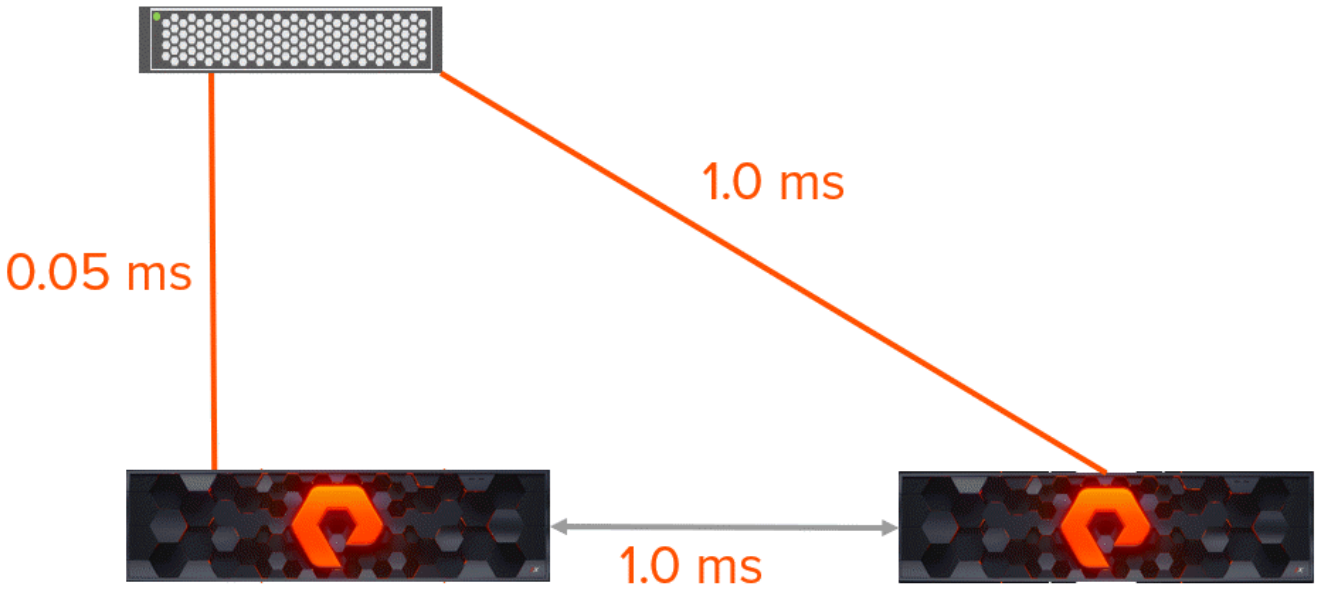
Latency-based PSP in ESXi 6.7 Update 1: A test drive



Yesterday, I wrote a post introducing the new latency-based round robin multipathing policy in ESXi 6.7 Update 1. You can check that out here:

Latency Round Robin PSP in ESXi 6.7 Update 1

In normal scenarios, you may not see much of a performance difference between the standard IOPS switching-based policy and the latency one. So don't necessarily expect that switching policies will change anything. But then again, multipathing primarily exists not for healthy states, but instead exists to protect during times of poor health.

