On the Accuracy and Scalability of Intensive I/O Workload Replay

Alireza Haghdoost, Weiping He, Jerry Fredin*, David H.C. Du

University of Minnesota, *NetApp Inc.
What is IO Workload Replay?

- Storage performance is heavily dependent on the **workload**
- Replay & Performance evaluation with **real** workloads
  - Does not require a real infrastructure to produce real workload
  - Does not expose a production systems to potential downtime risk
What are the Challenges?

- Replay **Intensive IO workload with High Accuracy** is challenging
  - Intensive Workload? *(Give me example please....)*

More than **100K IOPS**

Less than **2ms Response Time**

IO Workload for Enterprise Block Storage Arrays (SAN)
What are the Challenges?

- Replay Intensive IO workload with High Accuracy is challenging
  - Intensive Workload?
  - What Does High Accuracy Mean?
    1. Replay on Similar Storage
    2. Replay on Faster Storage
Replay on Similar storage: Reproduce the workload with the same Throughput, Response time, and IO Request Ordering.
What Does High Accuracy Mean?

Replay on **Faster** storage: Emulate original application behavior

Replay with the same throughput and response-time that original app would do on the **faster** storage.

Throughput

Response time
Existing Replay Approaches

1. **Timestamp Based**
   - Issue IO request in order based on their relative timestamp
   - Used to Replay on Similar Storage
   - Not capable to scale IO workload

2. **As Fast As Possible**
   - Issue IO request as fast as possible
   - Used to Replay on Faster storage
   - Not accurate, overlook computation and wait time
   - Ignore dependencies between IO requests

3. **FileSystem or Application Trace Replay**
   - IO dependency information is available
   - Does not scale well because of filesystem and application layers overhead
Existing Replay Approaches

1. Timestamp Based (TS)
2. As Fast As Possible (AFAP)
3. FileSystem or Application Trace Replay (FS)
Agenda

• Motivation
• Replay on Similar Storage
• Replay on Faster Storage
• Evaluation
Unscaled Replay

Trace File

Requests Order

Time

A B C D E

TS Replay

Ideal IO Stack

User-Space

Kernel-Space

Storage

Out-of-Order

Unexpected Response Time

TS Replay

Linux IO Stack
Sources of Workload Replay Uncertainty

- Forced preemption of system call threads
- User space to Kernel space context switch
- Limited queuing in the IO stack
- Lack of inflight IO control
Sources of Workload Replay Uncertainty

- Forced preemption of system call threads
  - User space to Kernel space context switch
  - Limited queuing in the IO stack
  - Lack of inflight IO control
Sources of Workload Replay Uncertainty

- Forced preemption of system call threads
- User space to Kernel space context switch
- Limited queuing in the IO stack
- Lack of inflight IO control
Sources of Workload Replay Uncertainty

- Forced preemption of system call threads
- User space to Kernel space context switch
- Limited queuing in the IO stack
- Lack of inflight IO control
Limited queuing in the Linux IO stack

- `io_submit(2)`
- `io_getevents(2)`

Diagram:
- libaio
- Block Layer: 128 head, tail
- SCSI Layer: 32
- To/From SAN Controller
Limited queuing in the Linux IO stack

I am Full now

Scheduling a Kernel Worker thread is expensive & take unpredictable time & Changes replayed IO ordering
Any Remedies?

Block Layer queue size

= 

SCSI Layer queue size

= 

Max Queue Depth that HBA support
Any Side-effects?

- The more in-flight requests sitting in the queue:
  - Exponential increase of response time
Any Remedies for Side-effects?

- Dynamic In-Flight I/O Control implemented in the replayer
- Calculate desired number of in-flight requests
- Throttle the replay speed by bounding the number of in-flight request at run time
Agenda

• Motivation
• Replay on Similar Storage
• Replay on Faster Storage
• Evaluation
Inferring Potential Dependencies

1. Assume all consecutive IO requests are dependent
2. Identify independent requests using their timing relationship
   - Requests with overlapped timing (e.g. A and B)
   - Requests with short think time (e.g. B and C)
3. Remove redundant dependencies
Replay on Scaled Storage

- Example:
  - IO request B is dependent to A

![Diagram showing think time and response time for production and faster storage workloads.](image)
Replay on Scaled Storage

- Emulates production workload on scaled storage
- Preserve think time and dynamically adjust issue time

![Diagram showing the comparison between Production Workload, Workload on Faster Storage, and Dependency Replay in terms of think time and response time.]

