Spool : Reliable Virtualized NVMe Storage Systems in Public Cloud Infrastructure

†‡Shuai Xue, †‡Shang Zhao, †‡Quan Chen, ‡Gang Deng, ‡Zheng Liu, ‡Jie Zhang, ‡Zhuo Song
‡Tao Ma, ‡Yong Yang, ‡Yanbo Zhou, ‡Keqiang Niu, ‡Sijie Sun, †Minyi Guo

†Dept. of Computer Science and Engineering, SJTU

‡Alibaba Cloud

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

目录 Contents

1 Introduction and Motivation

2 Design of Spool

3 Spool key ideas

4 Evaluation

Introduction

HDD	SATA NAND SSD	NVMe NAND SSD	NVMe V-NAND SSD
LOW PERFORMANCE	AFFORDABLE PERFORMANCE	HIGH PERFORMANCE	EXPTRME PERFORMANCE
500 IOPS	25 K IOPS	400 K IOPS	1,500 K IOPS
2 ms LATENCY	100 us LATENCY	100 us LATENCY	10 us LATENCY

With the development of storage hardware, **software has become the performance bottleneck.**

Introduction

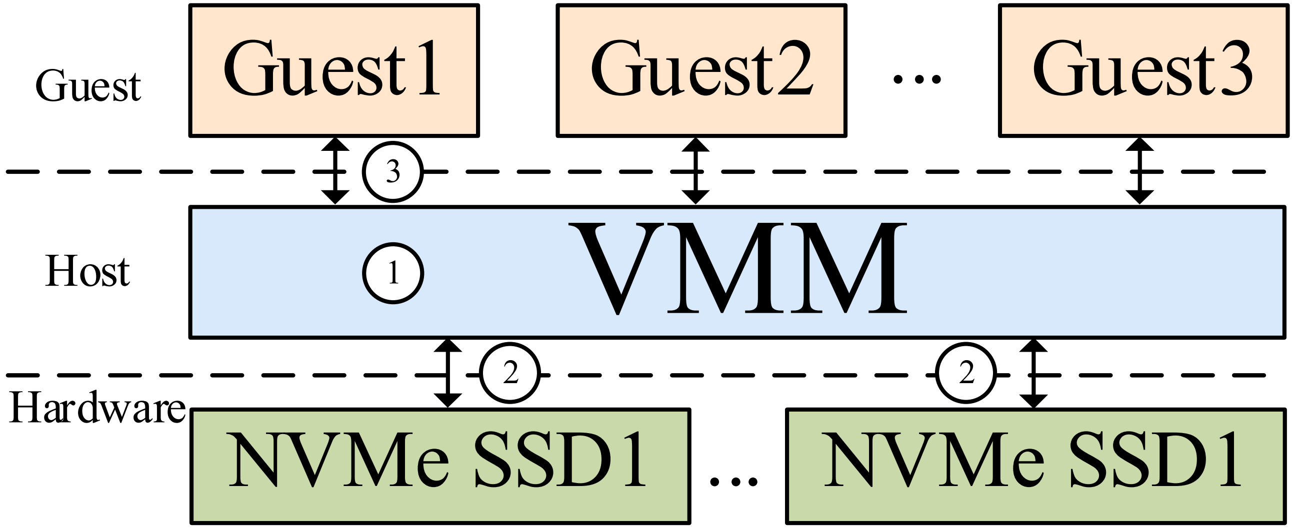
The local NVMe SSD-based instance storage provided fo

- Amazon EC2 I3 series
- Azure Lsv2 series
- Alibaba ECS I2 series

The local NVMe SSD-based instance storage optimized for:

- low latency
- high throughput
- high IOPS
- low cost

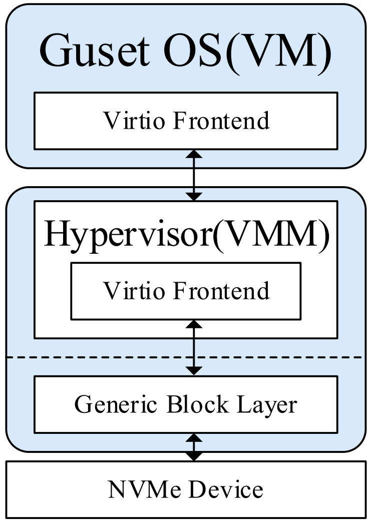
Introduction



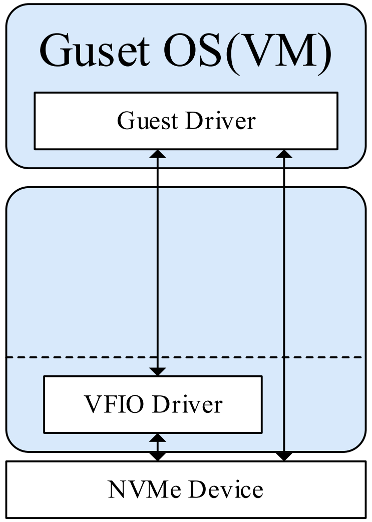
High reliability is the most important and challenging problem:

- restarting the virtualization system
- removing the failed device
- performing the upgrade

