**Analysis of SSD architecture and read/write on Solid State Drive**

**Analysis of SSD architecture and read/write on Solid State Drive**

A project report submitted in partial

fulfillment of the requirements for the degree of

Master of Science

By

MENG XU

DRISHTI GOEL

AESHA ALTHNYAN

CATHALINA FONTENELLE

May 2019

University of Colorado Denver

Table of Contents

[**1.**](#_gjdgxs) **Introduction 1**

[1.1 Motivation 1](#_30j0zll)

[1.2](#_1fob9te) Scope 1

[1.3](#_3znysh7) Testing Environment & Platforms 2

[**2.**](#_2et92p0) **IMPLEMENTAION 3**

[2.1 Intel Open Storage Kit 3](#_tyjcwt)

[2.1.1](#_3dy6vkm) Implementation Detail 3

[2.2](#_4d34og8) Eagle Tree Simulator 11

[**3.**](#_2s8eyo1) **Challenges 13**

[**4.**](#_17dp8vu) **Conclusion 14**

[**References 15**](#_3rdcrjn)

# Introduction

## 1.1 Motivation

The main motivation for doing this project was to understand the internal architecture and working of a Solid-State Drive. Since we already know that SSD’s are way faster than traditional spinning disks, we wanted to research more into the internal architecture and mechanism of Flash based drives.

One of the key factors which got our attention was the presence of different factors that delimited the SSD performance and we tried to understand and implement these intrinsic factors.

## Scope

The project will span across the implementation of different factors that have affect the overall performance of an SSD. These tests are implemented using Intel Open Storage Toolkit. The factors are:

1. Observe individual time performances for sequential and random reads/writes.
2. Effects of increased read randomness on latency.
3. Effects of increased random writes on the performance.
4. Effect of disk cache on the performance.
5. Read and Write Interference with each other.
6. Background operations affecting the performance.
7. Effect of Increased workload randomness (seek range) on performance.
8. Effect of Fragmentation on the performance.

Due to lack of the time, we were able to analyze on the first five factors and the last three factors are kept for the future work. Apart from these factors, we also implemented using Eagle Tree.

## Testing Environment & Platforms

* Installing Intel Open Storage Tool Kit on the system being tested.
* Heracles –
  + HGST SN100 1.6TB NVMe 2.5” SSD
  + 120GB Intel DC S3510 2.5” SATA 6Gbps MLC SSD
* Eagle Tree Simulator

# IMPLEMENTAION

## 2.1 Intel Open Storage Kit

It contains and iSCSI initiator and target, a SCSI RAM disk, a block-level micro-benchmark, a performance monitor and I/O tracing and replay. It can generate various typesof I/O workloads to directly access block devices with different configurations, such as read/writes ratio, random/sequential ratio, request size, time etc. It reports bandwidth, IOPS and latency [1].

## Implementation Detail

The ./fitness command is used with various switches to check for the performance test. The below switches have been used for ./fitness for the project implenetation [2]:

* ***--wr <int>*** This switch specifies the read to write ratio for the workloads in % “--wr 0” would be all reads and “- -wr 100” would be all writes on the workload.
* ***--wrnd <int>*** This switch specifies the % of random writes among the write operations in the workload. “- -wrnd 0” would be sequential writes in the workload.
* ***-- rrnd<int>*** This switch specifies the % random reads among the write operations in the workload. “- - rrnd 0” would be 100% random reads and “—wrnd 0” would be sequential reads in the workload.
* ***--qdep <int>*** This is the queue depth and specifies in int value as to how many operations can be queued at a time.
* ***-- wrsz <int>*** This switch specifies the write request size in the workload and can be at maximum of 256 Kb for this tool.
* ***--rdsz <int>*** This switch specifies the read request size in the workload and can be at maximum of 256 Kb for this tool.
* ***--wr\_stride<int>*** In Kb is the stride value for the data to be written in the memory during the write operations.
* ***--rd\_stride<int>*** In Kb is the stride value for the data to be read from the memory during the read operations*.*
* ***--Iters <int>*** This switch specifies the number of iterations to be performed to run multiple test for consistency check.
* ***--direct*** This switch if specified, runs the workload directly on the SSD without the influence of the page cache.
* ***--warm <int>*** This switch is used to warm all data blocks to maintain consistency among all bocks rather than hot or cold blocks.
* ***--test <int>*** This is time in seconds that the test should be performed.
* ***-- cap <int>*** In Mb specifies the capacity of the test file to be used for the operations.

The above directions can be utilized in varieties as changes to ./fitness to play out the execution of the task and check for the execution measures with yield in the organization of Throughput in Mb/sec, IOPS and Latency in milliseconds.

**Note:** The commands run on the master node SSD by default. Switch to any node to perform analysis on a node such as: ssh node10

The commands and their respective conditions check is as below:

The defaults used for all command to mainatain a uniform performance check are:

--qdep 64 \*\*queue depth of 64\*\*

-- wrsz 64 \*\* write request size of 64Kb\*\*

--rdsz 64 \*\* read request size of 64Kb\*\*

--rd\_stride 4 \*\* read stride size of 4Kb\*\*

--wr\_stride 4 \*\* write stride size of 4Kb\*\*

We have played out the accompanying investigations to comprehend the various parts of execution in SSD.

1. **Read Write base parameters with and without cache influence**

In this we have gathered different experiment for Random/Sequential Read/Write as the base values without any hindrances. We have tested in two scenarios with cache and without cache. The experiments results are shown in the table and graphs below.

