Blockchain Storage – Drive Configurations and Performance Analysis

Jesse Garner
Harrisburg University of Science and Technology, jgarner@my.harrisburgu.edu

Aditya A. Syal
Harrisburg University of Science and Technology, asyal1@harrisburgu.edu

Ronald C. Jones
Harrisburg University of Science and Technology, rcjones@harrisburgu.edu

Blockchain Storage – Drive Configurations and Performance Analysis

Jesse L. Garner, Aditya A. Syal, Ronald C. Jones

Department of Computer and Information Science: Harrisburg University of Science and Technology

Abstract: This project will analyze the results of trials implementing various storage methods on Geth nodes to synchronize and maintain a full-archive state of the Ethereum blockchain. The purpose of these trials is to gain deeper insight to the process of lowering cost and increasing efficiency of blockchain storage using available technologies, analyzing results of various storage drives under similar conditions. It provides performance analysis and describes performance of each trial in relation to the others.

I. Introduction

Blockchain technology introduces a decentralized approach to data verification and retrieval [1]. Growing amounts of data bring concerns regarding privacy and security on how data is stored and may be leaked [2]. Because of this, it is in the best interest of individuals and organizations that participate in such a network to invest in a machine which will function as a trustworthy peer on the network [2].

This project compares the performance of several trials which implement different tiers of storage and storage configurations for maintaining a full archive blockchain state, using a drive to performance analysis of results obtained. Standardization is ensured by running each trial as a Geth node within the Ethereum network. Additionally, each trial is performed on a Debian based distributions on Linux.

Harrisburg University's High-Performance Computing (HPC) Research Lab (HPCRL) facilitated the trials for this project through funding and partnerships with organizations with mutual interest in performance results. HPCRL resources were utilized for the purposes of this research, including hardware, network connection, and monitoring through the use of Grafana – a metric visualization tool hosted on a VM within the HPC oVirt cluster.

The following sections will begin by providing the steps taken to allow for reproducible results. The actions taken will vary depending on the setup and versions of services available compared to those used at the time of this writing. Afterwards, a breakdown of performance with comparative analysis of the critical elements within each trial; the sections maintain a linear flow of the actions/analysis of each trial.

II. Overview of Trials

Storage methods in each trial vary by quality and configurations of drives – each organized though Logical Volume Manager (LVM) to form main storage [3]. These trials represent various tiers of storage quality for individuals looking into setting up their own Geth node to verify transactions on the blockchain. These incremental tiers include SSDs to serve as cache for HDDs, standard SSDs, Scaleflux compression NVMe drives, and standard NVMe drives. Additionally, a GCP cloud instance with SSDs is used as a separate comparison point.

In addition, each trial uses Go to run an Ethereum client (Geth), synchronizing the state of the Ethereum blockchain, becoming a Geth node in the Ethereum network [4]. Due to the use of LVM, logical drives mounted on each Geth node serve as the destination for all blockchain data, specified through the command given to launch Geth.

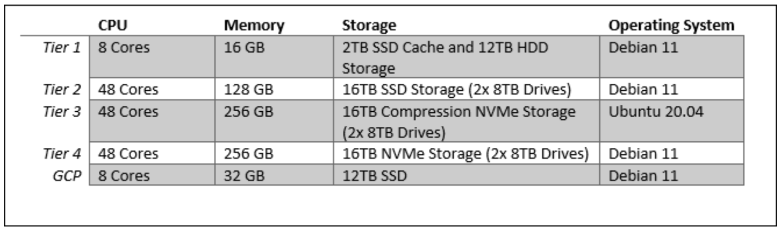
Metric collection is performed with Node Exporter and Prometheus running on each server, forwarding this data to a Grafana VM to analyze the performance results [5]. The collection of data is analyzed for several things – CPU activity, memory cache utilization, network transfer speeds, and the performance of drives trialed in terms of I/O operations. Additionally, the progress of each trial is recorded regularly to compare the speed over time in comparison with other trials.

The first trial employs a 2TB SSD as a writeback cache for a 12TB HDD used as main storage. An 8-core CPU and 16GB of RAM is used within a Dell XPS desktop running Debian 11, this replicates a personal machine closer than a server. Since a Geth node can be run on any machine, this trial simulates the commodity option for standing a Geth node, because of its price. This will uncover performance results of caching blockchain data to slow, affordable, more available storage.

Next, a Supermicro server running Debian 11 represents the next tier of trials which utilizes two SSDs combined through LVM, producing a logical volume of 16TB. It's equipped with a 48-core CPU and 128GB of RAM, this steps into the realm of an enterprise's budget and expectations for performance. It serves as a middle-ground for comparison between the former and upcoming trials.

The following trial features a Dell server with compression NVMe drives from ScaleFlux. This server runs Ubuntu 20.04 instead of Debian due to driver dependencies which are needed specific to the Ubuntu OS. In this case there are two 8TB compression NVMe drives organized as a single logical drive through the use of LVM. It is equipped with a 48-core CPU and 256GB of RAM. The compression drives allow for higher I/O operations since writes to the drive are able to be processed faster due to hardware compression. This trial serves to explore the capabilities of compression drives storing blockchain data in terms of read/write operations in comparison to other trials. Trial four is run on the same server, with a change to the operating system and storage drives used.

Table 1 shows a comparison of each trial as it relates to the resources each will be using.



	CPU	Memory	Storage	Operating System
Tier 1	8 Cores	16 GB	2TB SSD Cache and 12TB HDD Storage	Debian 11
Tier 2	48 Cores	128 GB	16TB SSD Storage (2x 8TB Drives)	Debian 11
Tier 3	48 Cores	256 GB	16TB Compression NVMe Storage (2x 8TB Drives)	Ubuntu 20.04
Tier 4	48 Cores	256 GB	16TB NVMe Storage (2x 8TB Drives)	Debian 11
GCP	8 Cores	32 GB	12TB SSD	Debian 11

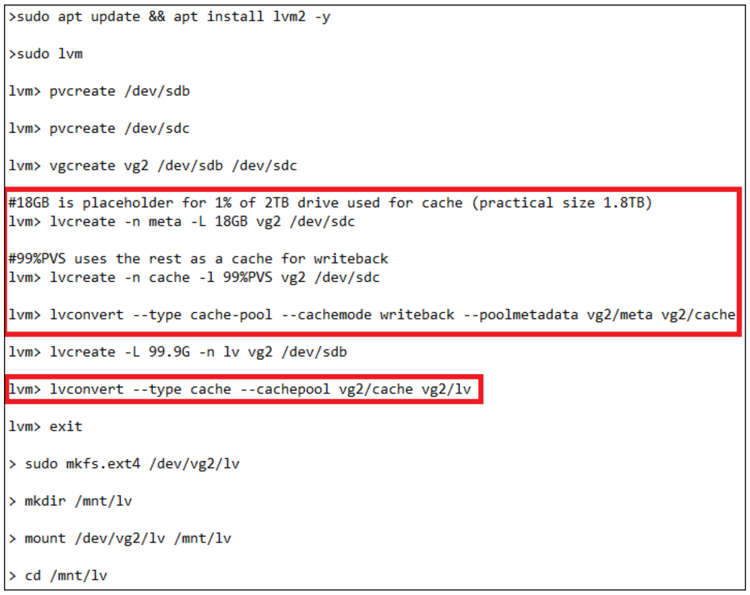
Table 1 – Trial attribute comparison

III. Combining Drives – LVM

The first step to begin each trial is combining drives together to form a single logical drive which is used to store the Geth data. This can be done on the main storage itself, but for this project the operating system was held on dedicated physical storage, separate from the drives used for testing of blockchain storage performance. The GCP trial, while useful for comparative analysis, is a cloud instance which did not follow the same steps of organizing storage through LVM; the remainder of this section will exclude the GCP trial from its scope.