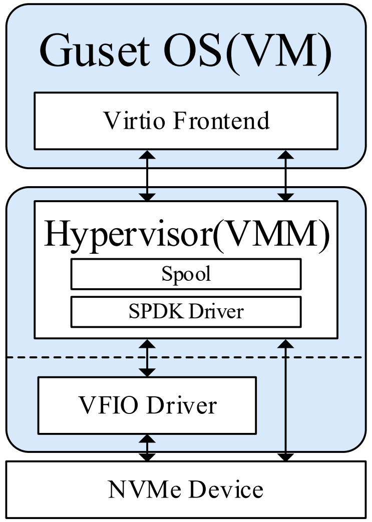
Introduction



Virtio



PCI passthrough



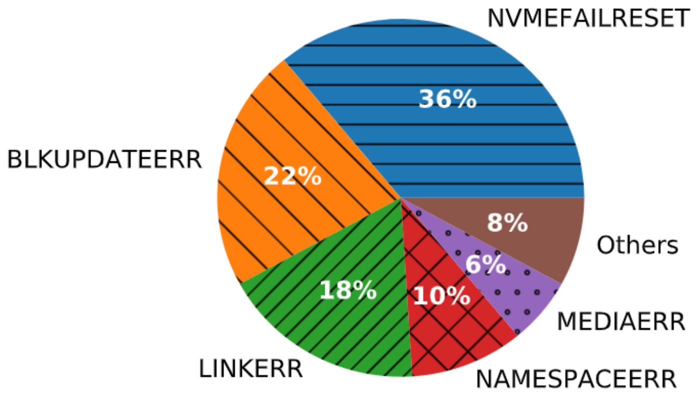
Spool based on SPDK

Spool is proposed based on the SPDK NVMe driver but **focuses on the reliability** of the virtualized storage system.

Motivation

Unnecessary Data Loss: reset device controller

- For Azure, a device failure results in the entire machine being taken offline for repair.
- For SPDK, the administrator directly replaces the failed device through hot-plug.
- Only 6% of the hardware failures are due to real media errors.

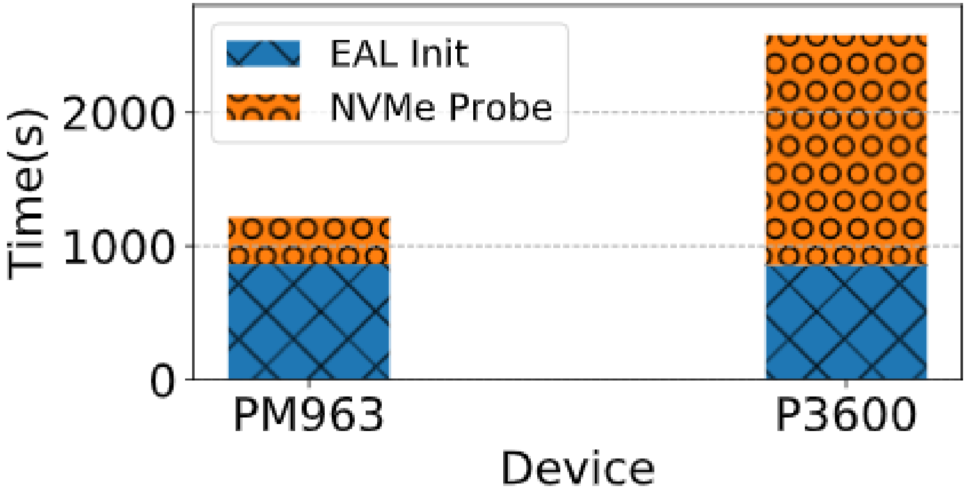


The current failure recovery method results in significant unnecessary data loss.

Motivation

Poor Availability

- VM live migration is too costly.
- The downtime for SPDK restart is up to 1,200 ms.



The long downtime hurts the availability of the I/O virtualization system.

目录 Contents

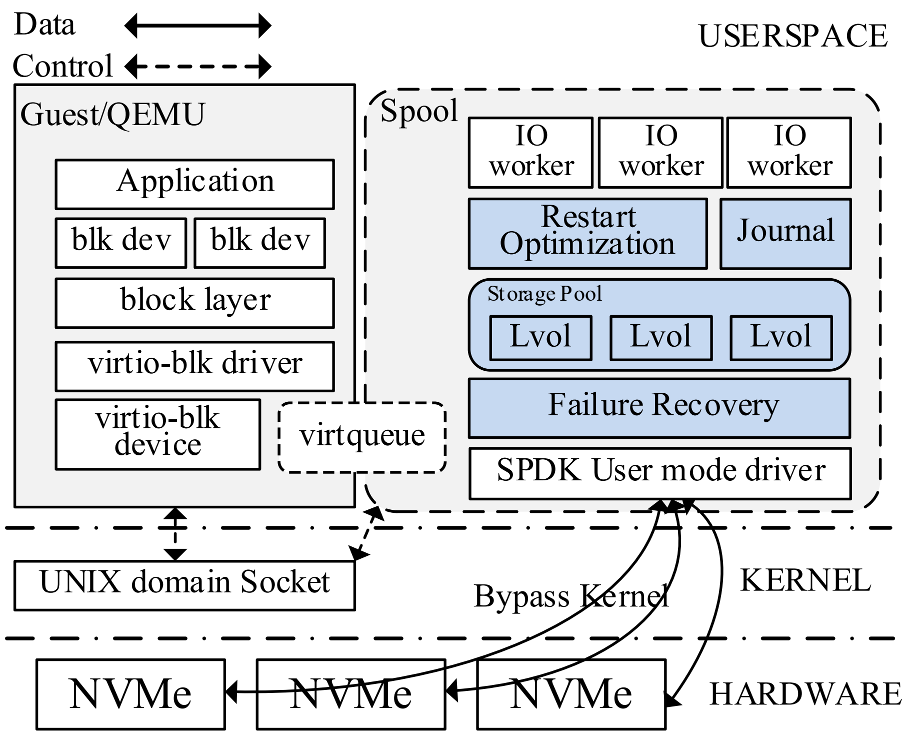
1 Introduction and Motivation

2 Design of Spool

3 Spool key ideas

4 Evaluation

Design of Amoeba



Spool is comprised of:

- A cross-process journal: records each I/O request and its status to ensure data consistency.
- A fast restart component: records the runtime data structures of the current Spool process reduce the downtime.
- A failure recovery component: diagnoses the device failure type online to minimize unnecessary disk replacement.