We can see that the Read is the fastest in terms of the cache. In terms of cache influence we can see the if we use the cache the SSD is far faster than the non-cache but in terms of write we can see little difference in the performance with the cache. All the three graphs shown below have same results except for the one anomaly which is the random write in non-cache is faster than the sequential one. This may be due to background operation.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Latency** |  |  | **Bandwidth** |  | **Band width** | |  |  | **IOPS** |  |  | **IOPS Master** |  |  | **Latency** |  |  | **Latency** |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | **node** |  |  | **Master** |  |  |  |  |  | **Node** |  |  |  |  |  | **Master** |
|  |  |  |  |  |  | **Node** |  |  |  |  |  |  |  |  |  |  |  | **Node** |
| Random Read |  |  | 37627.67 |  |  | 35840 |  |  | 602039 |  |  | 573444 |  |  | 0.01 |  |  | 0.02 |
| with cache |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Random Write | | 66.46 | |  |  | 2734 |  | 1060 | |  |  | 43745 |  | 33.42 | |  | 1.23 | |
| with cache | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Sequential Read |  |  | 38467.49 |  |  | 36513 |  |  | 615479 |  |  | 584217 |  |  | 0.02 |  |  | 0.02 |
| with cache |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Sequential Write | | 86.96 | |  |  | 2388 |  | 1385 | |  |  | 38215 |  | 7.38 | |  | 1.14 | |
| with cache | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Random Read |  |  | 1418.03 |  |  | 292.96 |  |  | 22682 |  |  | 4635 |  |  | 2.82 |  |  | 13.81 |
| Without cache |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Random Write | | 292.24 | |  |  | 162.57 |  | 4669 | |  |  | 2595 |  | 13.69 | |  | 24.69 | |
| without cache | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Sequential Read |  |  | 1397.2 |  |  | 309.69 |  |  | 22357 |  |  | 4949 |  |  | 3.27 |  |  | 12.93 |
| without cache |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Sequential Write | | 220 | |  |  | 164 |  | 3529 | |  |  | 2618 |  | 18.17 | |  | 24.34 | |
| without cache | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Table -1

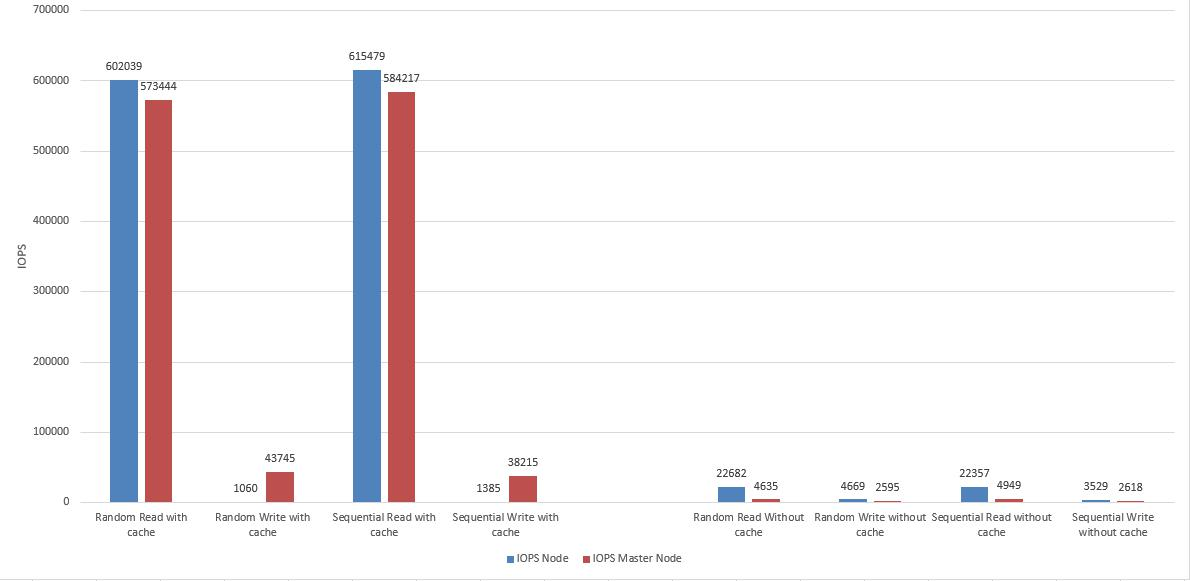
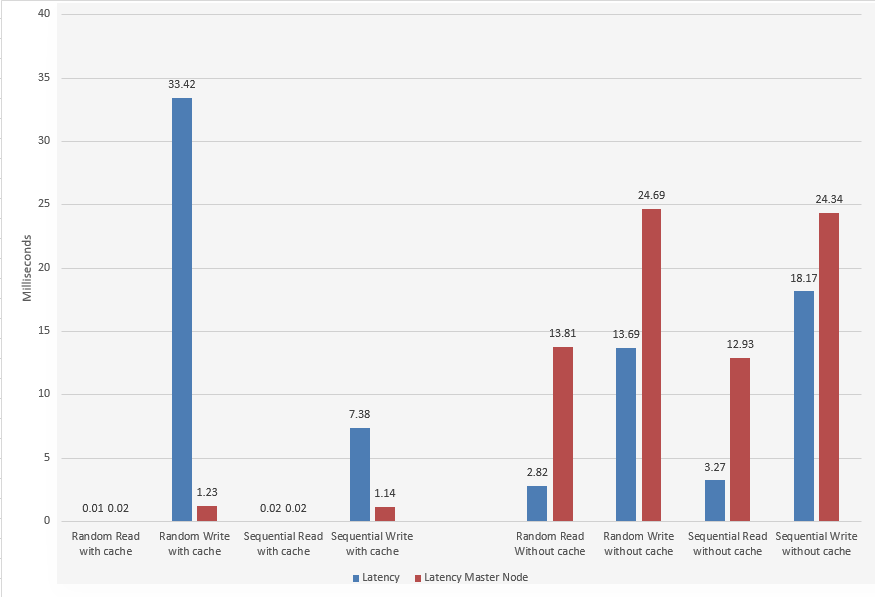


Figure 1 – IOPS



1. **Is random write the worst case?**

The beneath direction is utilized to run outstanding task at hand with 100% write activities and 0% randomness:

./fitness -- file test -- wr 100 -- wrnd 0 -- qdep 64 -- wrsz 64 -- rdsz 64 -- wr\_stride 4 -- rd\_stride 4 –- rrnd 0 -- iter 3

The above direction is then rehashed with – wrnd values 10,20,30,40,50,60,70,80,90,100 to check for the write tasks with the respective random write activities to check if the 100% random is the worst scenario for SSD execution. The values incremented in the above statement from 10 to 100 specifies the percentage of random writes thereby checking each condition from random writes being 10% to the 100 % random writes thereby testing is the worst case of performance is measured at 100% random writes irregular composes.

