ISSN 2706-655X Vol.5, Issue 2: 50-57, DOI: 10.25236/IJFET.2023.050209

# A virtualisation performance evaluation approach for distributed block device

**Linxiangyi Li[1], Haiyan Wang[2,*], Yanjin Wu[3]**

*[1]Distributed Storage, Shaanxi University of Science & Technology, Xi'an, China*
*[2]Signal and Information Processing, Shaanxi University of Science & Technology, Xi'an, China*
*[3]Embedded and Simulator, Shaanxi University of Science & Technology, Xi'an, China*
*[*]Corresponding author*

***Abstract:*** *With the development of the information age, the volume of data is exploding and the operational and performance requirements of massive data are difficult to balance. The high cost of distributed storage systems with high data reliability and excellent random read and write performance, often consisting of multiple racks, multiple nodes and hundreds of disks, is a thorny issue for researchers. A tool to predict the performance of distributed storage architectures has been built based on a study of the cost of distributed storage architectures. Without hardware, enabling low cost and fast performance prediction. It also provides performance data and latency density to compare the performance of different workloads and architectures more intuitively, providing ideas for optimising the performance of distributed systems.*

***Keywords:*** *Low cost, No hardware, Performance evaluation, Distributed systems*

## 1. Introduction

Distributed storage architecture is the most widely used storage architecture today [1-4]. While possessing high performance and reliability, distributed storage supports high-capacity scaling and horizontal expansion [5,6] and can meet long-term data storage needs in the information age where data volumes are increasing. Distributed storage is divided into three categories: object storage [7], block storage [8,9] and file storage, of which block storage has the most outstanding read and write performance, combined with redundancy technology to ensure high data reliability, and can be used in hybrid disk arrays [10].

Highly configurable large-scale distributed storage systems require high hardware investment, and the cost of hardware devices has become a stumbling block for research work [11,12]. At the same time, the performance of block storage is greatly affected by hardware configurations, and the performance metrics obtained from different combinations of hardware configurations vary greatly [13-16]. In order to comprehensively evaluate the impact of different components on performance, performance testing of various hardware combinations is required, which makes evaluating data manipulation and performance testing very tedious and labor-intensive and time-consuming just for research environment setup.

To this end, this paper proposes a performance evaluation platform for virtualisation of block storage, the Shape simulator, designed with interception and pre-processing algorithms to extract workloads, which enables developers to find breakthroughs in optimising the architecture more quickly.

## 2. Related Research

In previous research on distributed storage systems, most developers would take two approaches to predict device performance: first, predicting based on experience [17], and second, getting actual devices for testing [14-16]. The former is done by testing a small number of physical devices and thus extrapolating the performance of the majority of devices, but distributed storage has a complex structure in a non-simple linear relationship that cannot be simply calculated to obtain performance data. As well as taking into account factors such as development cost and efficiency. The plausible simulator proposed in this paper can discard the physical equipment required for distributed systems and predict system performance in the pre-development phase, reducing the financial pressure on the researcher and compressing the development cost of the study.

ISSN 2706-655X Vol.5, Issue 2: 50-57, DOI: 10.25236/IJFET.2023.050209

Common testing software includes Flexible I/O [18], Iozone [19], etc. The IOP parameters in these tests are artificially set, such as the size of the data to be operated, the read/write ratio, etc., but these test data often do not match the real work. In this paper, the proposed plausible simulator uses bcc-tools (BPF compiler collection) [20] to trace IOPs to get the workload, and the method can be applied to any software test, and the tpc series test will be used in this paper [21].

## 3. Imeplementation

The performance predictions in this paper do not require any hardware devices, which means that no software programs can be installed and run on them. Therefore, this paper obtains test results by obtaining the workload of a real program in operation and running it simulated on a plausible simulator.