Out of the four trials evaluated, three trials combine two drives of equal size and speed into one logical volume. The other uses a smaller SSD as a writeback cache for a larger HDD; this trial involves all of the steps to set LVM as the other trials, with the addition of several commands needed to initiate a caching relationship between the two drives. In **Figure 1** it identifies the steps taken to organize logical volumes on each trial, along with the additional steps needed for the trial caching storage.

On Debian, LVM must be installed manually. Issuing the *lsblk* command from the CLI exposes the characteristics of physical drives mounted to the server, this will indicate the path the drive is mounted i.e. - */dev/$drivename*. From within the LVM configuration console the physical volumes are created from physical storage drives currently mounted to Linux system, as the last command described. The physical volumes in LVM are combined to create a virtual group, this virtual group will be formed into a logical volume.

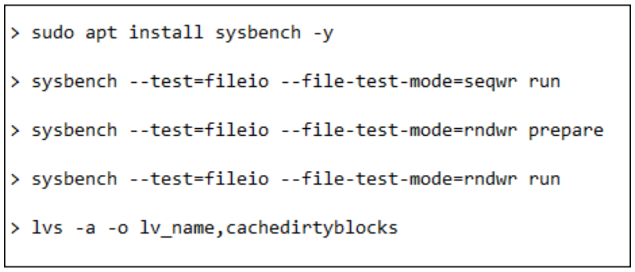


```
>sudo apt update && apt install lvm2 -y

>sudo lvm

lvm> pvcreate /dev/sdb

lvm> pvcreate /dev/sdc

lvm> vgcreate vg2 /dev/sdb /dev/sdc

#18GB is placeholder for 1% of 2TB drive used for cache (practical size 1.8TB)
lvm> lvcreate -n meta -L 18GB vg2 /dev/sdc

#99%PVS uses the rest as a cache for writeback
lvm> lvcreate -n cache -l 99%PVS vg2 /dev/sdc

lvm> lvconvert --type cache-pool --cachemode writeback --poolmetadata vg2/meta vg2/cache

lvm> lvcreate -L 99.9G -n lv vg2 /dev/sdb

lvm> lvconvert --type cache --cachepool vg2/cache vg2/lv

lvm> exit

> sudo mkfs.ext4 /dev/vg2/lv

> mkdir /mnt/lv

> mount /dev/vg2/lv /mnt/lv

> cd /mnt/lv
```

Figure 1 – Steps to setup LVM, additional steps for cache drive highlighted

If arranging caching through LVM, additional steps must be taken at this time before creating the logical volume from the virtual group. The physical volume which will function as the cache is given two partitions – metadata and cache. The metadata partition should be about one percent of the logical volume's practical capacity, the remainder is used for the purposes of caching. These two partitions are converted into a cache pool with a writeback policy, prioritizing high throughput of writes to cache. There is now a logical volume for the cache drive; another logical volume is created for other drive and then combined with the cache drive with the final command in caching section.

The work of LVM is now complete and the logical volumes are now visible from the */dev/* directory; they are a representation of the physical drives seen by LVM. A filesystem is created for the logical volume, and it is mounted within the */mnt/* directory. A few commands are left at the end of **Figure 2** to evaluate the function of a logical volume using cache with the *sysbench* tool.



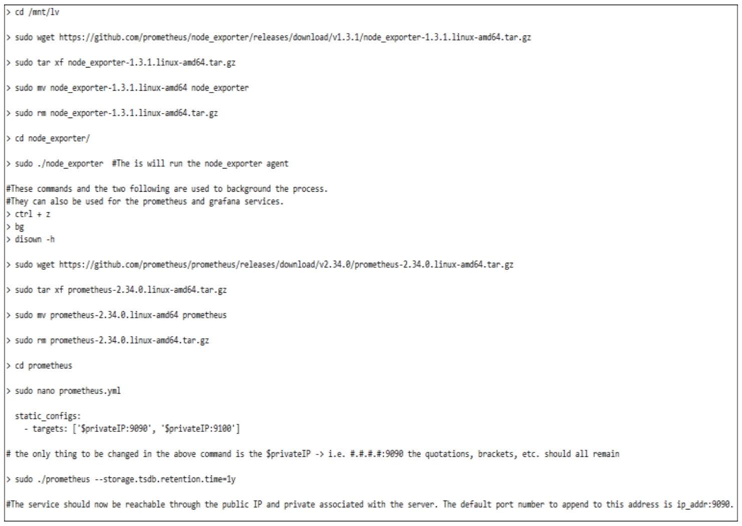
```
> sudo apt install sysbench -y

> sysbench --test=fileio --file-test-mode=seqwr run

> sysbench --test=fileio --file-test-mode=rndwr prepare

> sysbench --test=fileio --file-test-mode=rndwr run

> lvs -a -o lv_name,cachedirtyblocks
```

Figure 2 – Commands to install sysbench and test cache setup through LVM

IV. Preparing Monitoring Services

Each trial collects metrics of performance through the use of *Node Exporter* and *Prometheus* which are installed onto the local storage; these packages are found on https://prometheus.io/download/ [5]. *Prometheus* will then pass along those metrics to a data visualization tool – *Grafana*, which will connect to *Prometheus* through the host's private IP address and port number. **Figure 3** describes the steps taken to install the Prometheus service.