The – iter 3 switch demonstrated outcomes for 3 cycles to assess the accuracy of the outcomes. The outcomes for this question alongside its correlation with the master node SSD execution are as underneath. This table gives a lot of recorders esteems to screen the execution dependent on the latency metrics considering the % of random writes from 10 augmenting towards 100 theorizing the outcome to tend towards the worst scenario of latency.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Write random% |  |  |  |  |  | |  |  | | |
|  |  |  |  | Latency |  |  | | |
|  |  |  | Iter1 |  |  | Iter2 | Iter 3 |  |  | | |
| 0 | |  | 8.55 |  |  | 9.69 | 8.19 |  | |  |
|  | 10 |  | 17.87 |  |  | 12.61 | 11.82 |  |  | | |
|  | 20 |  | 14.72 |  |  | 21.61 | 16.34 |  |  | | |
|  | 30 |  | 27.47 |  |  | 18.53 | 11.96 |  |  | | |
|  | 40 |  | 25.55 |  |  | 29.83 | 31.75 |  |  | | |
|  | 50 |  | 38.25 |  |  | 20.81 | 36.85 |  |  | | |
|  | 60 |  | 31.11 |  |  | 26.78 | 43.66 |  |  | | |
|  | 70 |  | 34.32 |  |  | 24.87 | 40.88 |  |  | | |
|  | 80 |  | 38.62 |  |  | 36.17 | 34.5 |  |  | | |
|  | 90 |  | 34.62 |  |  | 30.45 | 30.36 |  |  | | |
|  | 100 |  | 45.99 |  |  | 33.93 | 20.58 |  |  | | |

The above data is plotted in a graph to analyze the performance of various random writes as below on the compute node as below:

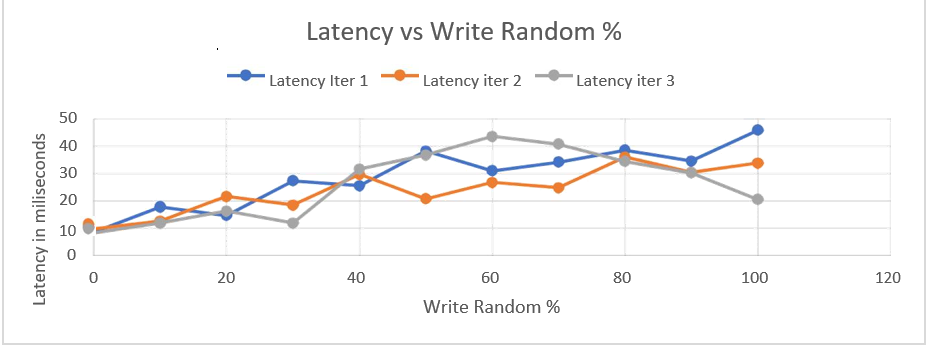


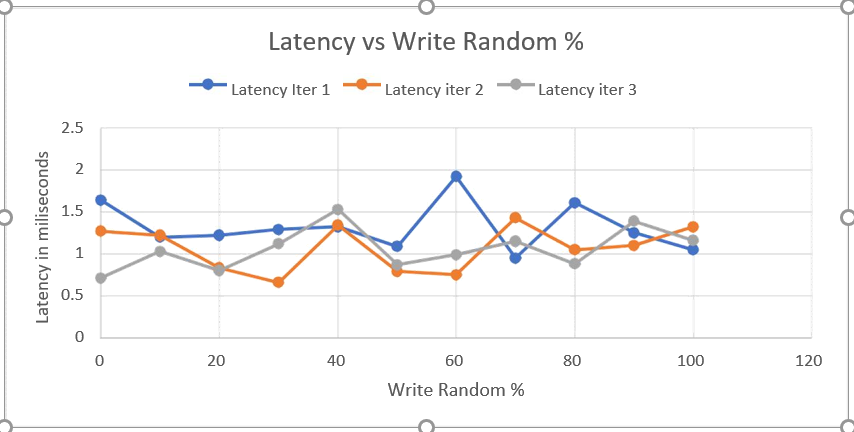
Figure 3

We can see that the charts for 2 out of 3 emphases are tending towards the worst scenario along these lines supporting our hypothesis of the random writes being the worst scenario of execution. The third cycle anyway could be a peculiarity because of foundation impacts.