### *3.1. Workload acquisition process*

Figure 1 shows the process of acquiring and processing the workload in this paper. The initial workload is obtained by bcc-tools interception during the application run, and then the initial snoop file is processed by the algorithm Pregen. This algorithm first filters the initial snoop file to ensure the validity of each IO; and then completes some environment variables needed in the snoop file to generate the complete block device request.

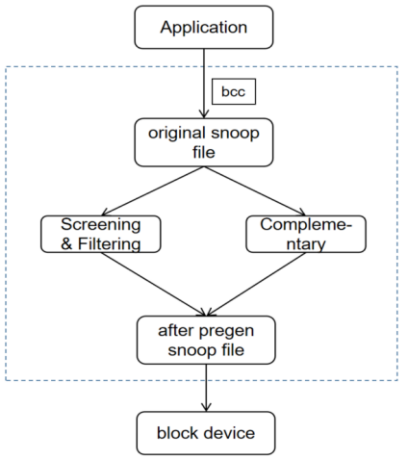


*Figure 1: Workload acquisition flow diagram*

#### *3.1.1. BCC interception*

Bcc-tools [20] is a collection of tools for bpftrace, where biosnoop is a trace record of block device IO to provide performance prediction and analysis to plausible simulators. biosnoop intercepted IO records (referred to as snoop files) are shown in Figure 2:

```
TIME(s)     COMM           PID    DISK    T SECTOR     BYTES  LAT(ms)
0.000000    jbd2/nvme0n1p1 746    nvme0n1 W 417109896  12288     0.04
0.000172    jbd2/nvme0n1p1 746    nvme0n1 W 424968456  151552    0.12
0.001611    jbd2/nvme0n1p1 746    nvme0n1 W 424968752  4096      0.74
```

*Figure 2 Sample snoop file*

As shown in Figure 2 the snoop file contains TIME (relative time), COMM (command), PID (process id), DISK (located disk), T (type of operation), SECTOR (sector number where the operation starts), BYTE (size of the operation in bytes), and LAT (latency in milliseconds). Each line in the Snoop file (excluding the header line) is an IOP that initiates a request to the block device.

#### *3.1.2. Pregen algorithm*

There will be underlying threads running simultaneously in the Linux system, which will result in many unrelated and hard to distinguish IOPs in the intercepted snoop file, requiring preprocessing operations of the snoop file. In this paper, we first use a separate disk for testing, and then filter out the invalid IO through the Pregen algorithm. Invalid IO is generally manifested as.

1) the items recorded in the Snoop file are incomplete.

ISSN 2706-655X Vol.5, Issue 2: 50-57, DOI: 10.25236/IJFET.2023.050209

2) the disk on which it is located is not the disk used for the test.

3) the sector where it is located is outside the range of the underlying physical device.

4) the sector on which it is located is outside the range of the underlying physical device.

The entire application process can be divided into two phases: the data generation phase and the data run phase (referred to as p1, p2). Ideally, the application completes all initial data writing operations and metadata creation in p1, and the p2 phase runs the data written in p1. However, analysis of the p1 snoop file reveals that the p1 phase reads the environment information of the underlying system and the environment information of the test program, and as this data is written earlier than the BCC intercept time, it will be treated as unknown data directly causing the corresponding operation to fail. To ensure the integrity of the workload, the Pregen algorithm will simulate running from a metadata-free space, complementing the missing IOPs. The pseudo-code of the Pregen algorithm is as follows:

Algorithm 1 Pregen.

Input: Initial snoop file.

Output: After processed snoop file.

```
step1. //read the IOPs in the snoop file
read();
step2. //check if the data for each IOP exists
if (items ! = 8)
continue;
step3. //check the reasonableness of each data of IOP
if (disk ! = TESTDISK)
continue;
if (sect > MAXDISK)
                continue;
if (byte>MAXBYTE || byte/4096!=0 || byte==0)
continue;
step4.//Find the page where the IOP is located
pno= getpg(sect, byte);
step5.//check the page data and add
if (OP == READ)
int miss = Check(pno);
else
save(pno);
if (miss)
add(IO);
step6. out(snoop);
```

The Pregen algorithm first filters for valid IOs and assumes that the entire workload is running from a metadata-free store. And records the generation operation of this data in the form of a snoop file based on it. Without the Pregen algorithm, these IOPs would be identified by the Shape simulator as spurious or worthless IOPs and discarded, resulting in a lack of integrity in the loaded IO load and further causing the system to crash. The lightweight Pregen algorithm yields pure and complete workloads as real IO requests in the fastest way possible.