目录 Contents

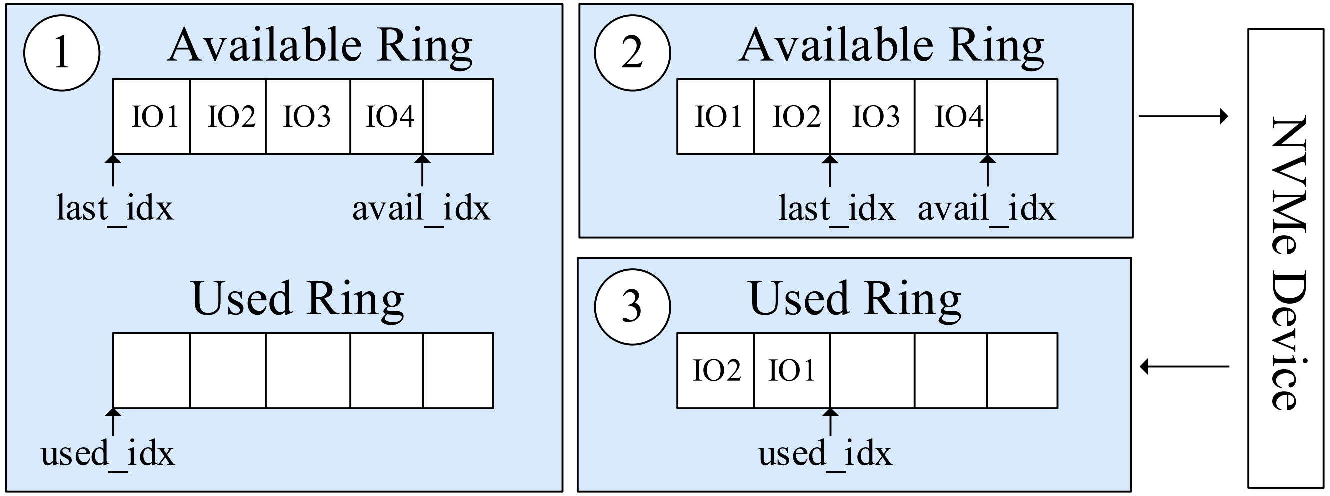
1 Introduction and Motivation

2 Design of Spool

3 Spool key ideas

4 Evaluation

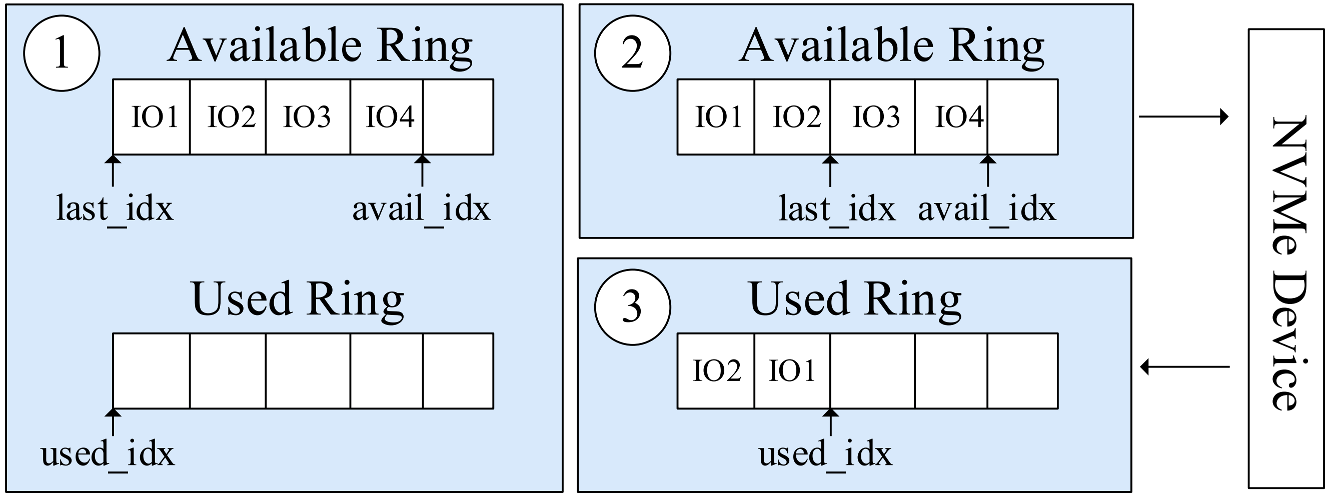
Reliable Cross-Process Journal



The I/O requests are processed in a *producer-consumer* model:

1. The guest driver places the head index of descriptor chain into the next ring entry of the available ring, and *avail_idx* of the available ring is increased.
2. The backend running in the host obtains several head indexes of the pending I/O requests in the available ring, increases *last_idx* of the available ring and submits the I/O requests to hardware driver.
3. Once a request is completed, the backend places the head index of the completed request into the used ring and notifies the guest.