We further contrast this information with the execution estimated at the master node as underneath:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Write random %** |  |  | |  | | |  | Latency |  |
|  |  | |  | | |  |  |
|  |  | **Iter 1** | |  | | |  | **Iter 2** | **Iter 3** |
| **0** | |  | 1.64 | |  | 1.27 | | | 0.71 |
| **10** |  | 1.2 | |  | | |  | 1.22 | 1.03 |
| **20** | |  | 1.22 | |  | 0.83 | | | 0.8 |
| **30** |  | 1.29 | |  | | |  | 0.66 | 1.12 |
| **40** | |  | 1.32 | |  | 1.34 | | | 1.53 |
| **50** |  | 1.09 | |  | | |  | 0.79 | 0.87 |
| **60** | |  | 1.92 | |  | 0.75 | | | 0.99 |
| **70** |  | 0.95 | |  | | |  | 1.43 | 1.15 |
| **80** | |  | 1.61 | |  | 1.05 | | | 0.88 |
| **90** | |  | 1.25 | |  | 1.1 | | | 1.39 |
| 100 |  | 1.05 | |  | | |  | 1.32 | 1.16 |

The graph for which would be as follows:

Figure 4

We can see that the latency on the master node is a lot lesser than that of the compute node since the SSD on the master is a high-level SSD which has a lot higher exhibition and costly. We can additionally see that the projection of the cycles on this diagram has no incrementation towards the worst scenario other than a couple of spikes in information which could be inconsistencies in data. We could state that the impacts of the arbitrariness on the higher end SSD do not impacts of the random write on the latency since there could be well-defined write management algorithms on the higher end master node as compared to the compute node SSD. Along these lines, we could see that the random write impacts the execution on the compute node and is the worst scenario while this does not have any significant bearing to the top of the line SSD utilized for the master node.

Further gathering the execution information dependent on the IOPS and bandwidth speed in Mb/sec measurements for the impacts of the random writes on execution. We could see that the random writes don't influence the bandwidth and the IOPS of the SSD as appeared in the charts beneath.

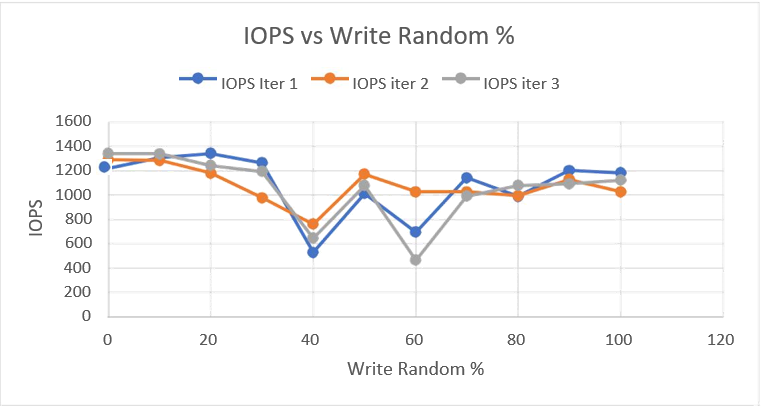


Figure 5

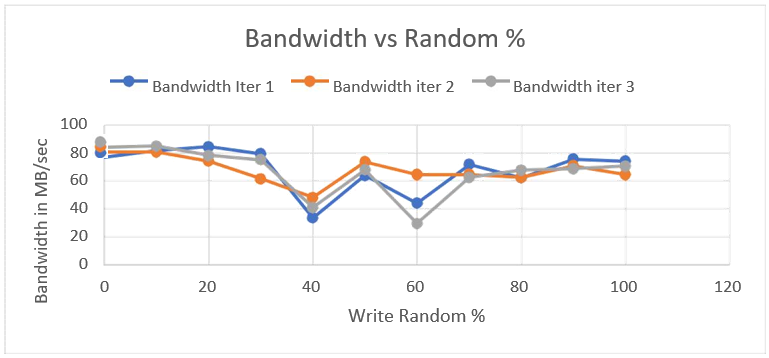


Figure 6

We can likewise watch a couple of irregularities or spikes of change in information which could be caused because of foundation activities or various clients using the resources.

We could likewise watch a comparable condition on the master node however with a lot higher bandwidth and IOPS as saw in the diagrams beneath:

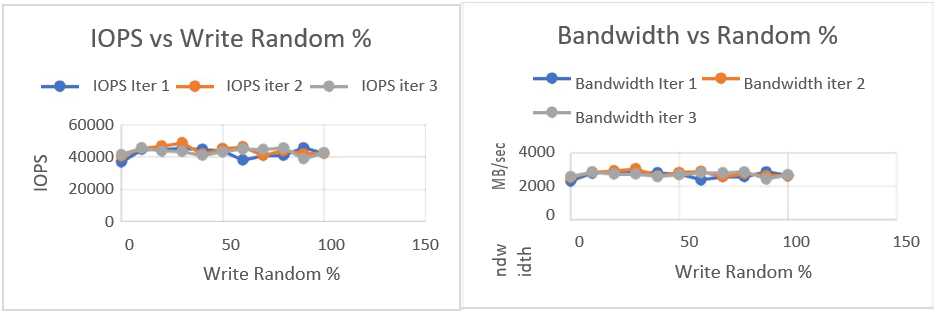


Figure 7

Considering every one of the perceptions above, we could state that the random writes can be considered as the worst scenario in the mid-level SSD, for example, the one utilized in the compute node. This factor anyway would not make a big deal about a distinction on the top of the line SSD which have a greatly improved IO the executives calculations to work productively even at the very least irregular compose conditions.

1. **Effects of increased read randomness on latency**

To check the impacts of read randomness on the drive, we run the accompanying direction on Heracles Master node server which has the high-end HGST SSD.

./fitness --file readtest --qdep 64 --wrsz 64 --rdsz 64 --rd\_stride 4 --wr\_stride 4 --wr 0 --rrnd 10

Aside from the –-rrnd10 parameter, we keep everything else the equivalent since we just need to comprehend the impacts of changing the read randomness. We note the values for 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100. In the wake of plotting the qualities, we plot the chart to read randomness versus latency.