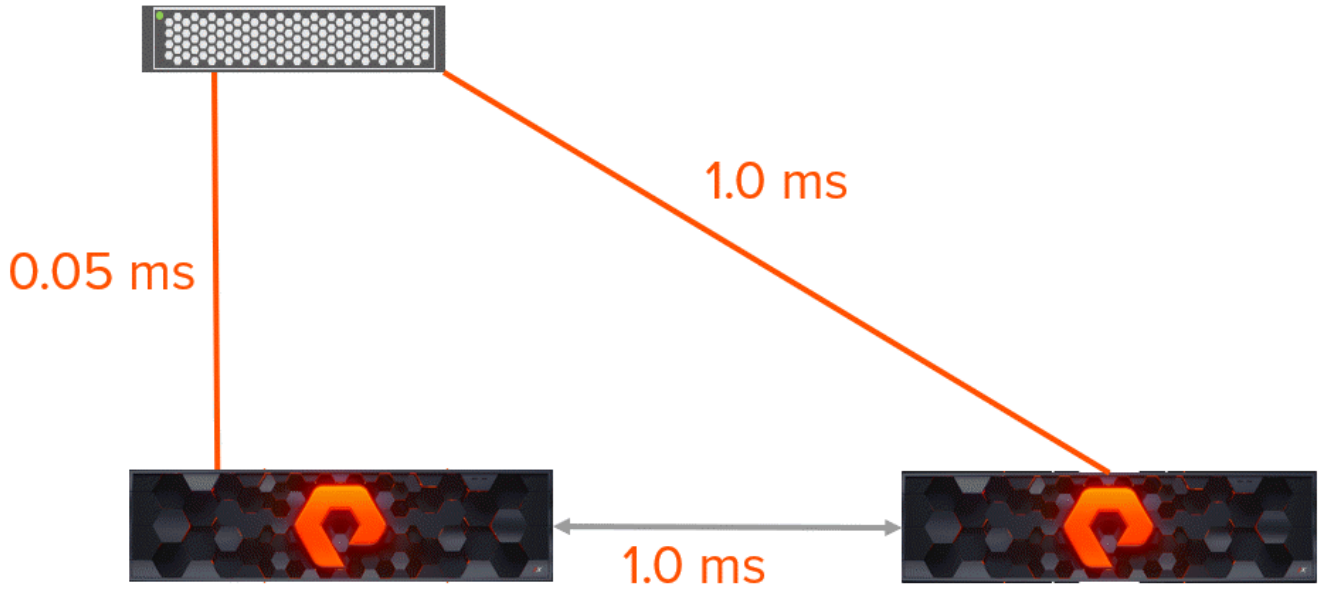
In the case of the latency policy, it works to route I/Os for a given device to the paths that exhibit the best performance (which is weighted for I/O size). Paths that are exhibiting higher latency are excluded until the next sampling period proves otherwise.

High latency of course doesn't necessarily mean a problem though–it could simply mean the end result of the speed of light–the I/O has to travel some distance. For most SANs this number is almost inconsequential, but in the case of active/active replication–this comes into play.

Let's take the following example. I have a host connected to two arrays. That host is accessing a volume that is protected by active-active replication by the two arrays. In other words, that volume exists on both arrays and can be written to and read from at the same time from either array. The host is in the same datacenter as one of the arrays, so the distance is small and the latency between that host and that array is only .05 ms one way.

The second array is in a different datacenter than the host and the other array. So the distance is greater

between the remote array and the host, and therefore the latency between them is also higher–1 ms one way.



This host can write to either array for its storage. But the paths to the local array are a better option than the paths to the remote array. Why? Well because in active-active replication (also standard synchronous for that matter) writes have to be acknowledged on both of the arrays before the host can be told the data is committed–this is what allows both arrays to serve the data at once.

Because of the geographic disparity, not all paths are equal. Depending on which array the host writes to the latency can differ greatly. Let's first take the case that the host writes to the local array:

1. Host sends a write buffer request to local array. COST: .05 ms.
2. The array opens the buffer request and responds back to the host. COST: .05 ms
3. The host sends the write to the array. COST: .05 ms
4. The array forwards the write to the remote array. COST: 1 ms.
5. The remote array stores and acknowledges the write back to the local array. COST: 1 ms.
6. The local array acknowledges back to the host that the write is complete. COST: .05 ms

.05 + .05 + .05 + 1 + 1 + .05 = 2.2 ms.

This is called an optimized write.

If the host though, instead decides to use a path to the remote array, the total time will be much longer.

1. Host sends a write buffer request to remote array. COST: 1 ms.
2. The remote array opens the buffer request and responds back to the host. COST: 1 ms
3. The host sends the write to the remote array. COST: 1 ms
4. The remote array forwards the write to the local array. COST: 1 ms.
5. The local array stores and acknowledges the write back to the remote array. COST: 1 ms.
6. The remote array acknowledges back to the host that the write is complete. COST: 1 ms.

1 + 1 + 1 +1 +1 +1 = 6 ms.

2.2 ms compared to 6 ms.

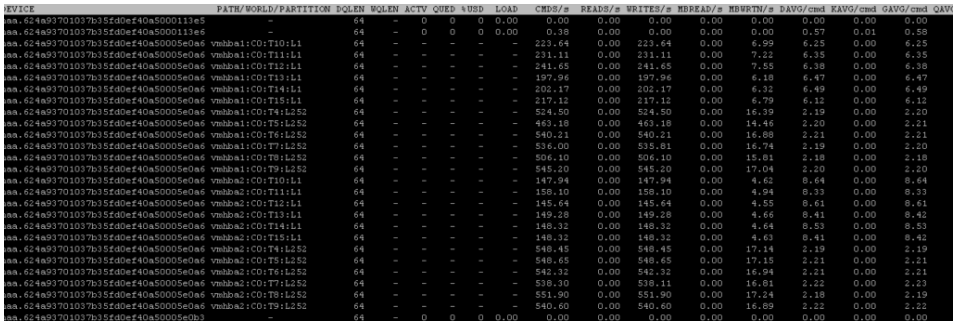
Even though they achieve the same thing: a protected write.