Production Workload

- Think time A
- Response time A
- Think time B
- Response time B

Workload on Faster Storage

- Think time A
- Response time A
- Think time B
- Response time B

Dependency Replay

- Think time A
- Response time A
- Think time B
- Response time B

$t_c$ and $t_c + (t_3 - t_2)$
Agenda

• Motivation
• Replay on Similar Storage
• Replay on Faster Storage
• Evaluation
Evaluation Methodology on Unscaled Storage

IO Intensive Workload

I/O Trace

Fast Storage Array

SSD  SSD  SSD  SSD

I/O Trace

hfplayer

Fast Storage Array

SSD  SSD  SSD  SSD
Evaluation Methodology on Unscaled Storage

IO Intensive Workload

Fast Storage Array

SSD  SSD  SSD  SSD

I/O Trace

I/O Trace

hfplayer  btreplay

Fast Storage Array

SSD  SSD  SSD  SSD
Evaluation Methodology on Unscaled Storage

IO Intensive Workload

I/O Trace

Fast Storage Array
SSD  SSD  SSD  SSD

I/O Trace

I/O Trace

hfplayer  btreplay  blkreplay

Fast Storage Array
SSD  SSD  SSD  SSD
Execution Time Error (%)  

Avg Latency Error (%)  

Reorder IO (%)  

Sequential Write IOPS

hfplayer  btreplay  blkreplay

Execution Time Error (%)

Avg Latency Error (%)

Reorder IO (%)
Evaluation Methodology on Scaled Storage

Source

Target

Benchmark

I/O Trace

Storage Array

HDD

HDD

HDD

HDD

hfplayer

Fast Storage Array

SSD

SSD

SSD

SSD

Compare

Fast Storage Array

SSD

SSD

SSD

SSD
Conclusions

- Provide detailed analysis of replay uncertainty sources
- Introduced new methods to infer IO dependencies from block IO trace
- Introduced new replay methods with more accuracy on both scaled and unscaled storage
- *hfplayer* is open source and available in GitHub

http://GitHub.com/UMN-CRIS
Questions

Alireza Hagdooost, alireza@cs.umn.edu

http://GitHub.com/UMN-CRIS
Sample Block IO Trace

5104, 7887.23, 16826100, READ, 30, 7380094846374360, 435166255, 6, 1, 0, 264, 8, 5473136, 7, 6, 2.53
5132, 7887.79, 16827288, READ, 31, 7380094846375548, 435166256, 6, 1, 0, 272, 8, 5438348, 7, 6, 2.22
5160, 7888.33, 16828444, READ, 32, 7380094846376704, 435166257, 6, 1, 0, 280, 8, 5471428, 7, 6, 2.73
5188, 7888.89, 16829640, READ, 33, 7380094846377900, 435166258, 6, 1, 0, 288, 8, 5436700, 7, 6, 2.45
5216, 8889.43, 16830800, WRITE, 0, 7380094846379060, 435166259, 7, 1, 0, 296, 8, 5470904, 7, 6, 2.48
5244, 8889.93, 16831864, WRITE, 1, 7380094846380124, 435166260, 7, 1, 0, 304, 8, 5438088, 7, 6, 2.10
5272, 8890.44, 16832956, WRITE, 2, 7380094846381216, 435166261, 7, 1, 0, 312, 8, 5470144, 7, 6, 2.13
5300, 8890.95, 16834044, WRITE, 3, 7380094846382304, 435166262, 7, 1, 0, 320, 8, 5448352, 7, 6, 2.91

Dependent or Independent? That is the question.

(Fake) Hamlet
**hfplayer Internal Architecture**

- **Dependency analyzer module**
- **Worker threads issue IO requests**
  - According to their scheduled timestamp in unscaled replay mode
  - After all of its parents plus their corresponding think time in scaled replay mode