/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* The Instructions for the AppleS Artifact Evaluation \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\* Ning Li \*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\* ning.li@uta.edu \*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\* 02/01/2022 \*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

1. Abstract

AppleS aims to improve database scalability by delivering the right amount and pattern of user parallel I/O requests to the database system under excessive user parallelism, aligning with the concurrency supported by the database and its underlying I/O stack. In doing so, AppleS improves user-level I/O performance in terms of user-level I/O fairness, throughput and latency stability. Implemented as a user-space module based on system call interception, AppleS is compatible with and portable to different types/versions of databases, different versions of OS kernels and their resource management tools, e.g., Cgroups.

2. Description & Requirements

This section provides all the information necessary to recreate the same experimental setup used in this paper to run the AppleS artifact.

2.1 Hardware dependencies

There are no hardware dependencies. A.3.1 provides guidance to constructing the physical testbed in such a way that the expected excessive user parallelism, which is required by the evaluation of the AppleS artifact, can be effectively reproduced under the given benchmarks.

2.2 Software dependencies

AppleS is implemented based on a syscall\_intercept library.

2.3 Benchmarks

The TPC-C benchmark is used to establish multiple user connections to concurrently access a database consisting of 1,000 warehouses (for a total dataset size of about 200GB) built on MySQL. Similarly, the YCSB Benchmark runs on MongoDB with multiple user connections, each generating a Zipf distributed key-value request workload. User request workloads are write-heavy (50% GET, 50% SET) unless otherwise noted, accessing the underlying MongoDB that stores a 150 GB dataset. Since very small key-value objects (typically smaller than 1KB) are prevalent in enterprise-level stores, we set object size at $1KB$ for the MongoDB dataset.

3. Set-up

This section provides instructions for installation and configuration required to prepare the environment to be used for the evaluation of the AppleS artifact.

3.1 Hardware setup

The physical testbed consists of two PowerEdge R630 servers, a computing server and a storage server. The former is used to run database instances, the AppleS artifact, and benchmarks while the database files reside on the latter. Specifically, the computing server is configured with 2 Intel Xeon E5-2650 V4 processors, 64GB of RAM, a Broadcom NetXtreme II BCM57810 10Gb NIC, and 2 \* 1TB SATA HDDs while the storage server is equipped with 2 Intel Xeon E5-2603 V4 processors, 64GB of RAM, a Broadcom NetXtreme II BCM57810 10Gb NIC, a 800GB SATA MLC Solid State Drives (SSDs) (for the OS installation), and a RAID-0 SSD array with five 800GB SATA MLC SSDs, consolidating all the logical volumes (LVs) (formatted as Ext4) for databases. All the servers are connected by a Dell N4032F switch with a peak bandwidth of 10Gb.

3.2 OSes setup

For the computing server, please use CentOS 8.3 with the Linux kernel 5.10.10 that should be enabled for full functionality of Cgroup(v2) while CentOS 7.3 with default Linux kernel can be installed on the storage server. All the OSs should enable the iSCSI protocol and install the necessary development tools (i.e., check "File and Storage Server" and "Development Tools" when installing the OS).

3.3 Database setup

Two types of databases, i.e., MySQL and MongoDB, are used in this setup. They include three versions used in the AppleS evaluation, i.e., MySQL 8.0.23 and MongoDB 4.4.3. You can install them from their official repositories by using yum tool or package installation.

3.3.1 The tips about installing and configuring MongoDB 4.4.3

1) Create a /etc/yum.repos.d/mongodb-org-4.4.repo file so as to install MongoDB directly using yum. Use the following repository file:

[mongodb-org-4.4]

name=MongoDB Repository

baseurl=https://repo.mongodb.org/yum/redhat/8/mongodb-org/4.4/x86\_64/

gpgcheck=1

enabled=1

gpgkey=https://www.mongodb.org/static/pgp/server-4.4.asc

2) Install MongoDB 4.4.3 by using the command "yum install -y mongodb-org-4.4.3 mongodb-org-server-4.4.3"

3) If mongd is installed into the directory /usr/bin (one can locate it by using the command "whereis mongod"), which is the running directory of MongoDB for AppleS, rename it to mongdx by using the command "mv /usr/bin/mongod /usr/bin/mongodx".