This is called a non-optimized write.

You can understand how customers who use this might be interested in the latency policy. Avoiding non-optimized writes important when the distances are significant–and having some intelligence to automatically figure this out is even better.

Side note: Now reads have less penalty than writes because they don't need to go to both arrays and there is also slightly less back and forth between the host and array so the write is the worse case scenario. Also this is SCSI–the write process is a bit more efficient with NVMe-oF, but regardless the penalty will exist at some level on non-optimized I/Os over distance.

I have a workload running that is configured to tell the VM to push as much as it can at 32 outstanding I/Os of 100% 32 KB writes to a replicated volume. With regular IOPS switching round robin all paths will be used equally, both optimized and non-optimized:



DEVICE	PATH/WORLD/PARTITION	DQLEN	WQLEN	ACTV	QUED	%USD	LOAD	CMDS/s	READS/s	WRITES/s	MBREAD/s	MBWRTN/s	DAVG/cmd	KAVG/cmd	GAVG/cmd	QAVG/cmd
naa.624a93701037b35fd0ef40a5000113e5	-	64	-	0	0	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-
naa.624a93701037b35fd0ef40a5000113e6	-	64	-	0	0	0	0.00	0.38	0.00	0.00	0.00	0.00	0.57	0.01	0.58	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T10:L1	64	-	-	-	-	-	223.64	0.00	223.64	0.00	6.99	6.25	0.00	6.25	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T11:L1	64	-	-	-	-	-	231.11	0.00	231.11	0.00	7.22	6.35	0.00	6.35	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T12:L1	64	-	-	-	-	-	241.65	0.00	241.65	0.00	7.55	6.38	0.00	6.38	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T13:L1	64	-	-	-	-	-	197.96	0.00	197.96	0.00	6.18	6.47	0.00	6.47	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T14:L1	64	-	-	-	-	-	202.17	0.00	202.17	0.00	6.32	6.49	0.00	6.49	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T15:L1	64	-	-	-	-	-	217.12	0.00	217.12	0.00	6.79	6.12	0.00	6.12	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T4:L252	64	-	-	-	-	-	524.50	0.00	524.50	0.00	16.39	2.19	0.00	2.20	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T5:L252	64	-	-	-	-	-	463.18	0.00	463.18	0.00	14.46	2.20	0.00	2.21	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T6:L252	64	-	-	-	-	-	540.21	0.00	540.21	0.00	16.88	2.21	0.00	2.21	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T7:L252	64	-	-	-	-	-	536.00	0.00	535.81	0.00	16.74	2.19	0.00	2.20	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T8:L252	64	-	-	-	-	-	506.10	0.00	506.10	0.00	15.81	2.18	0.00	2.18	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T9:L252	64	-	-	-	-	-	545.20	0.00	545.20	0.00	17.04	2.20	0.00	2.20	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T10:L1	64	-	-	-	-	-	147.94	0.00	147.94	0.00	4.62	8.64	0.00	8.64	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T11:L1	64	-	-	-	-	-	158.10	0.00	158.10	0.00	4.94	8.33	0.00	8.33	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T12:L1	64	-	-	-	-	-	145.64	0.00	145.64	0.00	4.55	8.61	0.00	8.61	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T13:L1	64	-	-	-	-	-	149.28	0.00	149.28	0.00	4.66	8.41	0.00	8.42	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T14:L1	64	-	-	-	-	-	148.32	0.00	148.32	0.00	4.64	8.53	0.00	8.53	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T15:L1	64	-	-	-	-	-	148.32	0.00	148.32	0.00	4.63	8.41	0.00	8.42	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T4:L252	64	-	-	-	-	-	548.45	0.00	548.45	0.00	17.14	2.19	0.00	2.19	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T5:L252	64	-	-	-	-	-	548.65	0.00	548.65	0.00	17.15	2.21	0.00	2.21	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T6:L252	64	-	-	-	-	-	542.32	0.00	542.32	0.00	16.94	2.21	0.00	2.21	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T7:L252	64	-	-	-	-	-	538.30	0.00	538.11	0.00	16.81	2.22	0.00	2.23	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T8:L252	64	-	-	-	-	-	551.90	0.00	551.90	0.00	17.24	2.18	0.00	2.19	-
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T9:L252	64	-	-	-	-	-	540.60	0.00	540.60	0.00	16.89	2.22	0.00	2.22	-
naa.624a93701037b35fd0ef40a50005e0b3	-	64	-	0	0	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-