Node Exporter and *Prometheus* are installed within the same */mnt/lv* path to the logical volume which was created in the previous section. Consider the time period for data retention before running the command to start the *Prometheus* process; by default, it will delete metrics that are older than 15 days. This may be changed by appending a flag to the end of the command. For example, to run *Prometheus* with a data retention period of one year, issue the following command: *sudo ./prometheus --storage.tsdb.retention.time=1y*.



```
> cd /mnt/lv

> sudo wget https://github.com/prometheus/node_exporter/releases/download/v1.3.1/node_exporter-1.3.1.linux-amd64.tar.gz

> sudo tar xf node_exporter-1.3.1.linux-amd64.tar.gz

> sudo mv node_exporter-1.3.1.linux-amd64 node_exporter

> sudo rm node_exporter-1.3.1.linux-amd64.tar.gz

> cd node_exporter/

> sudo ./node_exporter  #The is will run the node_exporter agent

#These commands and the two following are used to background the process.
#They can also be used for the prometheus and grafana services.
> ctrl + z
> bg
> disown -h

> sudo wget https://github.com/prometheus/prometheus/releases/download/v2.34.0/prometheus-2.34.0.linux-amd64.tar.gz

> sudo tar xf prometheus-2.34.0.linux-amd64.tar.gz

> sudo mv prometheus-2.34.0.linux-amd64 prometheus

> sudo rm prometheus-2.34.0.linux-amd64.tar.gz

> cd prometheus

> sudo nano prometheus.yml

  static_configs:
    - targets: ['$privateIP:9090', '$privateIP:9100']

# the only thing to be changed in the above command is the $privateIP -> i.e. #.#.#.#:9090 the quotations, brackets, etc. should all remain

> sudo ./prometheus --storage.tsdb.retention.time=1y

#The service should now be reachable through the public IP and private associated with the server. The default port number to append to this address is ip_addr:9090.
```

Figure 3 – Commands to run Prometheus

Grafana is installed on a separate machine, in this case it is on a VM in the HPCRL's oVirt cluster, on the same private network. The steps to install *Grafana* can be followed from https://grafana.com/docs/grafana/latest/installation/c, which will vary depending on the operating system [6]. **Figure 4** shows that steps taken after setup to login to *Grafana* and add the *Prometheus* data source. Afterwards, the 'Node Exporter Full' dashboard is imported as the specified method of visualizing collected metrics.

```
https://grafana.com/docs/grafana/latest/installation/

Follow guide to install to Operating System in use

Login through $ip_addr:3000

Default login is admin:admin

# $privateIP is the IP of the server running Prometheus
Add a prometheus data source -> URL = http://$privateIP:9090

Create new dashboard, import 'Node Exporter Full' template by entering 1860
```

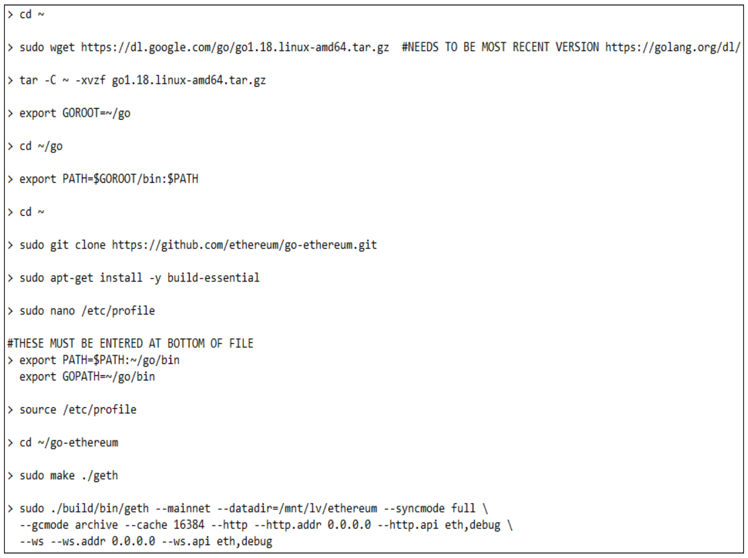
Figure 4 – Instructions to setup Grafana

For the purposes of these trials, only a handful of metrics collected will be the point of focus for comparative analysis. These include CPU use, memory utilization, disk performance (in terms of IOps and data collected in timeframe), and network traffic speed (packets per second). The 15-day mark of each trial is used as the point of comparison for overall speed of blockchain synchronization.

V. Installing Geth

Once metric collection and analysis services are setup and verified it is time to begin the installation of *Geth* (Go-Ethereum) which uses the Go language to initiate and maintain the server to function as an Ethereum peer. The steps of this process will vary depending on the version of Linux used but should work for Debian 11 and Ubuntu 20.04.

Figure 5 displays the steps taken to install *Geth* on each trial, with notes for commands that are relevant to the time of this writing, like the current version of *Go* for example. In this example, *Go* and *Go-Ethereum* are installed in the home directory of the server, but this can be changed to the preferred directory of installation so long as the exported variable paths are altered to match this change [7]-[8].



```
> cd ~

> sudo wget https://dl.google.com/go/go1.18.linux-amd64.tar.gz  #NEEDS TO BE MOST RECENT VERSION https://golang.org/dl/

> tar -C ~ -xvzf go1.18.linux-amd64.tar.gz

> export GOROOT=~/go

> cd ~/go

> export PATH=$GOROOT/bin:$PATH

> cd ~

> sudo git clone https://github.com/ethereum/go-ethereum.git

> sudo apt-get install -y build-essential

> sudo nano /etc/profile

#THESE MUST BE ENTERED AT BOTTOM OF FILE
> export PATH=$PATH:~/go/bin
  export GOPATH=~/go/bin

> source /etc/profile

> cd ~/go-ethereum

> sudo make ./geth

> sudo ./build/bin/geth --mainnet --datadir=/mnt/lv/ethereum --syncmode full \
  --gcmode archive --cache 16384 --http --http.addr 0.0.0.0 --http.api eth,debug \
  --ws --ws.addr 0.0.0.0 --ws.api eth,debug
```

Figure 5 – Commands to install Geth

To install *Go*, the most recent version is installed from the source specified in the figure. The *GOROOT* variable is set to the path that *Go* is installed to, as well as the *PATH* variable while in the *~/go* directory. Next, *Go-Ethereum* is downloaded from its GitHub repository and stored in the home directory. Once completed, the *build-essential* repository is installed, and path variables changed and confirmed in the */etc/profile/* directory.

Geth is prepared with the *make* command before finally starting the process. There are multiple flags issued with the command to start the Geth node, several of these are particularly important with respect to the testing of storage drives mounted to the server. Each trial also reserves 16GB of RAM to function as a cache for the main storage.

In this case the *datadir* flag indicates the path to the directory which will contain the blockchain data, it is set to the logical volume with a subdirectory named *Ethereum* inside, which has not been created before running the command. The *syncmode* and *gcmode* flags specify the server to act as a full archive node, downloading all blocks (headers, transactions, receipts) that are a part of the *mainnet* blockchain. The *cache* flag indicated the amount of MB to use as cache.

The Geth process is started with the last command seen in figure and is executed from the *~/go-ethereum/* directory. This will begin the synchronization of the blockchain state. To make this process run in the background, issue the following three commands: *Ctrl+Z*, *bg*, and *disown -h*. These are the same actions taken to background the *Node Exporter* and *Prometheus* processes.

