of publishing this article), some disk attributes on some devices support the -U flag on the chdev command. This flag instructs chdev to attempt a dynamic update of the attribute value. With this flag, the attribute value can be changed without closing the disk and the change takes effect immediately.

```
TST ---hdisk1---
algorithm load_balance
hcheck_interval 60
hcheck_mode nonactive

PROD ---hdisk1---
algorithm round_robin
hcheck_interval 60
hcheck_mode nonactive
```

```
for x in `lspath |awk '{print $2}' |sort -k1.6n |uniq`; do status=`lsattr -El ${x} | awk '{print $1" "$2}' | egrep "algorithm|hcheck_interval|hcheck_mode|rw_timeout|timeout_policy"`; echo "---${x}---"; echo "${status}"; echo " "; sleep 1 ; done
```

```
for x in `lspath |awk '{print $2}' |sort -k1.6n |uniq`; do status=`lsattr -El ${x}`; echo "---${x}---"; echo "${status}"; echo " "; sleep 1 ; done
```

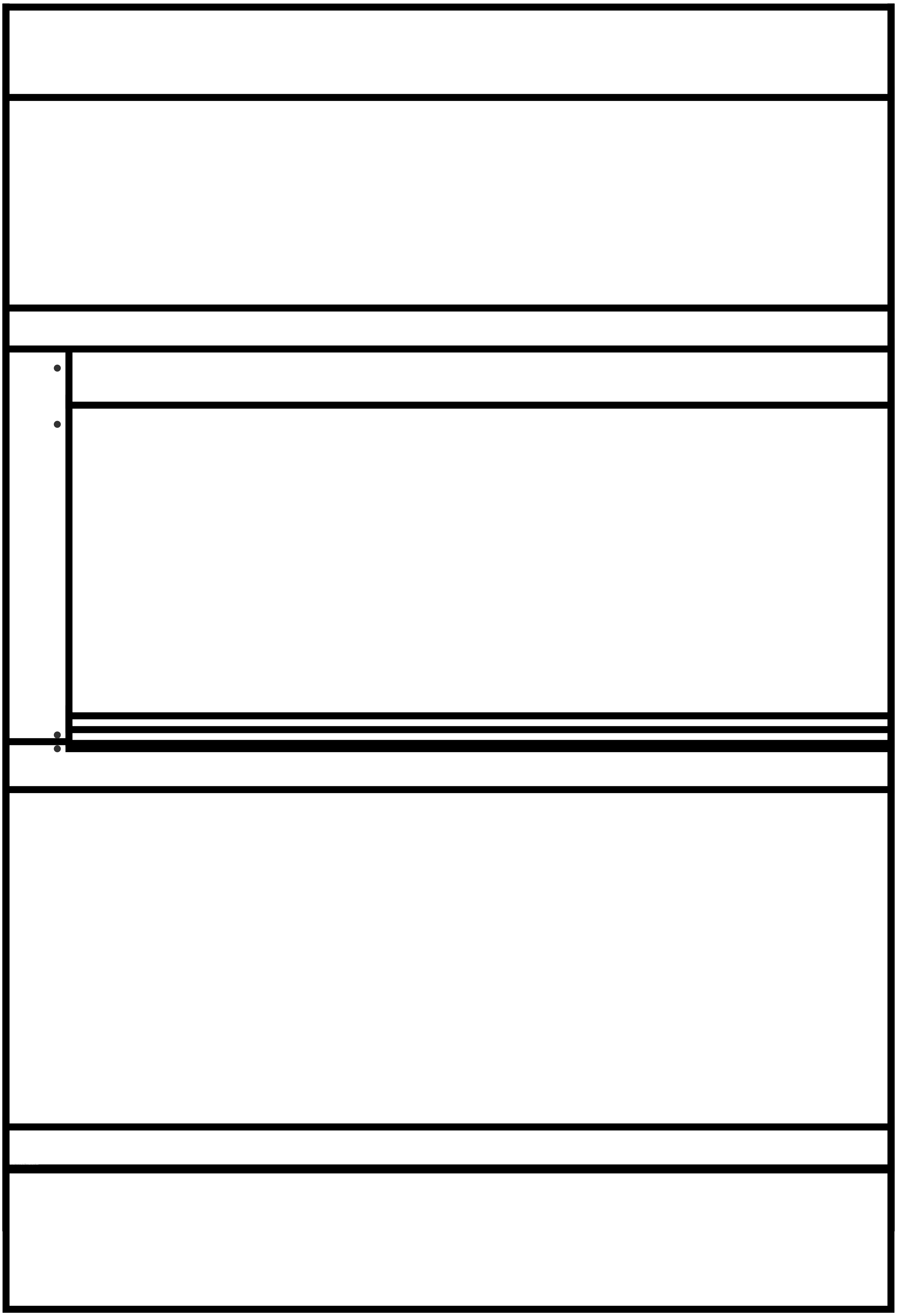
New version from Shannon Moore

This document describes best practices and considerations for an optimal MPIO configuration for AIX.

[illegible]

[illegible]

●
●
●



1.

2.

4.

5.

6.

Posted - Wed, May 30, 2018 10:47 AM. This article has been viewed 11309 times.

Online URL: http://kb.ictbanking.net/article.php?id=87