As you can see half of the paths are at that 6 ms (or worse). The rest (the optimized paths) are right at ~2.20. With all of the paths (20 in total) the workload is pushing about 8,500 IOPS in total. Even though there is a big difference in the paths, they are being used equally.

This is not ideal.

Active-Optimized/ Non-Optimized Advertising

Prior to the latency round robin option there were other ways of avoiding non-optimized paths. What we (Pure) do on the FlashArray is use the concept of a preferred array. If a host has access to both arrays–we provide the option to configure on the arrays, which host is "local" to it. If the host is local to it, the paths from the host to that array are advertised back to the host as "active-optimized". The paths on the other array are advertised back as active-nonoptimized.

When I configure this, in the below screenshot you can see some paths are marked Active I/O (meaning they are being actively used for I/O) and other just Active (meaning they are available, but not being used).

Device: PURE Fibre Channel Disk (naa.624a93701037b35fd0ef40a50005e0a6)

Multipathing Policies

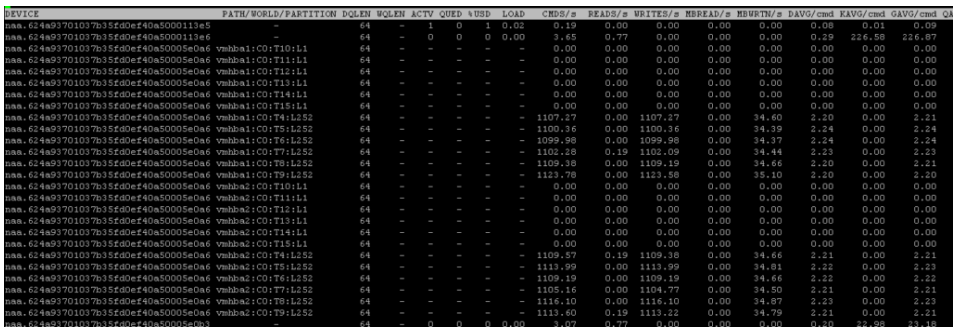
Path Selection Policy	Round Robin (VMware)
Storage Array Type Policy	VMW_SATP_ALUA
Owner Plugin	NMP

Paths

Refresh | Enable Disable

Runtime Name	Status	Target
vmhba1:C0:T5:L252	Active (I/O)	52:4a:93:75:15:74:75:11 52:4a:93:75:15:74:75:11
vmhba2:C0:T4:L252	Active (I/O)	52:4a:93:75:15:74:75:00 52:4a:93:75:15:74:75:00
vmhba1:C0:T12:L1	Active	52:4a:93:72:e3:85:21:03 52:4a:93:72:e3:85:21:03
vmhba2:C0:T11:L1	Active	52:4a:93:72:e3:85:21:10 52:4a:93:72:e3:85:21:10
vmhba1:C0:T8:L252	Active (I/O)	52:4a:93:75:15:74:75:07 52:4a:93:75:15:74:75:07
vmhba2:C0:T7:L252	Active (I/O)	52:4a:93:75:15:74:75:16 52:4a:93:75:15:74:75:16
vmhba1:C0:T11:L1	Active	52:4a:93:72:e3:85:21:01 52:4a:93:72:e3:85:21:01
vmhba2:C0:T10:L1	Active	52:4a:93:72:e3:85:21:00 52:4a:93:72:e3:85:21:00
vmhba2:C0:T15:L1	Active	52:4a:93:72:e3:85:21:16 52:4a:93:72:e3:85:21:16

Using traditional round robin, this was the only way to automatically have ESXi avoid using non-optimized paths. Non-optimized paths would only be used if ALL of the of optimized paths went away.