VI. Synchronization Progress

The trials, while starting at different times, were measured by the total amount of blocks synchronized 15 days after the sync began. **Table 2** shows the amount of blocks each trial synchronized by the 15-day mark.

	Blocks Synchronized	Used Disk Space	Daily Average After 15-Day Mark
Trial 1 (12TB HDD + 2TB SSD Cache)	5,047,901 blocks	708 GB	27,657 blocks
Trial 2 (16TB SSD)	8,091,488 blocks	2.7 TB	98,215 blocks
Trial 3 (16TB Compression NVMe)	10,993,536	5.1 TB	124,803 blocks
Trial 4 (16TB NVMe)	10,550,640	4.6 TB	116,505 blocks
GCP	10,900,000*	4.8 TB*	120,000 blocks*

*Estimates based on GCP metrics

Table 2 – 15-day mark of trials

Figure 6 shows a comparison of the trials, with respect to the blocks pulled after the 15-day mark. The reason the status of each trial is shown after the 15-day mark is due to the change in block structure that occurs after the first few million blocks have been synchronized. This will ensure that the daily average from each trial is based on the same block structure. The dotted lines are projections of trial progress, due to the separate startup time of each trial.

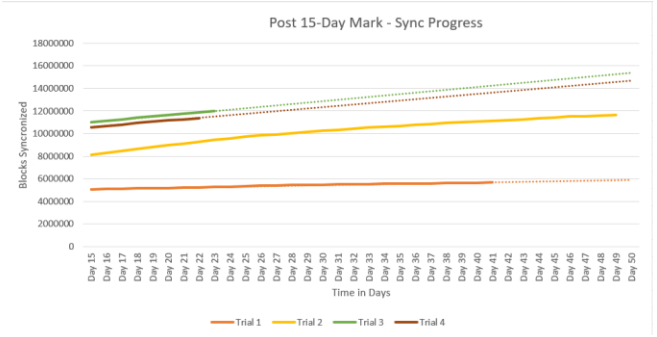


Figure 6 – Progress of trials after 15-day mark

There are more variables at play then just drive performance, however, since each server varies in its capabilities of processing and memory, mentioned in *Table 1* in the *Overview of Trials* section. Because of this there is some variance to be expected with relation to the results reflecting drive performance alone.

VII. Other Metrics of Interest

For the purposes of comparing particular metrics, a week-long sample after the 15-day mark will be used as a gauge between trials. There are two reasons for this, the first is that the progress of each trial is represented by the progress it makes within the first 15 days after beginning the sync, the second has to do with the structure of blocks later on in the chain – by day 15 in each trial the block structure had stabilized.

The variables across trials vary with respect to CPU power, size of memory, and storage drives, of course; this is important to keep in mind since the trials are not entirely standardized. The metrics measured are CPU utilization, Memory use and cache writeback, Disk performance (I/Ops and Read/Write Data), and network traffic (by packets).

CPU

Processing does not seem to be a critical component to the speed of blockchain synchronization. While the size of the CPU varied on these trials, even the trial with the weakest CPU did not have issues with keeping up. **Figure 7** shows the trial pairing a cache with storage and an 8-core CPU, which hovered under 50% CPU utilization. The other trials with stronger 48-core CPUs were stable at around 10% utilization, as shown in **Figure 8.**

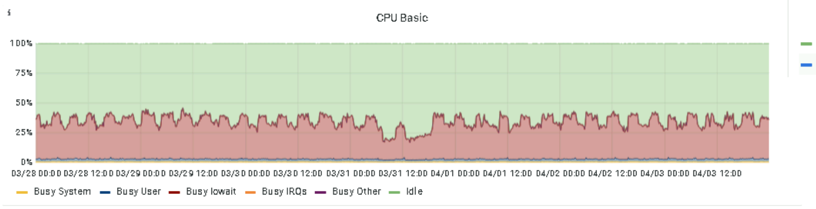


Figure 7 – 8 Core CPU below 50% utilization

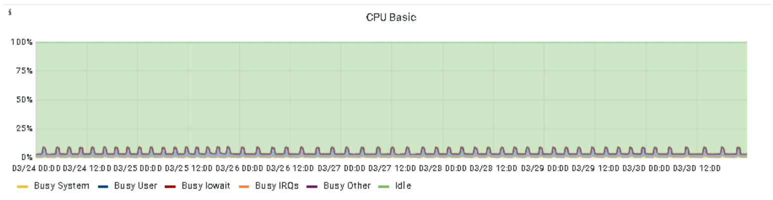
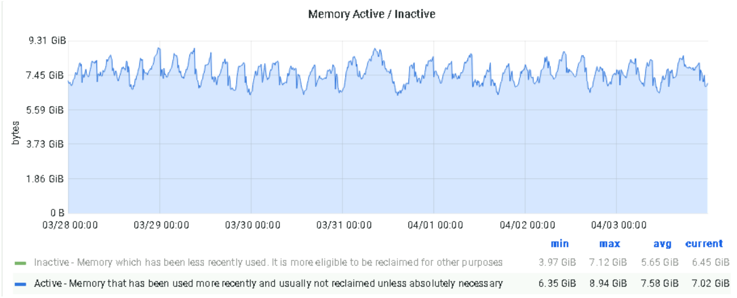


Figure 8 – 48 Core CPU at about 10%

Memory Utilization

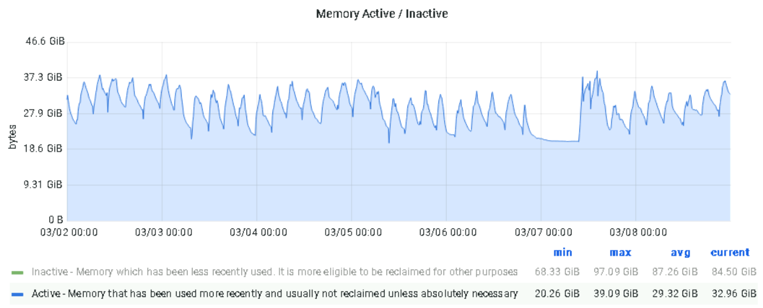
Memory does seem to have a considerable impact on the speed of synchronization, as it has a direct correlation regarding each trial's performance. The command used to run each server as a Geth node specifies that there be 16GB of the memory set aside to be used as cache for the sync (including the cache trial which implements an SSD cache to HDD main storage). For each trial there is a correlation to the amount of RAM provided and the amount of RAM used. **Figure 9, Figure 10, and Figure 11** show the performance of trials from least to most memory to illustrate this point, respectively.

This does indicate the importance of plentiful RAM when beginning the synchronization. Even in the trials with 256GB of memory available, there was still greater utilization with respect to its overall size. Of each metric measured throughout this portion the differences in this section are the most notable.