ISSN 2706-655X Vol.5, Issue 2: 50-57, DOI: 10.25236/IJFET.2023.050209

### *3.2 Design and implementation of the Shape simulator*

#### *3.2.1 Environment variables*

User-adjustable parameters are provided in the Shape simulator, as shown in Table 1.

*Table 1: Environment variables*

| environment | Implication | example |
|---|---|---|
| NSIMPOOL | number of simpool | NSIMPOOL=1 |
| QD | number of queue depth | QD=32 |
| QPG | max page in a batch | QPG=2048 |
| SNOOP_<n> | path of workload | SNOOP_1=TPROC-C/p1.txt |

#### *3.2.2 Structure design for IO requests*

The IO information output after bcc interception is in snoop file format, and IOPs in this format cannot be directly applied to the Shape simulator. In this paper, the lunioq structure is designed to be used as an executable IOP for the Shape simulator, and the IO queue generated by lunioq is made available to the Shape simulator at the beginning of its operation. Lunioq is designed as shown in Table 2.

*Table 2: Lunioq structure*

| variable | implication |
|---|---|
| Ts,ets | start time and end time of IOP |
| Op | operation type of IOP |
| Simidx | simpool index |
| Poolid | pool index |
| Lunid | LUN id |
| Ls64 | ls64 number of IOP on the lun (ls64 has 64 pages) |
| Ls64pno | offset on the ls64 (page) |
| Npg | number of page |
| ser | IOP index |
| Done | done flag, if finish will set to 1 |

#### *3.2.3 Design and implementation of the Shape simulation*

If the lunioq queues were pressed directly into the Shape simulator, the inter-constraints of real operation would be lost and the simulated runtime flow would lack realism. To solve this problem, the dep algorithm is designed to analyse the dependencies of a series of IOPs and establish dependencies that constrain the operation. Only the dependent IOP (referred to as depIO) will be run before the next IOP can be run. Dep algorithm will analyse the depIO based on the information in the snoop file and store it in the relationship table (adep table). Dep algorithm is as follows.

Algorithm 2; IO dependency analysis algorithm dep

Input: After Pregen snoop file.

Output: Dependency relationship table for IOPs adep

1. // Read IO_1 and calculate the relative start time of IO_1 in the program tim

Read(IO)

Time = Gettime(IO)

2. // Read the operations in the IO queue sequentially and find depIO

Read(nextIO)

Nexttime = time(nextIO)

isDep = Compare(time, nexttime)

3. //If their running times overlap then depIO for IO_3 is IO_1, otherwise depIO for IO_2.

if (isdep)

put(IO, nextIO)

ISSN 2706-655X Vol.5, Issue 2: 50-57, DOI: 10.25236/IJFET.2023.050209

Else

4.// Read the next IO in a loop and repeat steps 2-4

if (end)

Out(adep_table);

*Table 3: Example of an adep table*

| IO | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| depIO | 0 | 1 | 1 | 2 | 2 | 3 | 5 |

The adep table, created by the dep algorithm, holds the correspondence between each IO and depIO, as shown in Table 3. IO_1 is a standalone operation. depIO for both IO_2 and IO_3 is IO_1. The Shape simulator will use the information in the adep table to rationalise the order in which the IOs are run and automatically allocate resources to complete the IOPs. Before executing each lunioq, the adep table is queried to see if there is a depIO and if the depIO has finished executing, and when the depIO has finished executing or there is no depIO, the current lunioq can start executing dep algorithm will restore the interlocking relationship between IOPs in program operation, effectively simulate the real operation state and provide valuable performance analysis results.