DEVICE	PATH/WORLD/PARTITION	DQLEN	WQLEN	ACTV	QUED	%USD	LOAD	CMDS/s	READS/s	WRITES/s	MBREAD/s	MBWRTN/s	DAVG/cmd	KAVG/cmd	GAVG/cmd	QA
naa.624a93701037b35fd0ef40a5000113e5	-	64	-	1	0	1	0.02	0.19	0.00	0.00	0.00	0.00	0.08	0.01	0.09	
naa.624a93701037b35fd0ef40a5000113e6	-	64	-	0	0	0	0.00	3.65	0.77	0.00	0.00	0.00	0.29	226.58	226.87	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T10:L1	64	-	-	-	-	-	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T11:L1	64	-	-	-	-	-	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T12:L1	64	-	-	-	-	-	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T13:L1	64	-	-	-	-	-	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T14:L1	64	-	-	-	-	-	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T15:L1	64	-	-	-	-	-	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T4:L252	64	-	-	-	-	-	1107.27	0.00	1107.27	0.00	34.60	2.20	0.00	2.21	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T5:L252	64	-	-	-	-	-	1100.36	0.00	1100.36	0.00	34.39	2.24	0.00	2.24	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T6:L252	64	-	-	-	-	-	1099.98	0.00	1099.98	0.00	34.37	2.24	0.00	2.24	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T7:L252	64	-	-	-	-	-	1102.28	0.19	1102.09	0.00	34.44	2.23	0.00	2.23	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T8:L252	64	-	-	-	-	-	1109.38	0.00	1109.19	0.00	34.66	2.20	0.00	2.21	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba1:C0:T9:L252	64	-	-	-	-	-	1123.78	0.00	1123.58	0.00	35.10	2.20	0.00	2.20	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T10:L1	64	-	-	-	-	-	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T11:L1	64	-	-	-	-	-	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T12:L1	64	-	-	-	-	-	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T13:L1	64	-	-	-	-	-	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T14:L1	64	-	-	-	-	-	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T15:L1	64	-	-	-	-	-	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T4:L252	64	-	-	-	-	-	1109.57	0.19	1109.38	0.00	34.66	2.21	0.00	2.21	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T5:L252	64	-	-	-	-	-	1113.99	0.00	1113.99	0.00	34.81	2.22	0.00	2.23	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T6:L252	64	-	-	-	-	-	1109.19	0.00	1109.19	0.00	34.66	2.22	0.00	2.22	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T7:L252	64	-	-	-	-	-	1105.16	0.00	1104.77	0.00	34.50	2.21	0.00	2.21	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T8:L252	64	-	-	-	-	-	1116.10	0.00	1116.10	0.00	34.87	2.23	0.00	2.23	
naa.624a93701037b35fd0ef40a50005e0a6	vmhba2:C0:T9:L252	64	-	-	-	-	-	1113.60	0.19	1113.22	0.00	34.79	2.21	0.00	2.21	
naa.624a93701037b35fd0ef40a50005e0b1	-	64	-	0	0	0	0.00	3.07	0.77	0.00	0.00	0.00	0.20	21.38	21.58	

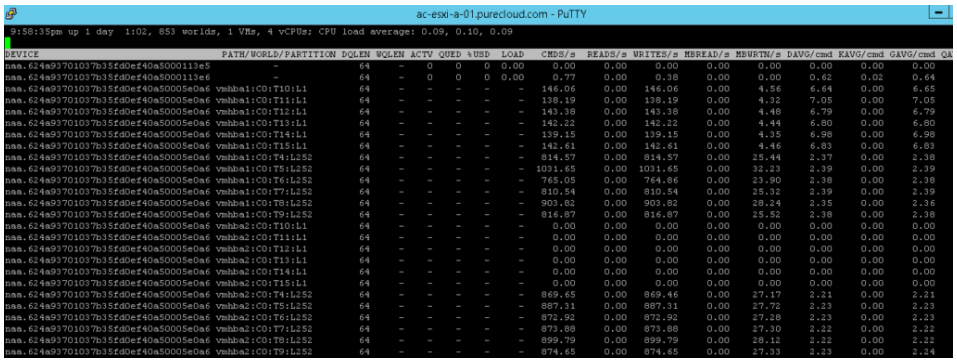
The workload increases to 13,500 IOPS–by simply excluding certain paths. Because ESXi no longer has to wait for half of the I/Os for twice the amount of time, it can push more I/Os in the same amount of time.