Reliable Cross-Process Journal



Reliable problem:

- The backend obtains two I/O requests, IO1 and IO2.
- Then, the last_idx is incremented from IO1 to IO3 in the available ring.
- If the storage virtualization system restarts at this moment, the last available index will be lost.

Spool persists:

- last_idx
- the head index of each request
- the states of each request:INFLIGHT, DONE,or NONE.

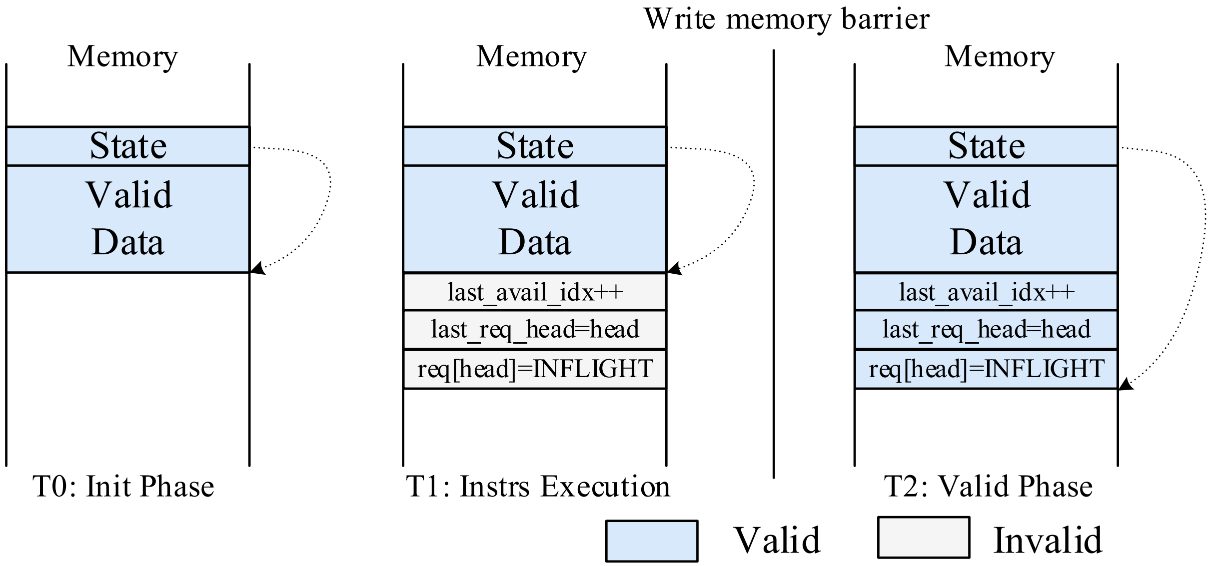
Reliable Cross-Process Journal

The challenge to ensure the consistency of the journal is to:

- guarantee that instructions to increase *last_idx* and change the request's status are executed in an atomic manner.

A multiple-instruction transaction model

- In T0, we make a copy of the variable to be modified.
- In T1, the transaction will be in the START state, and the variables are modified.
- After all the variables modified completely, the transaction will be in the FINISHED state in T2.

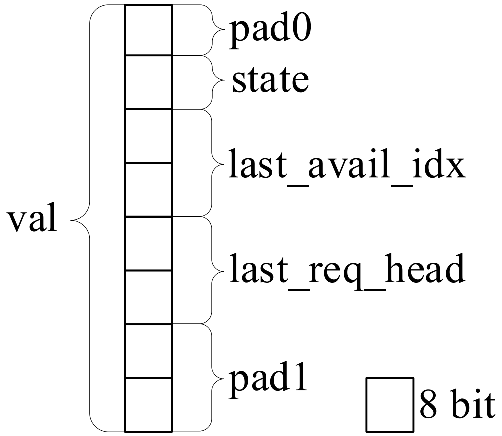


Reliable Cross-Process Journal

An auxiliary data structure:

- It is a valuable trick to efficiently maintain journal consistency to eliminate the overhead of making a copy in T0.
- The state, last available index, and head index of the related request are padding to 64 bits and a union memory block with a 64-bit value.
- The three records are updated within one instruction.

```
union atomic_aux {
    struct {
        uint8_t       pad0;
        uint8_t       state;
        uint16_t      last_avail_idx;
        uint16_t      last_req_head;
        uint16_t      pad1;
    };
    uint64_t          val;
};
```



Reliable Cross-Process Journal

Algorithm 1 Algorithm of cross-process journal

```
Require: head1: The head index of request in the available ring;
Require: head2: The head index of completed request from the driver;
Require: req[]: A ring queue to mark each I/O reqeust in the journal;
Require: aux: A temporary union variable to record multiple variables;
 1: poll head1 from the available ring;
 2: aux.state = START;
 3: aux.last_idx = journal->last_idx+1;
 4: aux.last_req_head = head1;
 5: *(volatile uint64_t *)&journal->val = *(volatile uint64_t *)&aux.val;
 6: req[head1] = INFLIGHT;
 7: journal->state = FINISHED;
 8: submit I/O request to driver;
 9: poll head2 completion;
10: journal->usd_idx++;
11: req[head2] = DONE;
12: put head2 to the used ring, may goto 10 or 13;
13: update used_index of the used vring with usd_idx of journal;
14: req[head2] = NONE;
```

Every step Spool takes in algorithm 1 is likely to restart for upgrade.