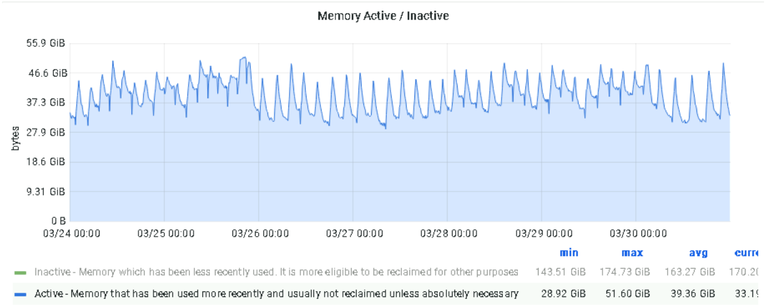
	min	max	avg	current
Inactive - Memory which has been less recently used. It is more eligible to be reclaimed for other purposes	3.97 GiB	7.12 GiB	5.65 GiB	6.45 GiB
Active - Memory that has been used more recently and usually not reclaimed unless absolutely necessary	6.35 GiB	8.94 GiB	7.58 GiB	7.02 GiB

Figure 9 – Trial 1 at average 7.58GB memory utilization (16GB)



	min	max	avg	current
Inactive - Memory which has been less recently used. It is more eligible to be reclaimed for other purposes	68.33 GiB	97.09 GiB	87.26 GiB	84.50 GiB
Active - Memory that has been used more recently and usually not reclaimed unless absolutely necessary	20.26 GiB	39.09 GiB	29.32 GiB	32.96 GiB

Figure 10 – Trial 2 at average 29.32GB memory utilization (128GB)

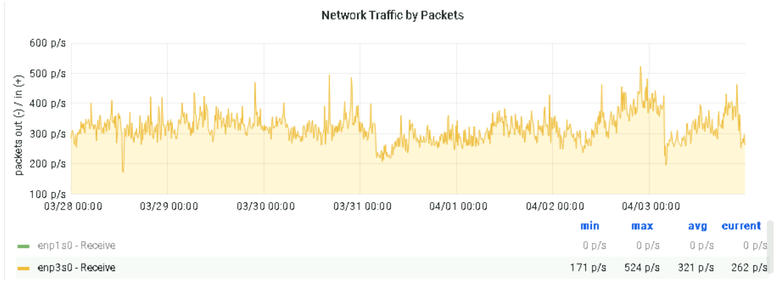


	min	max	avg	curr
Inactive - Memory which has been less recently used. It is more eligible to be reclaimed for other purposes	143.51 GiB	174.73 GiB	163.27 GiB	170.2
Active - Memory that has been used more recently and usually not reclaimed unless absolutely necessary	28.92 GiB	51.60 GiB	39.36 GiB	33.1

Figure 11 – Trials 3/4 at average 39.36GB memory utilization (256GB)

Network Traffic

To synchronize blocks, the Geth node must connect to other peers within the Ethereum network. For this reason, it is important to test and verify the network speeds on each trial to verify that the speeds are comparable. The network speeds are measured in packets per second, and each trial was measured approximately equal to 320 p/s average and 500 p/s max (packets per second). **Figure 12** shows the metrics recorded from Trial 1, but they are similar for each trial.



	min	max	avg	current
enp1s0 - Receive	0 p/s	0 p/s	0 p/s	0 p/s
enp3s0 - Receive	171 p/s	524 p/s	321 p/s	262 p/s

Figure 12 – Average network speeds of about 320 p/s recorded for all trials

This portion is mostly used as a control to ensure that network speeds to not differ among trials. Because of the consistency of this metric across trials, it is not considered a metric of interest for determining optimal setup with regard to collection of metrics examined.

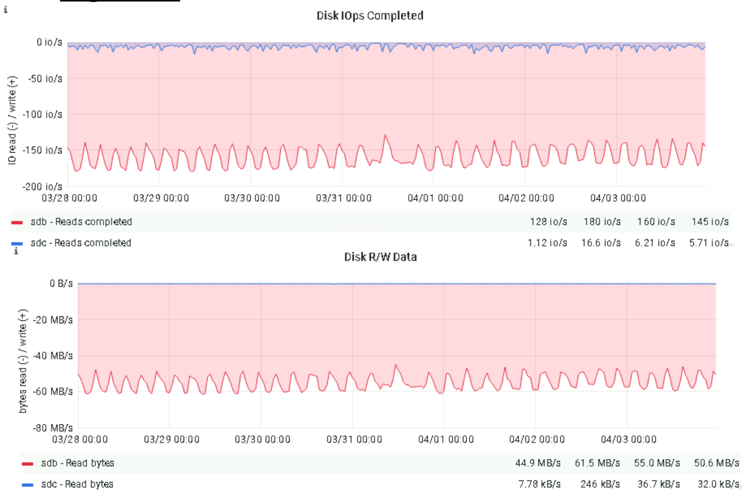
VIII. Disk Performance

Disk Performance is measured in Inputs and Outputs per second (IOps) and amount of Read/Write data (R/W Data). This section will outline the performance of each trial's drives used with respect to read and write performance of each drive. A one-week time frame after each trials 15-day mark is used as a sample for drive performance analysis. Each section will relate to a single trial, and the final portion will contain a table comparison of each trial's drive performance.

Trial 1: SSD Cache to HDD Storage

The Dell XPS running as a server and utilizing a 2TB SSD as cache for a 12TB HDD measured with the overall slowest drive speeds of all trials measured. In terms of reads from disk, IOps for main storage revealed about 160 io/s and 6.21 io/s for the drive acting as cache. Additionally, in terms of writes per second the main storage averages about 9.64 io/s and 9.51 io/s for the drive acting as cache.

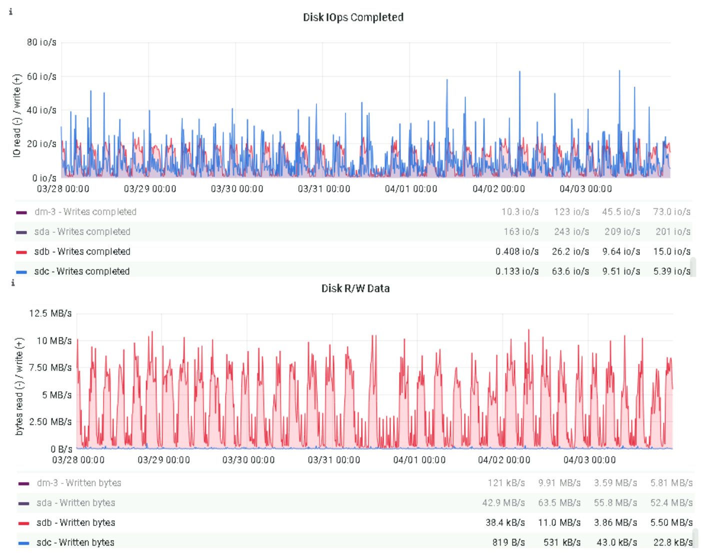
In terms of R/W data reads from disk, main storage averages 55.1 MB/s and 36.7 kB/s for the drive acting as cache. Measuring the R/W data in write performance of each drive, the main storage averages 3.86MB/s, the drive acting as cache averages at 43 kB/s. The IOps and R/W metrics can be visualized in **Figure 13** and **Figure 14**.