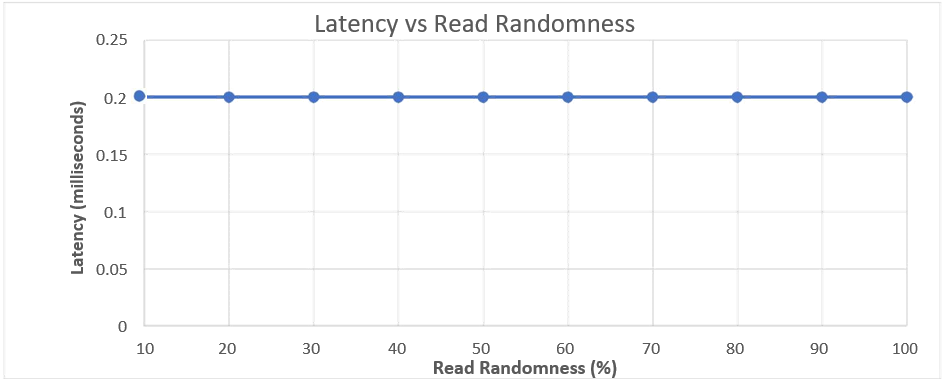
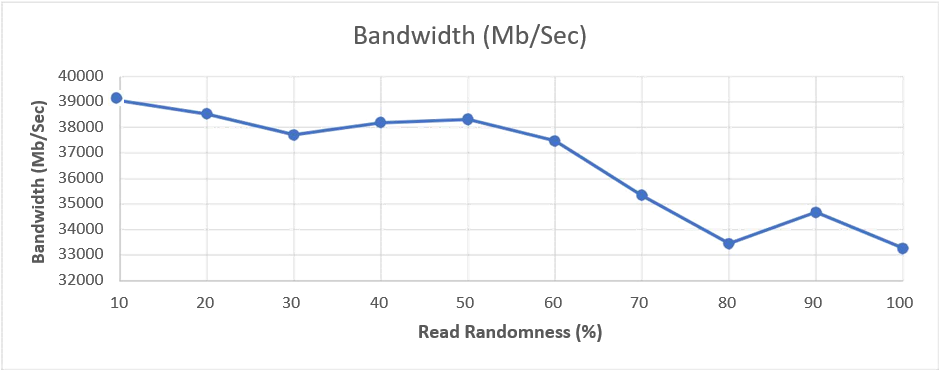


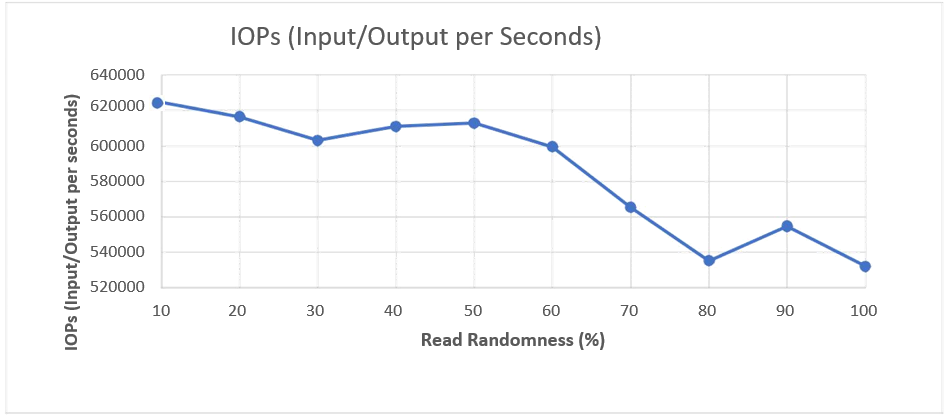
Figure 8

In the wake of watching the chart, we can see that the latency stays consistent in the event that we increment the read randomness. In this manner, we can say that increased read randomness has no impact on the latency.

We likewise plot the charts for the Read randomness versus Bandwidth and Read Randomness versus IOPs.



**Figure 9**



**Figure 10**

We see that as the read randomness builds, the bandwidth and IOPs diminishes. This can be on the grounds that the disk must access data from a wide range of areas. Bandwidth and IOPs dependably go connected at the hip and we can anticipate comparative outcomes for both.

Likewise, we are just adhering to testing the SSD on the master node. We were getting the comparable outcomes for the SSD present in the compute node so we can focus around the SSD at the master node for this case as it were.

1. **Read-write interdependence**

By and large, we could state that an I/O outstanding task at hand can be described by its access patterns (random or sequential), read/write proportion, request size and concurrency. We have considered the access patterns in all the conditions discussed so far and would now be considering the effects of read to write ratio on the performance of the SSD. The execution tests for the read and write interdependencies are observed in 8 separate cases as clarified beneath:

**Case 1:**

Command for 50:50 read to write ratio respectively with sequential read and write operations:

./fitness --file test --wr 50 --wrnd 0 –rrnd 0 --qdep 64 --wrsz 64 --rdsz 64 --wr\_stride 4 - rd\_stride 4 --iter 3

In this case, we observe the effects on performance considering the IO operations to be 50% read and 50% write operation with both read and write being sequential operations.

**Case 2:**

Command for 50:50 read to write ratio respectively with sequential read and random write:

./fitness --file test --wr 50 --wrnd 100 –rrnd 0 --qdep 64 --wrsz 64 --rdsz 64 --wr\_stride 4 - rd\_stride 4 --iter 3

In this case, we observe the effects on performance considering the IO operations to be 50% read and 50% write operation with sequential read operations and random write operations.

**Case 3**:

Command for 50:50 read to write ratio respectively with sequential write and random reads:

./fitness --file test --wr 50 --wrnd 0 –rrnd 100 --qdep 64 --wrsz 64 --rdsz 64 --wr\_stride 4 - rd\_stride 4 --iter 3

In this case, we observe the effects on performance considering the IO operations to be 50% read and 50% write operation with sequential write operations and random read operations.