Reliable Cross-Process Journal

The recovery algorithm:

- The new Spool process before the restart only needs to check the state and decide whether to redo the transactions.
- The states of IO request is repaired based on used index of vring and the last used index in the journal.

Algorithm 2 Algorithm for recovering I/O requests

```
if (state == START) {
        req[last_get_req_head] = INFLIGHT;
        state = FINISHED;
}
jstate = (last_used_idx == used_idx) ? NONE : INFLIGHT
change all requests with done status to jstate;

last_used_idx = used_idx;
submit all requests marked as INFLIGHT;
```

Optimizing Spool Restart

Start stage 1: Init EAL

- Obtaining memory layout information: 70.9% of the total time.
- The runtime configurations and memory layout information can be reused.

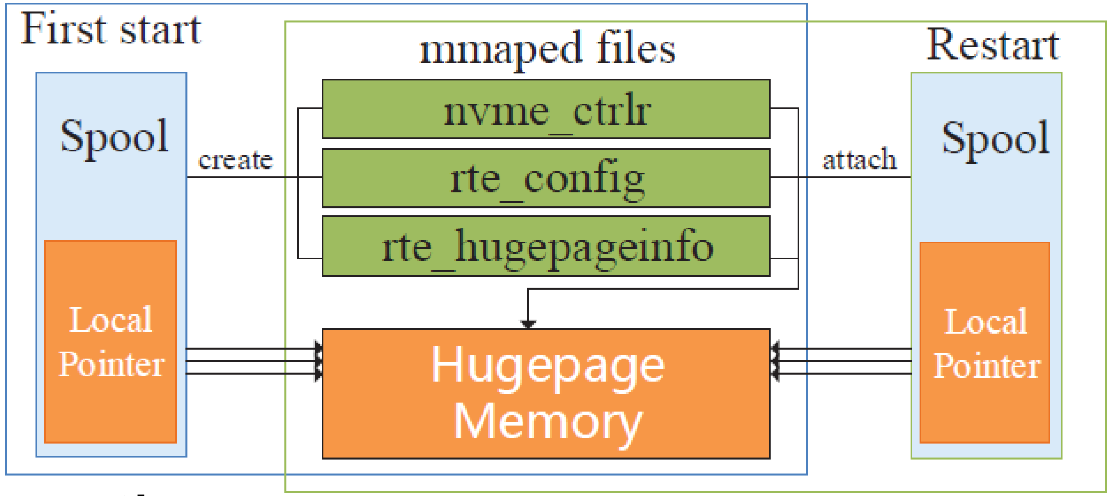
Start stage 2: Probe device

- Resetting the controller of NVMe devices: 90% of the total time.
- The controller information can be reused.

Optimizing Spool Restart

Reusing Stable Configurations

- Global runtime configurations.
- Memory layout information.



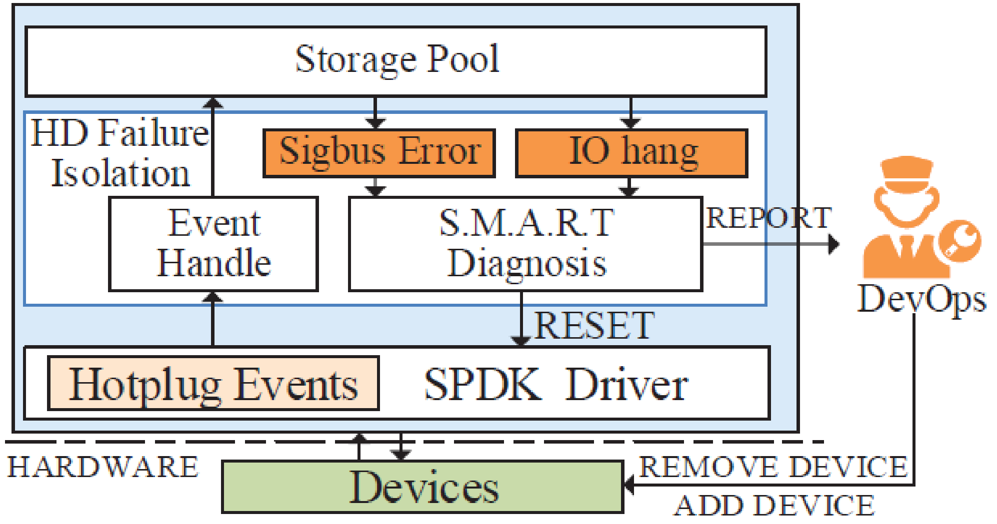
Skipping Controller

- NVMe device controller-related information.
- Gracefully terminate: SIGTERM and SIGINT signals.

Hardware Fault Diagnosis and Processing

Handling Hardware Failures

- A device failures or hot-remove cause process to crash.
- A SIGBUS handler is registered.



Failure Model

- Based on S.M.A.R.T. diagnosis
- Hardware media error: hot-plug a new device.
- Other hardware errors: reset the controller.