sdb - Reads completed	128 io/s	180 io/s	160 io/s	145 io/s
sdc - Reads completed	1.12 io/s	16.6 io/s	6.21 io/s	5.71 io/s

sdb - Read bytes	44.9 MB/s	61.5 MB/s	55.0 MB/s	50.6 MB/s
sdc - Read bytes	7.78 kB/s	246 kB/s	36.7 kB/s	32.0 kB/s

Figure 13 – Trial 1 read performance



dm-3 - Writes completed	10.3 io/s	123 io/s	45.5 io/s	73.0 io/s
sda - Writes completed	163 io/s	243 io/s	209 io/s	201 io/s
sdb - Writes completed	0.408 io/s	26.2 io/s	9.64 io/s	15.0 io/s
sdc - Writes completed	0.133 io/s	63.6 io/s	9.51 io/s	5.39 io/s

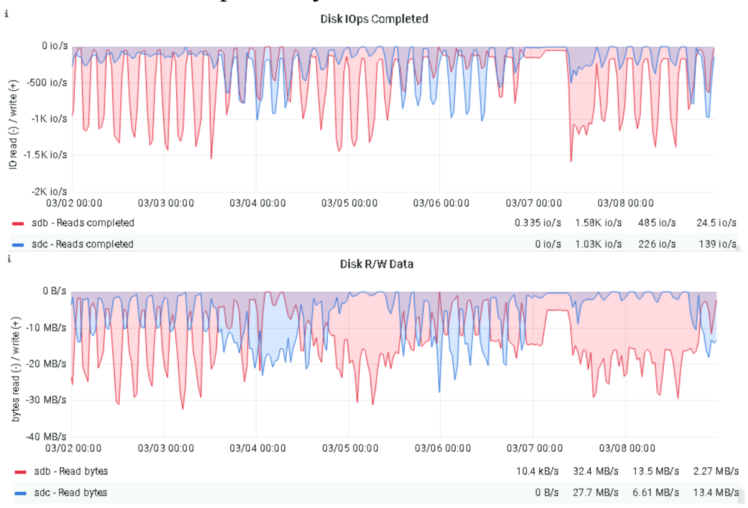
dm-3 - Written bytes	121 kB/s	9.91 MB/s	3.59 MB/s	5.81 MB/s
sda - Written bytes	42.9 MB/s	63.5 MB/s	55.8 MB/s	52.4 MB/s
sdb - Written bytes	38.4 kB/s	11.0 MB/s	3.86 MB/s	5.50 MB/s
sdc - Written bytes	819 B/s	531 kB/s	43.0 kB/s	22.8 kB/s

Figure 14 – Trial 1 write performance

Trial 2: SSD Storage

The second-tier trial combines two 8TB SSDs through LVM to observe the speed of blockchain synchronization to SSD storage. The drives perform considerably faster than those of the previous trial. The *sdb* and *sdc* drives seen in the following figures map to the SSD drives combined together into a single logical volume.

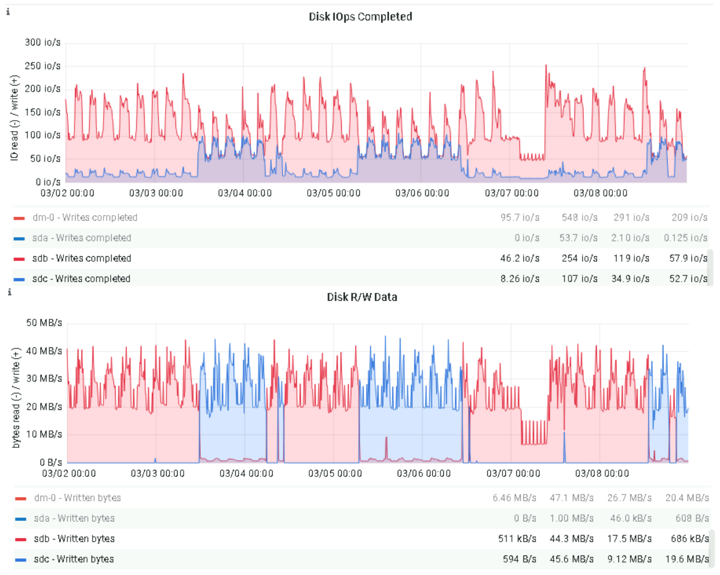
The measured IOps for this trial result in an average of 485 io/s and 226 io/s for each drive performing reads; writes were recorded slower at 119 io/s and 34.9 io/s, respectively. R/W data puts into perspective the amount of data pulled by each drive, again higher than the previous trial. The drives average 13.5 MB/s and 6.59 MB/s in terms of read speeds, and 17.5 MB/s and 9.12 MB/s in writes. **Figure 15** and **Figure 16** visualizes the read and write metrics of these drives, respectively.



sdb - Reads completed	0.335 io/s	1.58K io/s	485 io/s	24.5 io/s
sdc - Reads completed	0 io/s	1.03K io/s	226 io/s	139 io/s

sdb - Read bytes	10.4 kB/s	32.4 MB/s	13.5 MB/s	2.27 MB/s
sdc - Read bytes	0 B/s	27.7 MB/s	6.61 MB/s	13.4 MB/s

Figure 15 – Trial 2 read performance



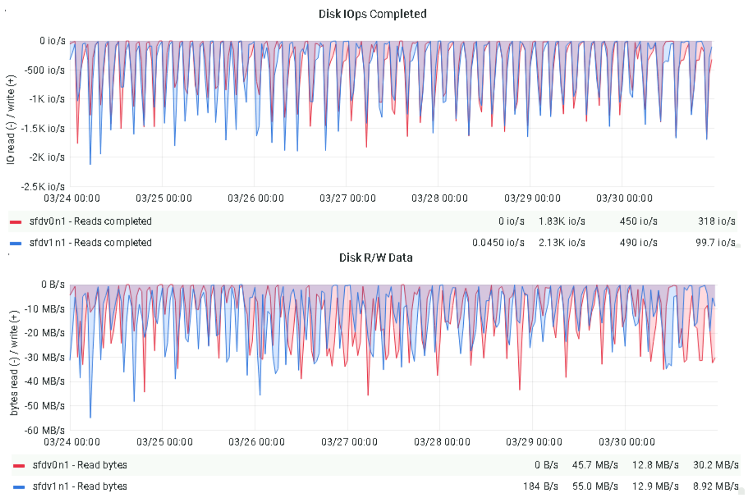
dm-0 - Writes completed	95.7 io/s	548 io/s	291 io/s	209 io/s
sda - Writes completed	0 io/s	53.7 io/s	2.10 io/s	0.125 io/s
sdb - Writes completed	46.2 io/s	254 io/s	119 io/s	57.9 io/s
sdc - Writes completed	8.26 io/s	107 io/s	34.9 io/s	52.7 io/s

dm-0 - Written bytes	6.46 MB/s	47.1 MB/s	26.7 MB/s	20.4 MB/s
sda - Written bytes	0 B/s	1.00 MB/s	46.0 kB/s	608 B/s
sdb - Written bytes	511 kB/s	44.3 MB/s	17.5 MB/s	686 kB/s
sdc - Written bytes	594 B/s	45.6 MB/s	9.12 MB/s	19.6 MB/s