**Case 4**:

Command for 50:50 read to write ratio respectively with random read and write operations:

./fitness --file test --wr 50 --wrnd 100 –rrnd 100 --qdep 64 --wrsz 64 --rdsz 64 --wr\_stride 4 -rd\_stride 4 --iter 3

In this case, we observe the effects on performance considering the IO operations to be 50% read and 50% write operation with both read and write operations being random.

**Case 5:**

Command for 75:25 read to write ratio respectively with sequential read and write operations:

./fitness --file test --wr 25 --wrnd 0 –rrnd 0 --qdep 64 --wrsz 64 --rdsz 64 --wr\_stride 4 - rd\_stride 4 --iter 3

In this case, we observe the effects on performance considering the IO operations to be 75% read and 25% write operation with both read and write operations being sequential.

**Case 6:**

Command for 75:25 read to write ratio respectively with random read and write operations:

./fitness --file test --wr 25 --wrnd 100 –rrnd 100 --qdep 64 --wrsz 64 --rdsz 64 --wr\_stride 4 -rd\_stride 4 --iter 3

In this case, we observe the effects on performance considering the IO operations to be 75% read and 25% write operation with both read and write operations being random

**Case 7:**

Command for 25:75 read to write ratio respectively with sequential read and write operations:

./fitness --file test --wr 75 --wrnd 0 –rrnd 0 --qdep 64 --wrsz 64 --rdsz 64 --wr\_stride 4 - rd\_stride 4 --iter 3

In this case, we observe the effects on performance considering the IO operations to be 25% read and 75% write operation with both read and write operations being sequential.

**Case 8:**

Command for 25:75 read to write ratio respectively with random read and write operations:

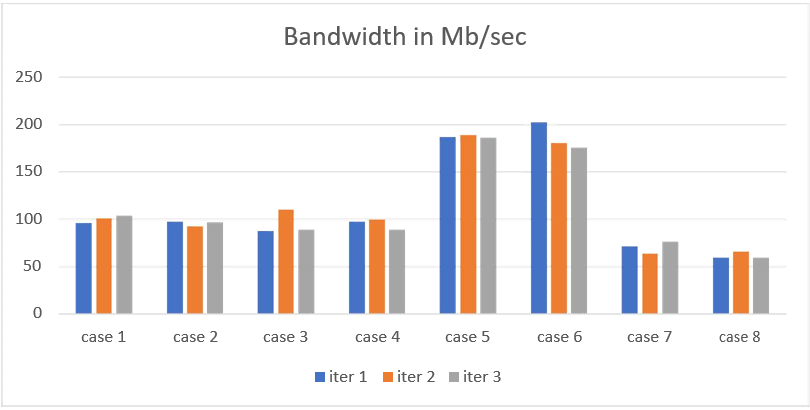
./fitness --file test --wr 75 --wrnd 100 –rrnd 100 --qdep 64 --wrsz 64 --rdsz 64 --wr\_stride 4 --rd\_stride 4 --iter 3

For this situation, we watch the consequences for execution believing the IO tasks to be 25% read and 75% write activity with both read and write activities being random. All the above cases can be abridges utilizing the beneath tables:

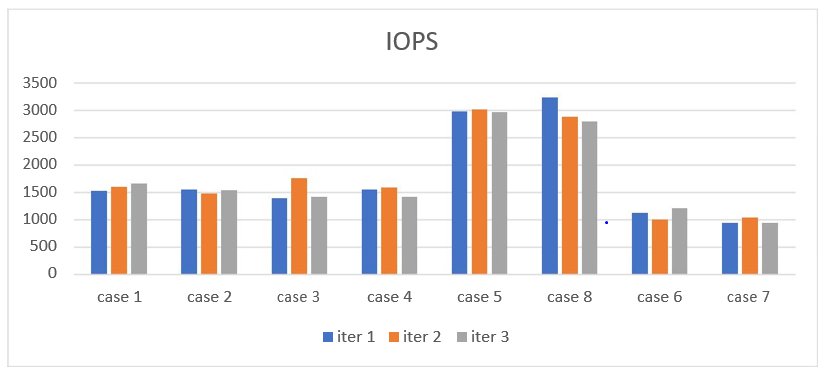
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **case 1** |  |  | **case 2** |  |  | **case 3** |  |  | **case 4** |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | 50- |  |  | 50- |  |  | 50- |  |  | 50- |  |
| **read:write** | 50 |  | **read:write** | 50 |  | **read:write** | 50 |  | **read:write** | 50 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **read** |  |  | **read** |  |  | **read** |  |  | **read** |  |  |
| **random** | N |  | **random** | N |  | **random** | Y |  | **random** | Y |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **write** |  |  | **write** |  |  | **write** |  |  | **write** |  |  |
| **random** | N |  | **random** | Y |  | **random** | N |  | **random** | Y |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **case 5** |  |  | **case 6** |  |  | **case 7** |  |  | **case 8** |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | 75- |  |  | 75- |  |  | 25- |  |  | 25- |  |
| **read:write** | 25 |  | **read:write** | 25 |  | **read:write** | 75 |  | **read:write** | 75 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **read** |  |  | **read** |  |  | **read** |  |  | **read** |  |  |
| **random** | N |  | **random** | Y |  | **random** | N |  | **random** | Y |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **write** |  |  | **write** |  |  | **write** |  |  | **write** |  |  |
| **random** | N |  | **random** | Y |  | **random** | N |  | **random** | Y |  |
|  |  |  |  |  |  |  |  |  |  |  |  |

The data collected on all the above cases are plotted together to observe their effects