Latency-based Round Robin

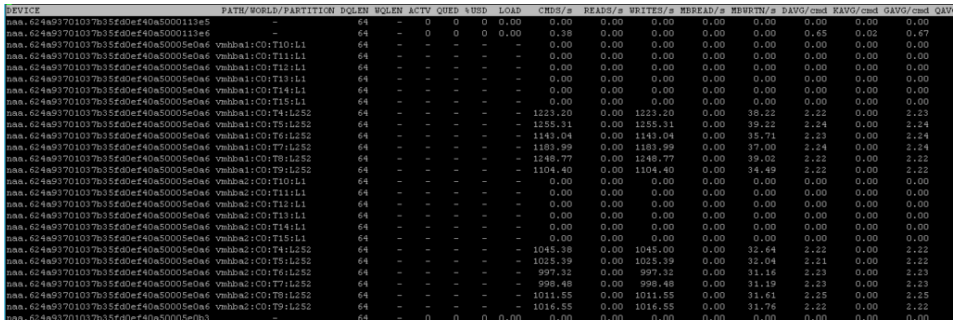
Now with 6.7 Update 1, you could instead opt to use the new latency-based policy. Using the same environment as above, I will disable my array from advertising optimized/non-optimized so that ESXi sees all paths the same. I will then set my datastore on that host to use the latency-based round robin policy.
[crayon-64277dd73b79e095107698/]
If you remember from above some of my non-optimized paths were around 6 ms and others were a bit above 8 ms. After some troubleshooting it seemed my latency simulator was slowing down some of the

paths more than the others, but I decided to keep it that way because it helps make my point here.

The worst paths (8ms+) were dropped pretty much immediately:



And the remaining non-optimized paths were still being used for a bit eventually (few minutes) those were dropped too in favor of only optimized paths.



The IOPS was back at 13,500 again.

Q&A

Before I get into it let me re-iterate this: as I said in my previous post, I am not going to 100% recommend the latency policy until we (Pure) can complete full testing internally on it. But for the purposes of this Q&A let's put that aside for a moment and let's talk in theory about the results I have seen and pretend we have finished testing.

As we do more testing, I will update these and possibly include links to more posts. (last updated 10/18/2018)

Which option should I use?

Good question. If you're replicating FlashArrays are pretty close where the latency is not that different, I would suggest not using the preferred array settings on the FlashArrays and letting the latency policy decide. This is what we recommended before the latency policy existed anyways. Preferred array settings on the FlashArray is really targeted for use at geographically separated arrays. If they are separated, you now have a choice. If the latency difference is quite high, I would still set the preferred array to ensure those paths are only used in times of need (all of the active go down). If your difference in latency is not as big (or you don't care about the difference) then just use the latency policy. If you want to reduce traffic in general across the WAN–definitely use the preferred array and latency policy. This allows the preferred array setting to have VMware only use local paths if they are available, and the latency policy to discard any local paths that happen to be misbehaving.

Is there a time I shouldn't use the latency policy?

Well, if you haven't tested it yet (or your vendor doesn't recommend it) then don't. Until I complete further testing I don't know the full extent of this answer.

What are the options for the latency policy?

There are three:

- **num-sampling-cycles**. Default: 16. Valid values are 16-2147483647
- **latency-eval-time**. Default: 180000. Valid values are 30000-2147483647
- **useANO**. Default 0. Valid values are 0 or 1. If set to 1, ESXi will ignore ALUA settings and still use active non-optimized paths if their latency is deemed good enough. This is basically like enabling the latency default policy without enabling the preferred array setting on the FlashArray.

Why 2147483647? Well I guess they gave a 32 bit value space for this and that is the highest signed binary integer 32 bit allows for. Fun fact: yes it's a prime number! More fun fact: yes it looks like a phone number

in Dallas. And yes I called it. And no, no one picked up

Should I change the default values?

At this point in time, I have not seen much value in changing the number of sample periods or the evaluation interval time. So as of now, keep the latency policy to the defaults.

What if I am not using active-active replication?

Yes I used that as an example, as it is probably the most common example of different latencies on different paths. But of course, you can use it for "normal" volumes. Situations can arise that could cause one path to be better than another–so yeah why not?!

Can it improve performance in normal circumstances?

Unknown. I have not found that to be the case yet, but much more testing is needed. More mixed, heavier workloads etc.

Will it work with RDMs? VMFS? VVols?

Yes, yes, and yes.

How do I see the current device config?

Run:
[crayon-64277dd73b7ab554284414/]
[crayon-64277dd73b7b2267100092/]