Figure 16 – Trial 2 write performance

Trial 3: Compression NVMe Drives

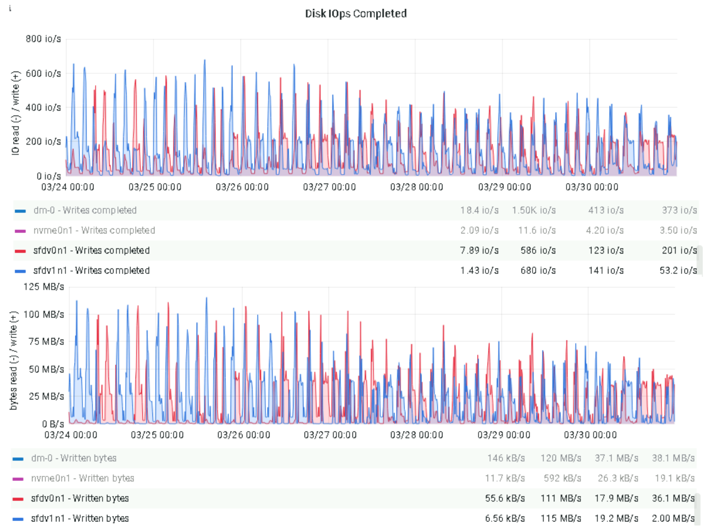
The third trial measures with better performance than the previous two trials. Again, two 8TB compression NVMe drives are combined through the use of LVM to create a single logical volume. IOps read performance are measured at an average of 450 io/s and 490 io/s, respectively; writes are measured at an average of 123 io/s and 141 io/s per drive. R/W data shows the amount of data pulled per second. In this case reads are measured at an average of 12.8 MB/s and 12.9 MB/s for each drive and writes average to be about 17.9 and 19.2 MB/s. **Figure 17** and **Figure 18** visualizes the read and write metrics of these drives, respectively.



sfdv0n1 - Reads completed	0 io/s	1.83K io/s	450 io/s	318 io/s
sfdv1n1 - Reads completed	0.0450 io/s	2.13K io/s	490 io/s	99.7 io/s

sfdv0n1 - Read bytes	0 B/s	45.7 MB/s	12.8 MB/s	30.2 MB/s
sfdv1n1 - Read bytes	184 B/s	55.0 MB/s	12.9 MB/s	8.92 MB/s

Figure 17 – Trial 3 read performance



dm-0 - Writes completed	18.4 io/s	1.50K io/s	413 io/s	373 io/s
nvme0n1 - Writes completed	2.09 io/s	11.6 io/s	4.20 io/s	3.50 io/s
sfdv0n1 - Writes completed	7.89 io/s	586 io/s	123 io/s	201 io/s
sfdv1n1 - Writes completed	1.43 io/s	680 io/s	141 io/s	53.2 io/s

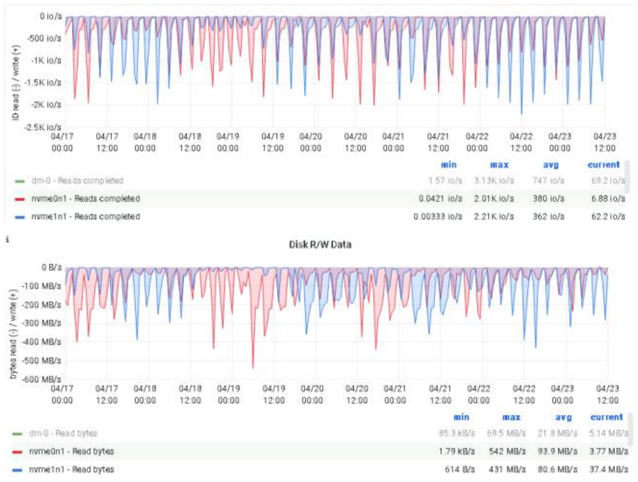
dm-0 - Written bytes	146 kB/s	120 MB/s	37.1 MB/s	38.1 MB/s
nvme0n1 - Written bytes	11.7 kB/s	592 kB/s	26.3 kB/s	19.1 kB/s
sfdv0n1 - Written bytes	55.6 kB/s	111 MB/s	17.9 MB/s	36.1 MB/s
sfdv1n1 - Written bytes	6.56 kB/s	115 MB/s	19.2 MB/s	2.00 MB/s

Figure 18 – Trial 3 write performance

These results appear to indicate the benefit of compression drives, however when compared with other variables such as available RAM used by the server, for example, it is difficult to attribute drive performance to the drives inherent ability to store data. After the first few days of initializing the sync the compression ratio of these drive decreased to 101% (1% compression). This is due to the change in block structure as the sync continues along since blocks have become more complex throughout the Ethereum network's lifespan. It is for this reason the upcoming trial was performed on the same server, to narrow down the effect of compression drives for storing blockchain data.

Trial 4: Standard NVMe Drives

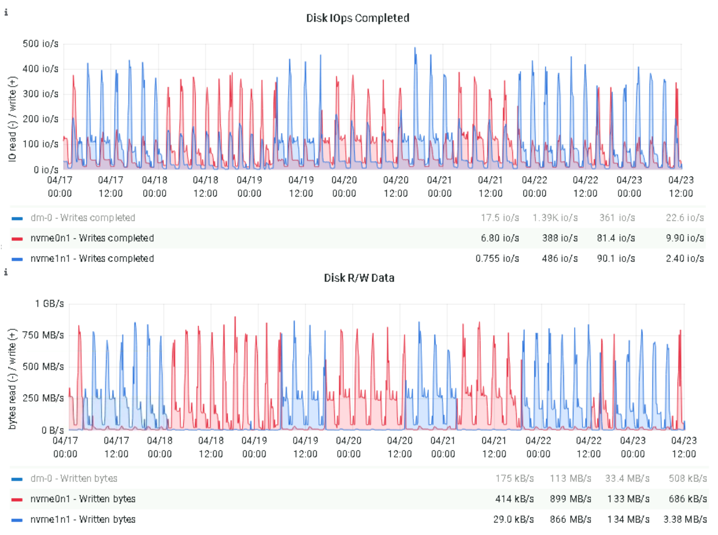
The performance of the standard NVMe is very similar to the drives in the previous trial, which use compression. The metric data varied with respect to those measured, the reads and writes measured by IOps and R/W data. With focus on the write metrics, the recorded R/W data of trial 3 and 4 are quite different when considering how similar their performance is. The recorded R/W for this trial was about seven times higher than that of the previous trial – 18MB compared to 133MB.