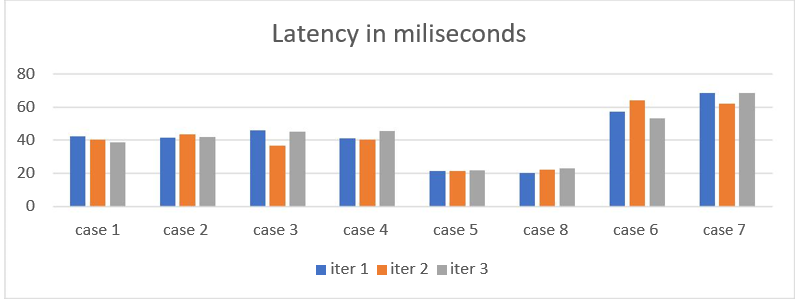
on the performance which can be monitored by the below graphs:



**Figure 11**



**Figure 12**



**Figure 13**

We can see that any cases including a bigger number of reads would beat outstanding tasks at hand with a bigger number of writes since the write activities on the SSD are of low execution than that of the read activities. An intriguing perception here anyway is that the randomness of the tasks does not appear to assume a bigger job on the execution as observed on the above charts. The diagrams for the read-write activities with proportion 50:50 with different randomness all appear to be at a vicinity to one another along these lines diminishing the effect of the randomness on the execution of the SSD. The quantity of read and write activities on the remaining task at hand appear to influence the general execution of the SSD significantly.

## Eagle Tree Simulator

Eagle tree is an SSD simulator which we used to analyse if we can use that for our studies. EagleTree is an open source simulator. It simulates an SSD, OS and applications utilizing it. It is thus possible to conduct large and complex design-space explorations, involving hundreds of experiments, in a tractable way [3].

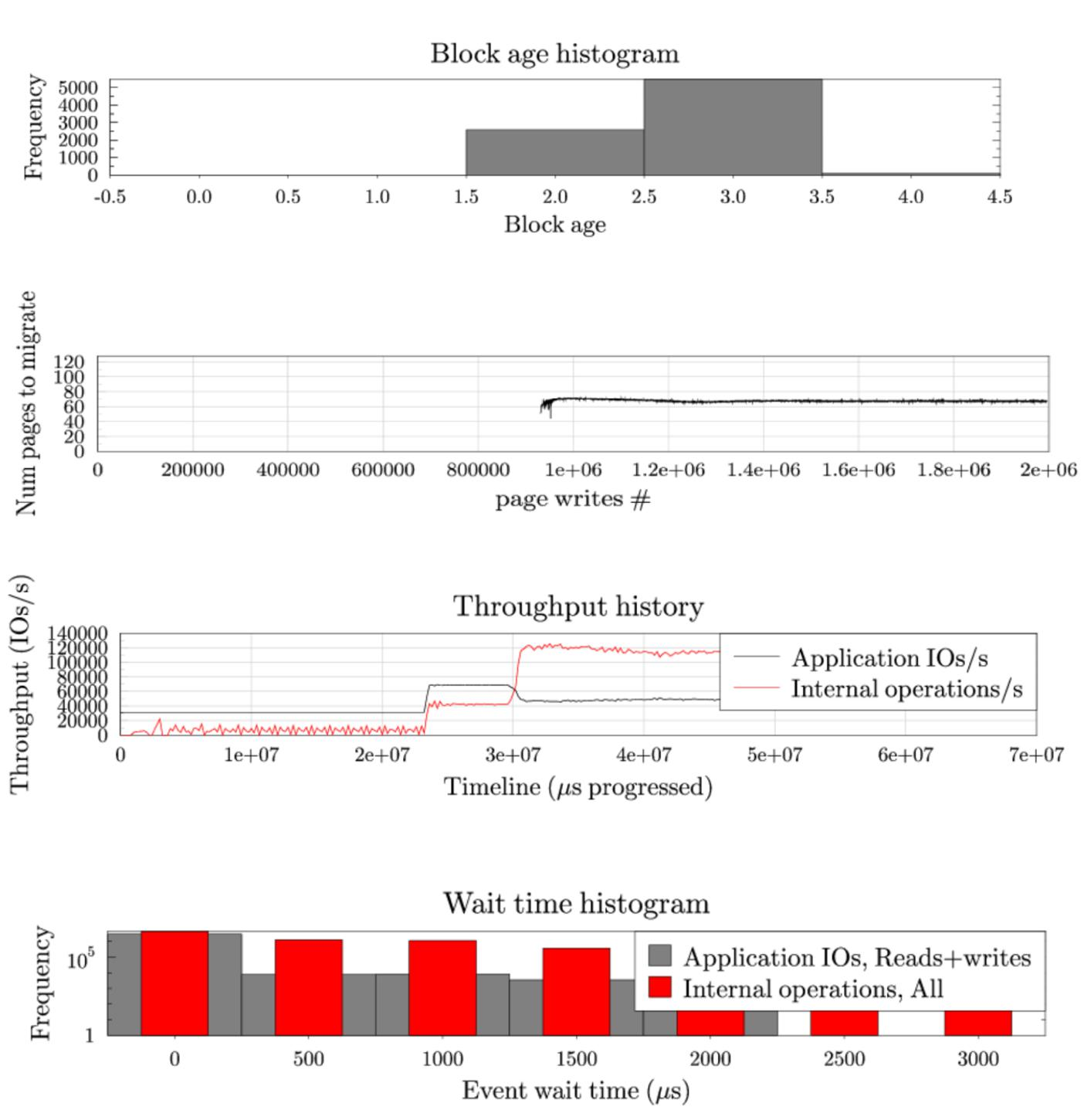
We can basically make a huge difference in the SSD arrangement in the config document and the run the examination to create the outcomes.