目录 Contents

1 Introduction and Motivation

2 Design of Spool

3 Spool key ideas

4 Evaluation

Experimental configuration

Table 1: Experimental configuration

Host configuration	
CPU & Memory	2x E5-2682v4 @2.5GHz; 128GB DDR4 Memory
NVMe devices	2 Samsung PM963 3.84TB SSDs
OS info	CentOS 7 (kernel verison 3.10.327)
Guest OS configuration	
CPU & Memory	4 vCPU; 8 GB
OS info	CentOS 7 (kernel verison 3.10.327)

Experimental configuration

Table 2: FIO test cases

Tested metrics	Test cases	FIO Configuration (bs, rw, iodepth, numjobs)
Bandwidth	Read	(128K, read, 128, 1)
	Write	(128K, write, 128, 1)
IOPS	Randread	(4K, randread, 32, 4)
	Mixread	(4K, randread 70%, 32, 4)
	Mixwrite	(4K, randwrite 30%, 32, 4)
	Randwrite	(4K, randwrite, 32, 4)
Average Latency	Randread	(4K, randread, 1, 1)
	Randwrite	(4K, randwrite, 1, 1)
	Read	(4K, read, 1, 1)
	Write	(4K, write, 1, 1)

Performance: Bandwidth, IOPS,Average Latency

Reliability of Handling Hardware Failure

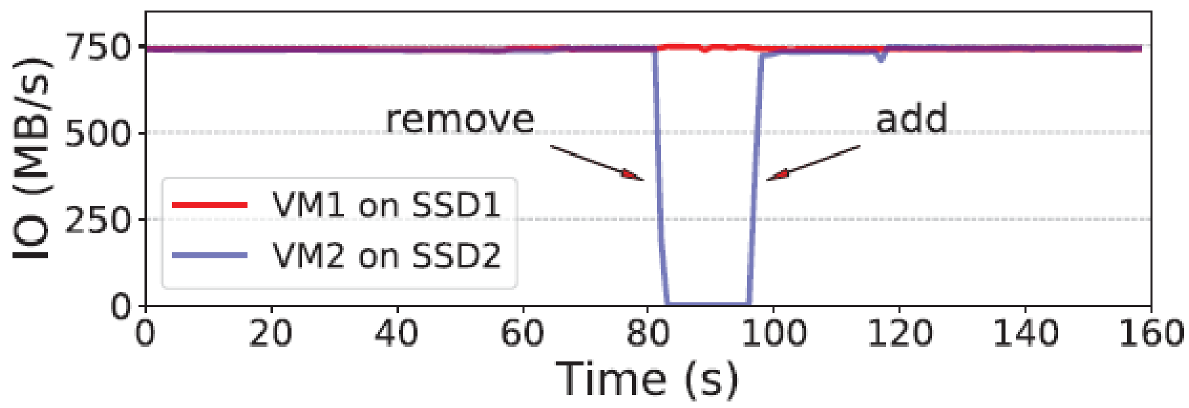


Figure 12: Handling hardware failure with Spool.

- SSD2 is hot-removed and hot-pluged.
- The storage service for VM2 is back online automatically.
- The storage service for VM1 is not affected.

Reliability of Handling Random Upgrades

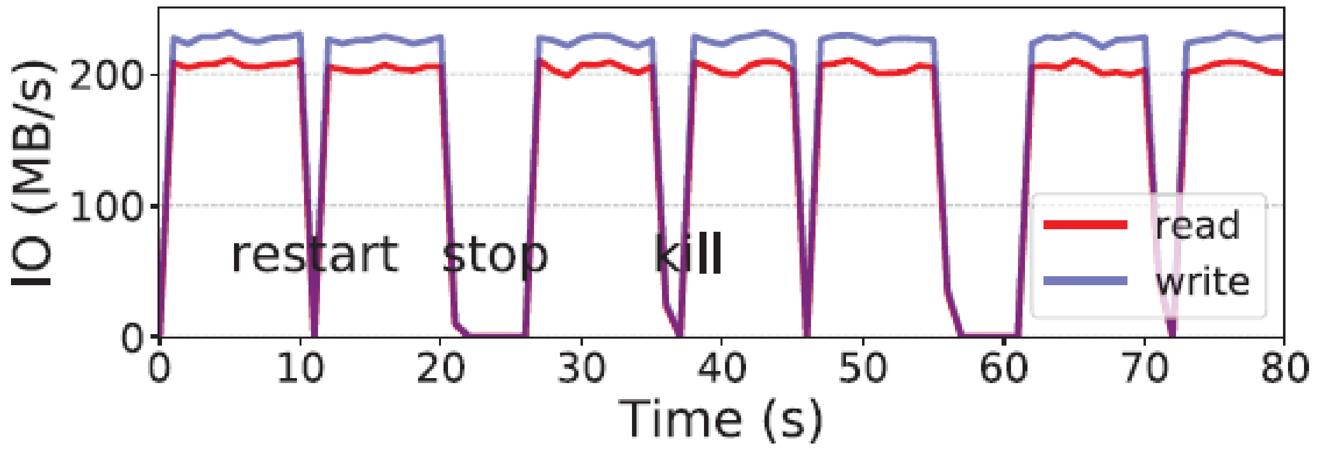


Figure 13: Data consistency at live upgrade with Spool.

- The file contents is verified with FIO on a Guest VM.
- Spool can guarantee data consistency during upgrades.