	min	max	avg	current
dm-0 - Reads completed	1.57 io/s	3.13K io/s	747 io/s	69.2 io/s
nvme0n1 - Reads completed	0.0421 io/s	2.01K io/s	380 io/s	6.88 io/s
nvme1n1 - Reads completed	0.00333 io/s	2.21K io/s	362 io/s	62.2 io/s

	min	max	avg	current
dm-0 - Read bytes	85.3 kB/s	69.5 MB/s	21.8 MB/s	5.14 MB/s
nvme0n1 - Read bytes	1.79 kB/s	542 MB/s	93.9 MB/s	3.77 MB/s
nvme1n1 - Read bytes	614 B/s	431 MB/s	80.6 MB/s	37.4 MB/s

Figure 19 – Trial 4 read performance



dm-0 - Writes completed	17.5 io/s	1.39K io/s	361 io/s	22.6 io/s
nvme0n1 - Writes completed	6.80 io/s	388 io/s	81.4 io/s	9.90 io/s
nvme1n1 - Writes completed	0.755 io/s	486 io/s	90.1 io/s	2.40 io/s

dm-0 - Written bytes	175 kB/s	113 MB/s	33.4 MB/s	508 kB/s
nvme0n1 - Written bytes	414 kB/s	899 MB/s	133 MB/s	686 kB/s
nvme1n1 - Written bytes	29.0 kB/s	866 MB/s	134 MB/s	3.38 MB/s

Figure 20 – Trial 4 write performance

Trial 5: GCP Trial

Write statistics appear much lower for the GCP trial with respect the amount of write IOps performed, about 20i/o for write speed compared to the other trials which were much higher. This GCP trial became synchronized at about 9.5 TB approximately seven weeks after beginning the synchronization. **Figure 21** shows the percent of disk used for the GCP trial over time, on January 29th the curve changes – at this point the node is fully synchronized.

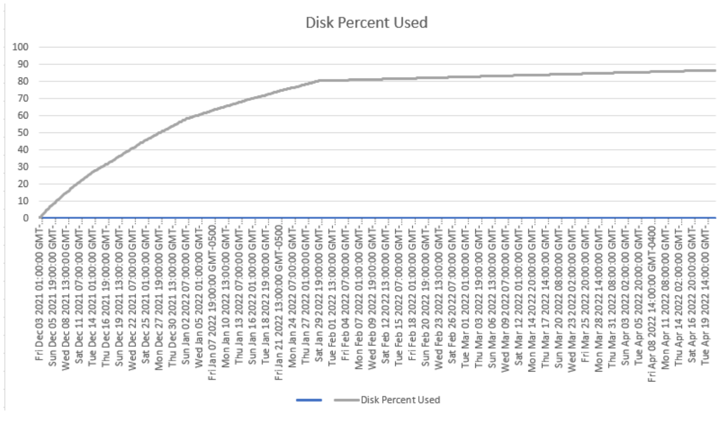


Figure 21 – GCP occupied disk space timeline

IX. Analysis of Results

From the trials tested there is a correlation between the memory available for each trial as well as the type of drive-in use – in both cases the higher-grade drives have more available memory, giving them an extra advantage. The network speeds among trials were measured to be within the margin of error and considered equal, so this variable can be safely disregarded.

Compression of blockchain data is not effective due to the already compressed nature of the blocks themselves – structured in a Merkle-tree [1]. The compression ratio measured in Trial 3 was at 101 percent (or one percent), displaying only slightly improved performance over the standard NVMe trial. For this reason, it is not cost-effective to run a full archive Geth node with compression NVMe drives.

X. Conclusion

Results from trial 1 likely are due to the insufficient resources outside of the disk configuration, however it is plain to see that using a cache does not help with syncing the node because there are likely no requests to read data until the node is fully synchronized. It is reasonable to conclude that caching is not an effective means of reaching a full point of synchronization.

Trial 2 demonstrates the use of SSDs to synchronize a node to the blockchain state much more effectively than the previous trial, but not as well as the following two. This likely is partially due to the decreased resources available to this trial particularly with respect to memory.

Trials 3 and 4 and similar since they were performed on the same server and have identical factors, apart from the disk type in use and the operating system. From the trials conducted it can be seen that compression drives have a slight advantage of standard NVMe, but likely not enough to justify the extra cost of compression drives. The 15-day mark and projection of synchronization show the compression trial slightly ahead of the standard NVMe trial for its entire duration.

References

[1] P. Frauenthaler, M. Sigwart, C. Spanring, M. Sober and S. Schulte, "ETH Relay: A Cost-efficient Relay for Ethereum-based Blockchains," 2020 IEEE International Conference on Blockchain (Blockchain), 2020, pp. 204-213, doi: 10.1109/Blockchain50366.2020.00032.

[2] P. V. Klaine, L. Zhang and M. A. Imran, "An Implementation of a Blockchain-based Data Marketplace using Geth," 2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS), 2021, pp. 15-16, doi: 10.1109/BRAINS52497.2021.9569838.

[3] G. A. Ananthadevan, “Instructions to create an LVM cache for root in Debian,” Gist-GitHub, 15-Mar-2020. [Online]. Available: https://gist.github.com/ntn888/b4d3817cb688e100ab0045ece218363e. [Accessed: 07-May-2022].

[4] “Ethereum/Go-Ethereum at GH-Pages,” GitHub. [Online]. Available: https://github.com/ethereum/go-ethereum/tree/gh-pages. [Accessed: 07-May-2022].

[5] “Prometheus - Monitoring System &amp; Time Series Database,” Prometheus Blog. [Online]. Available: https://prometheus.io/. [Accessed: 07-May-2022].

[6] “Grafana: The Open Observability Platform,” Grafana Labs. [Online]. Available: https://grafana.com/. [Accessed: 07-May-2022].

[7] W. Schwab, “Run an Ethereum node on debian on an external SSD,” Medium, 04-Feb-2020. [Online]. Available: https://betterprogramming.pub/run-an-ethereum-node-on-linux-late-2019-b37a1d35800e. [Accessed: 08-May-2022].

[8] B. Warrior, “How to set up an Ethereum geth node on linux,” EtherWorld.co, 08-Jul-2020. [Online]. Available: https://etherworld.co/2019/12/31/how-to-set-up-an-ethereum-geth-node-on-linux/. [Accessed: 08-May-2022].