### *3.3 Performance Data Acquisition*

In this paper, throughput, latency density and runtime are provided as performance data. Latency density can be used to compare the IO latency performance between different test programs or different devices.

#### *3.3.1 Storage pool information*

Based on the IO load running in the plausible simulator, the storage pool information records the type of operation, batch operation, database interaction and other information of the test program respectively, as shown in Table 4.

*Table 4: IO type*

| op | implication |
|---|---|
| prewrite | number of prewrite in the workload |
| postwrite | number of postwrite in the workload |
| rd | number of read in the workload |

1) Types of operations

This article introduces two concepts when analysing the performance of workloads: prewrite and postwrite.

prewrite: a write operation when data is first written to the data segment

postwrite: a non-first write operation to the segment

The first time data is written to a new data segment, the metadata for that segment is created. The two types of write operations are different in terms of resource utilization and need to be distinguished in order to analyse the composition of the IO load for developers to tune the system to improve performance.

Assuming that the size of a data segment is seg_size (in pages), the start disk number of a write operation is sect_num, the number of blocks per page is sect_per_page, and the data segment in which the write operation is performed is seg_num (ignoring cross-segment operations), the formula is as follows.

$$\mathrm{seg}\_num = \mathrm{sec}t\_num / \mathrm{sec}t\_per\_page * seg\_size \quad (1)$$

$$new(seg\_num) = \begin{cases} 1, prewrite \\ 0, postwrite \end{cases} \quad (2)$$

Assuming a total of N IOs in the workload, the relationship between operations is formulated as follows, where read's represent read operations and cross's represent cross-segment operations.

ISSN 2706-655X Vol.5, Issue 2: 50-57, DOI: 10.25236/IJFET.2023.050209

$$\mathrm{N} = \sum_0^N read - \sum_0^N postwrite + \sum_0^N prewrite - \sum_0^N cross \tag{3}$$

This is more comprehensive than traditional information such as throughput and rate, analysing detailed information about the IO load in operation and providing clear ideas for optimisation.

#### *3.3.2 Latency density information*

As mentioned in Section 3.2.2 each lunioq has a request start time ts and a run end time ets. To obtain delay time information, the Shape simulator internally stores the start run time sts for each IOP and the delay lat for the m lunioq is calculated as follows:

$$\mathrm{lat}(m) = sts(m) - ts(m) \tag{4}$$

After executing all the IOs, the Shape simulator sorts all the collected IO latency to obtain the maximum latency Max and the minimum latency Min. According to the number of IOs, all the IOPs are divided into n groups, with the same number of IOPs in each group, and the latency range of each group is recorded. Assuming that the total number of IOs is N, the delay range for group i is calculated as follows.

$$\max\_lat = \max((\frac{N}{n})*(i-1), (\frac{N}{n})*(i-1)+1, \cdots, (\frac{N}{n})*i-1) \tag{5}$$

$$\min\_lat = \min((\frac{N}{n})*(i-1), (\frac{N}{n})*(i-1)+1, \cdots, (\frac{N}{n})*i-1) \tag{6}$$

Where max_lat (i) represents the maximum latency for group i min_lat(i) represents the minimum latency for group i.

Usually most of the IOPs in a program have concentrated latency, but individual IOPs have particularly high latency, and the minimum and maximum latency are not representative enough to allow a comparative study between different workloads or different devices. The latency density proposed in this paper not only provides the maximum and minimum latency, but also explicitly lists information on the density of IO latency. The performance of different workloads on the same device, or the same workload on different devices, can be compared and analysed between control groups to provide clearer optimisation recommendations to developers.

## 4. Results

### *4.1 Test procedure and environment*

In this paper, we use TPROC-C provided by HammerDB [22] to perform pressure measurements, using postgreSQL as the database. P1 and p2 phases are included in the workgroup, for a total of two workloads.