Reducing Restart Time

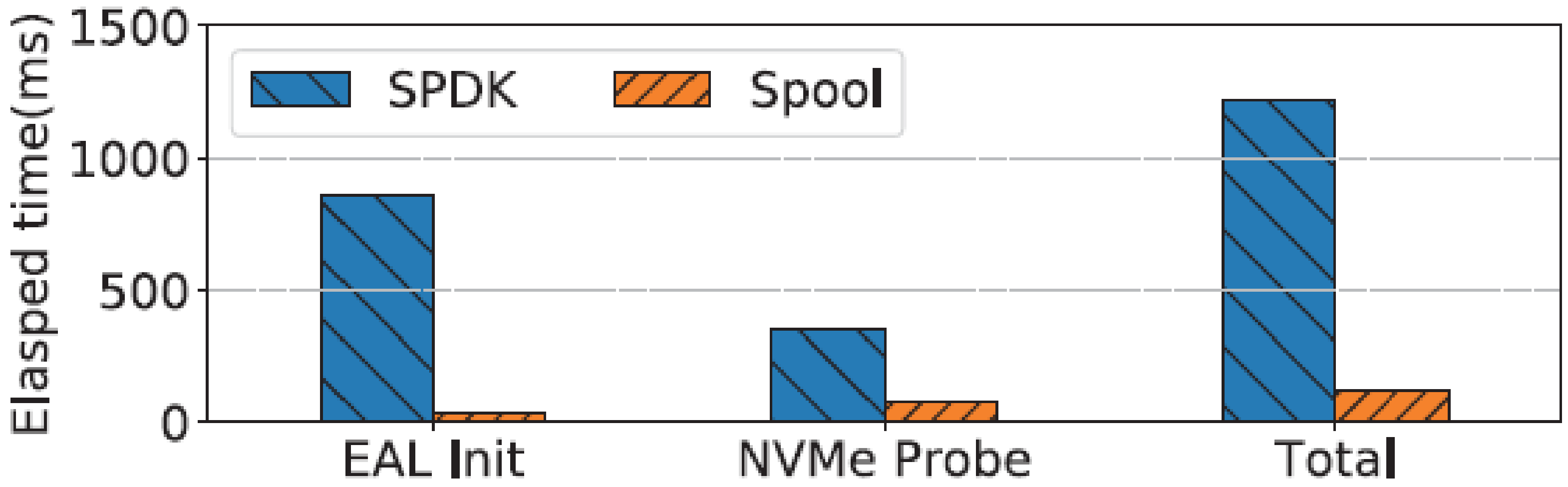
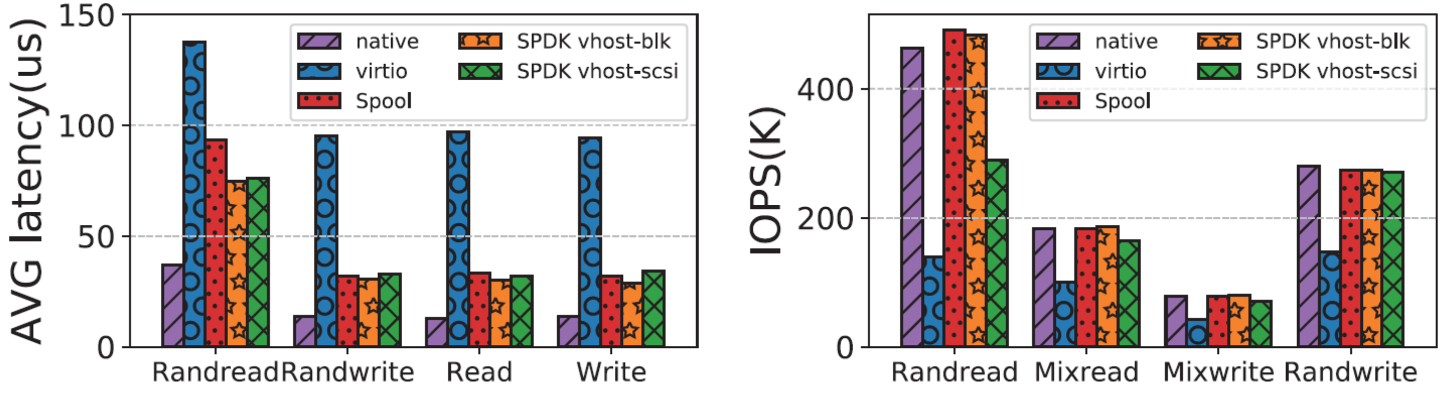


Figure 14: Restart times of Spool and SPDK.

➢ Spool reduces the total restart time from 1,218 ms to 115 ms.

I/O Performance of Spool

Case 1: Single VM Performance



(a) Average latency (b) IOPS

Figure 15: Average data access latency and IOPS of an NVMe SSD when it is used by a single VM.

- Spool achieves similar performance to SPDK.

I/O Performance of Spool

Case 2: Scaling to Multiple VMs

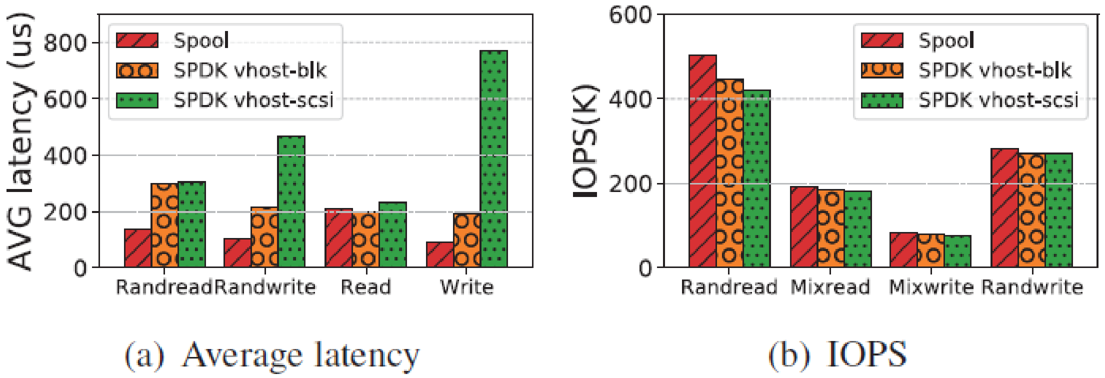


Figure 16: Average data access latency and IOPS of an NVMe SSD when it is shared by multiple VMs.