The output is created in demo/home/user/Desktop/EagleTree-

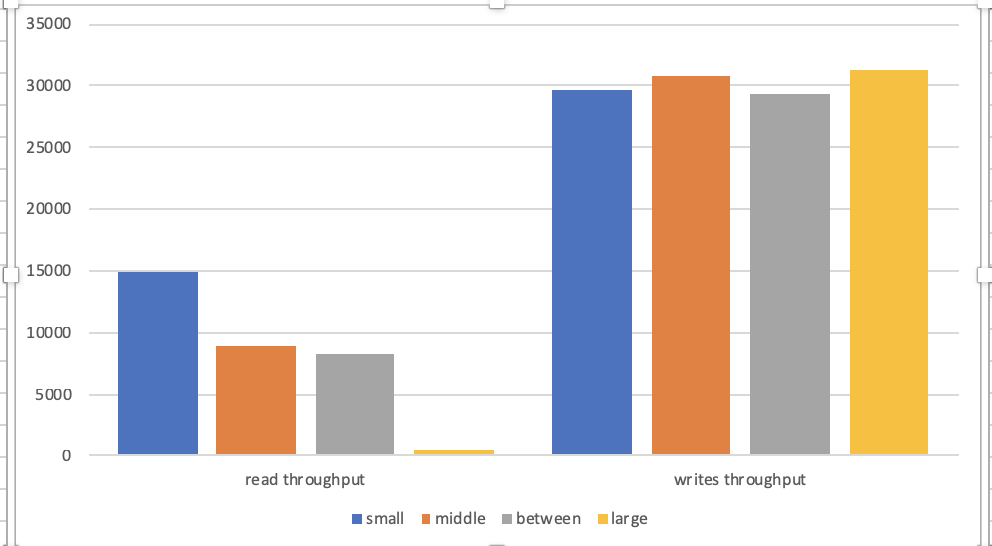
master/demo\_output/test/Global and is minimally jumbled. It creates a parcel of CSV records and png pictures (utilizing the gle library at first introduced).

A portion of the output records resemble the charts underneath. Aside from these records, we likewise get two documents called results.txt and configuration.txt. We can change the quantity of channels, blocks, basically everything in the configuration.txt document. Presently results for this setup is produced in the results.txt record.

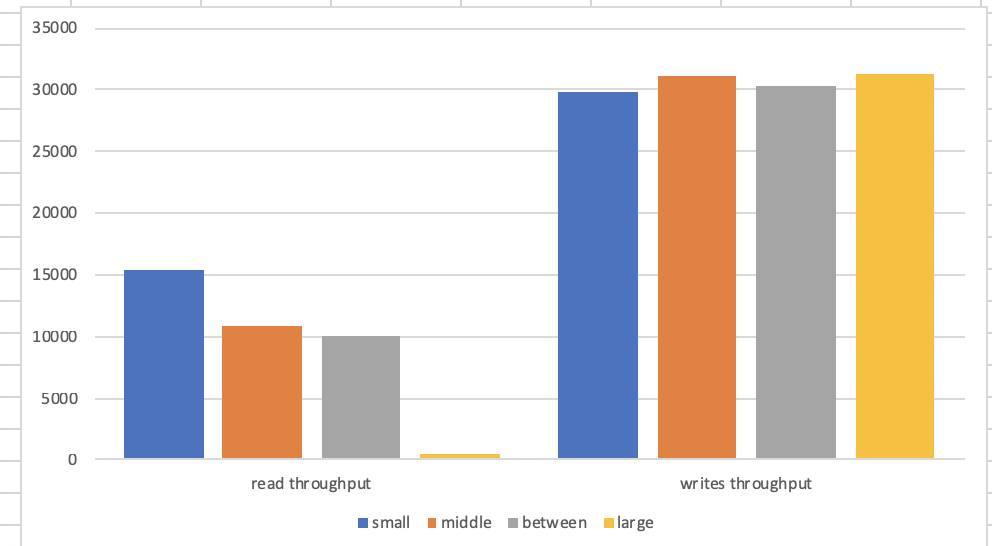
**Note:** Everytime EagleTree is run, this Output index is over-composed.



The eagle tree has following configurations which is possible to change in the simulator, like the ssd size, package size, plane size, block size and page size. In this project, we have changed different configuration to see how the ssd performances which evaluated by throughputs, here is result:



By seeing the result for the random w/r for different ssd configuration, we can see that with the growth of ssd itself, the read throughputs go down. And for the sequential w/r, the result came similar, which can be seen in the following figure.



Challenges

* At the underlying phase of our execution, we were not ready to run the Intel open storage toolbox because of the absence of progressing support.
* We attempted to work with various test systems including VSSIM and Eagle tree. Although Eagle tree was easy to install, VSSIM had OS reliance and was just working with Ubuntu 14.04.3. There is just a single activity (the installation process) which is utilized to produce the reads/writes for the disk. It is very limiting contrasted with EagleTree (which drove us to work with EagleTree as it were).

# Conclusion

We have tried almost all of the components in charge of the execution of a solid-state drive. A portion of these components have a striking impact, others not really.

* Is random write the worst scenario?

As indicated by our tests on Heracles, we can securely say that truly, random write is the worst scenario. We have enough experimentation and chart results to demonstrate our announcement.

* Effects of increased read randomness on latency.

We can say that Read randomness does not influence the latency of an SSD. At the end of the day, the latency stays consistent on random reads.

* Read-write interdependence.

As indicated by our execution, we find that read-write interdependence impacts the execution of the SSD. This can be supported up by the charts and the graphs in that individual part.

# References

|  |  |
| --- | --- |
| [1] | "Intel OST," [Online]. Available: https://sourceforge.net/projects/intel-iscsi/. |
| [2] | OST, "--Help command," [Online]. |
| [3] | E. simulator. [Online]. Available: https://github.com/ClydeProjects/EagleTree. |
| [4] | D. A. K. a. X. Z. F. Chen, " Understanding intrinsic characteristics and system implications of flash memory based solid state drives SIGMETRICS Perform.," vol. 37, pp. 181-192, 2009. |