The workgroup was configured with Ubuntu 20.04, kernel version 5.4.0-050400-generic, a six-core processor and ext4 5TB SATA HDDs.

The following test results simulate the following multi-tier storage device configuration: 10 1.6GB SSDs with DWPD of 3 and 10 20TB HDDs with 2 copies on the SSDs, striped HDDs with a redundancy policy of RAID6, and metadata storage provided by a key-value database on the SSDs.

### *4.2 Test results*

In the test program, the workload is divided into two parts: generating the test data workload (referred to as p1) and executing the test data workload (referred to as p2). In a real-world usage environment, p1 is executed much less often than p2, so in the following tests, the focus is more on the performance of p2.

The results of the TPROC-C working group tests in the NSIMPOOL=1, QD=32, QPG=2048 environment are shown in Table 5.

ISSN 2706-655X Vol.5, Issue 2: 50-57, DOI: 10.25236/IJFET.2023.050209

*Table 5: Results for NSIMPOOL=1, QD=32, QPG=2048*

| work load | throughput (MBS) | elapsed time (s) | type summary | latency density (us) |
|---|---|---|---|---|
| p1 | 680.94 | 55.79 | prewr 44300<br>postwr 242770<br>rd 77231 | max 42086 min 0<br>0% 0~21 10% 37~52 20% 86~171 30% 227~282 40% 376~461 50% 582~737 60% 883~1061 70% 1268~1511 80% 1809~2218 90% 2854~4067 |
| P2 | 234.96 | 80.59 | prewr 13747<br>postwr 606058<br>rd 353550 | max 23057 min 1<br>0% 1~64 10% 118~178 20% 247~324 30% 406~486 40% 575~664 50% 757~850 60% 947~1050 70% 1163~1292 80% 1443~1633 90% 1902~2374 |

The throughput of workload P2 is less than 500 MBS when the queue depth is 1. 50% of the IO latency of workload p2 is around 800 microseconds, and only 35% of the IO is able to keep the latency below 500 microseconds. Performance can be optimised by adjusting the value of the environment variable QD or QPG. With the queue depth increased to QD=24 and all other environment variables held constant, the results of the tests are shown in Table 6 (only the results for workload p2 are presented).

*Table 6: Results for NSIMPOOL=1, QD=24, QPG=2048*

| workload | throughput(MBS) | elapsed time (s) | type summary | latency density(us) |
|---|---|---|---|---|
| P2 | 648.92 | 29.18 | prewr 13758<br>postwr 603612<br>rd 353550 | max 98909 min 1<br>0% 1~33 10% 49~64 20% 90~112 30% 128~141 40% 153~164 50% 176~1884 60% 204~225 70% 252~312 80% 426~560 90% 693~943 |

Increasing the queue depth resulted in better system performance, as shown in the results above, with a 94% reduction in transaction processing time in batch processing and 80% of the IO latency in the workload staying below 500 microseconds.

The test results when NSIMPOOL=4, QD=24 and QPG=2048 are shown in Table 7.

*Table 7: Results for NSIMPOOL=4, QD=24, QPG=2048*

| workload | Throughput (MBS) | elapsed time (s) |
|---|---|---|
| P2 | 2639.11 | 28.7 |

The Shape simulator also supports running different workloads in parallel, i.e. different workloads on each simulated storage pool. The Shape simulator can be applied to any workloads and this paper only uses the TPROC-C in HammerDB as examples.

## 5. Concluslon

In an environment of high demand for distributed block device storage systems, in order to address the high development costs and post maintenance issues, this paper has focused on the development of a low cost physical device for undistributed systems and proposed a low cost performance prediction through a Shape simulator. In addition, the paper designs a workload acquisition method and the Pregen algorithm supports any test program for input and testing of the Shape simulator, enriching the performance test metrics and providing a clearer idea for developers to optimise system performance. The tpcc test is used as an example to demonstrate the feasibility of the plausible simulator and to optimise the performance based on the performance test metrics. The next research focuses on analysing other characteristics of the workload for performance prediction and a more comprehensive consideration of the optimisation path for distributed block devices.