- Spool improves the IOPS of Randread by 13% compared with SPDK vhost-blk.
- Spool reduces the average data access latency of Randread by 54% compared with SPDK vhost-blk.

Overhead of the Cross-Process Journal

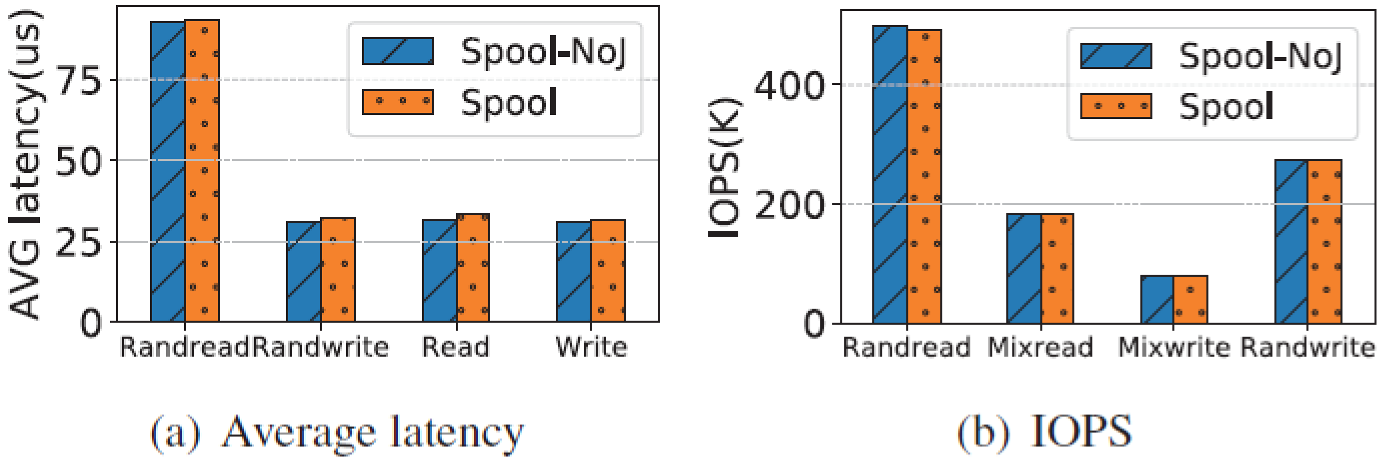


Figure 17: Overhead of the cross-process journal.

- Spool increases the average data access latency no more than 3%.
- And Spool reduces the IOPS by less than 0.76%

Deployment on an In-production Cloud

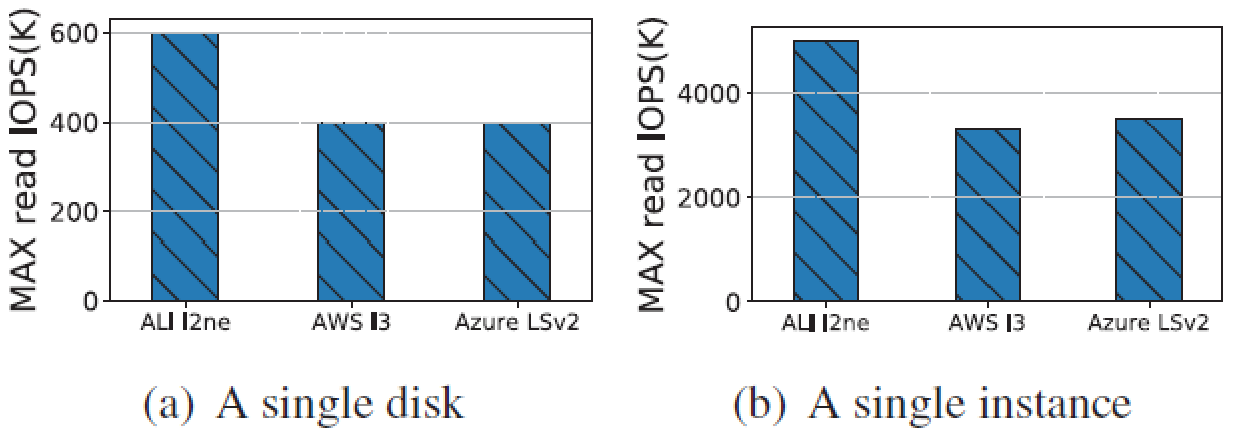


Figure 18: Maximum read IOPS compared with AWS and Azure.

- The maximum IOPS of a single disk is 50% higher.
- The maximum IOPS of a largest specification instance is 51% higher.

Thanks for listening！

Question?

xueshuai@sjtu.edu.cn or
chen-quan@cs.sjtu.edu.cn