## References

*[1] Raja F R, Akram A. A Highly Scalable and Efficient Distributed File Storage System [J]. 2022.*
*[2] Weil S A. Ceph: reliable, scalable, and high-performance distributed storage [J]. Santa Cruz, 2007.*
*[3] Zhangling W U, Wei M. Distributed storage system, data processing method, and storage node, US11262916B2 [P]. 2022.*
*[4] Mak R, Kalyanakrishnan A, Song G Y, et al. Object Tiering in A Distributed Storage System,*

ISSN 2706-655X Vol.5, Issue 2: 50-57, DOI: 10.25236/IJFET.2023.050209

*US2022121364A1 [P]. 2022.*
*[5] Inggs G, Thomas D B, Luk W. An Efficient, Automatic Approach to High Performance Heterogeneous Computing [J]. Computer Science, 2015.*
*[6] Li Z, Shen H. Measuring Scale-Up and Scale-Out Hadoop with Remote and Local File Systems and Selecting the Best Platform [J]. IEEE Transactions on Parallel and Distributed Systems, 2017, 28(11): 3201-3214.*
*[7] Aung, Mi K M, Zhu, et al. Building a large-scale object-based active storage platform for data analytics in the internet of things [J]. Journal of supercomputing, 2016.*
*[8] Zheng S, Chen R, Jin Y, et al. NeoFlow: A Flexible Framework for Enabling Efficient Compilation for High Performance DNN Training [J]. IEEE Transactions on Parallel and Distributed Systems, 2021.*
*[9] Batchu R K, Seetha H. A Hybrid Detection System for DDoS Attacks Based on Deep Sparse Autoencoder and Light Gradient Boost Machine[J]. Journal of Information & Knowledge Management, 2023, 22(01).*
*[10] Katkar R, Kulkarni K. Study on Block Device Driver and NVMe their Implementation Impacts on Performance. 2014.*
*[11] Shen Z, Chen F, Yadgar G, et al. One Size Never Fits All: A Flexible Storage Interface for SSDs [C]// 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2019.*
*[12] Wadhwa B, Byna S, Butt A. Toward Transparent Data Management in Multi-layer Storage Hierarchy for HPC Systems. 2018.*
*[13] Office of Law Enforcement Standards (OLES)United States of America, America U. Test Results for Hardware Write Block Device: FastBloc IDE (Firmware Version 16) [J]. Bureau of Justice Statistics.2006*
*[14] M Norell. Test Results for Hardware Write Block Device: Fastbloc Fe (Firewire Interface) [J]. 2012.*
*[15] Yang Y, Huang C, Zhang Y, et al. Processable Potassium–Carbon Nanotube Film with a Three-Dimensional Structure for Ultrastable Metallic Potassium Anodes[J]. 2022.*
*[16] Nguyen L D, Leyva-Mayorga I, Popovski P. Witness-based Approach for Scaling Distributed Ledgers to Massive IoT Scenarios[J]. 2020.*
*[17] Combs V T, Hurley P, Schultz C B, et al. DISTRIBUTED SYSTEM EVALUATION [J]. 2022.*
*[18] axboe/fio: Flexible I/O Tester [EB/OL]. [2022-12-23]. https://github.com/axboe/fio*
*[19] Iozone Filesystem Benchmark [EB/OL]. [2022-12-23]. www.iozone.org/*
*[20] BCC-Tools for BPF-based Linux IO analysis, networking, monitoring, and more [EB/OL]. [2022-12-23]. https://github.com/iovisor/bcc*
*[21] TPC benchmarks [EB/OL]. [2022-12-23]. www.tpc.org/*
*[22] HammerDB open-source software with source code hosted by the TPC [EB/OL]. [2022-12-23]. https://www.hammerdb.con*