4) Remain the default settings for MongoDB 4.4.3.

3.3.2 The tips about installing and configuring MySQL 8.0.23

1) Download the MySQL 8.0.23 RPM package Bundle from https://downloads.mysql.com/archives/community/ for Red Hat Enterprise Linux 8 / Oracle Linux 8 (x86, 64-bit).

2) Install RPM rackages.

3) Locate the directory of mysqld by using the command "whereis mysqld", which is the running directory of MySQL for AppleS.

4) Add the following lines in /etc/my.cnf:

#E.g., /sdata2/mysql is the directory storing database files;

datadir=/sdata2/mysql

socket=/sdata2/mysql/mysql.sock

#datadir=/var/lib/mysql

#socket=/var/lib/mysql/mysql.sock

max\_connections=2000

max\_prepared\_stmt\_count=500000

innodb\_lock\_wait\_timeout = 500

skip-grant-tables #unseal it when the first run!

3.4 Storage setup

Two LVs should be created on the storage server, one is for the MySQL database storage while the other is used for the MongoDB database. Based on the estimated size of data sets for benchmarks, the size of each LV should be 512GB or larger.

3.4.1 The tips about configuring the storage server for the computing server;

1) Create LVs on the block device of the SSD array (e.g., sdb) by the following shell;

#!/bin/bash

pvcreate /dev/sdb

vgcreate ssdx /dev/sdb

for ((i=1; i<=2; i ++))

do

echo "The $i th LV is creating.....\n"

lvcreate -L1024G -n xenlv$i ssdx

mkfs -t ext4 /dev/ssdx/xenlv$i

done

2) Start the iSCSI target to provide the database storage for the computing server by the following shell;

#!/bin/bash

service tgtd start

service iptables stop

#For the computing serverr, which IP is 192.168.11.17;

for ((i=1; i<=2; i ++))

do

echo "The $i th LV for the computing server ready for connecting.....\n"

tgtadm --lld iscsi --op new --mode target --tid $i -T Stoage1.2022.xenlv$i:cse

tgtadm --lld iscsi --op new --mode logicalunit --tid $i --lun $i -b /dev/ssdx/xenlv$i

tgtadm --lld iscsi --op bind --mode target --tid $i -I 192.168.11.17

done

3) Start the iSCSI initiator on the computing server to connect the storage server and setup the data directory for databases' files;

#!/bin/bash

#Connect the storage server, which IP is 192.168.11.12;

for ((i=1; i<=2; i ++))

do

iscsiadm -m node -T Stoage1.2022.xenlv$i:cse -p 192.168.11.12 --login

done

#If sde and sdf are presented on the computing server as the block devices provided by the storage server, you can setup the data directory for databases' files;

#For MongoDB 4.4.3;

mount /dev/sde /sdata1

#For MySQL 8.0.23;

mount /dev/sdf /sdata2

3.5 Software dependencies

Before compiling AppleS, syscall\_intercept library should be first installed. Its source code and installation instructions are available at https://github.com/pmem/syscall\_intercept.

3.5.1 The tips about installing syscall\_intercept on the computing server running CentOS 8 (glibc version 2.27);

1) One need to modify /root/syscall\_intercept-master/test/syscall\_format.c by commenting out or remove the follow two lines before compiling it;

line 118: #include <ustat.h>

line 586: ustat(2, p0);

2) Move or copy /root/syscall\_intercept-master/include/libsyscall\_intercept\_hook\_point.h to /usr/include before compiling it.

3) Copy libsyscall\_intercept.a, libsyscall\_intercept.so, libsyscall\_intercept.so.0, libsyscall\_intercept.so.0.1.0 to /usr/lib64 after compiling it.

3) When one carries out "make test", don't worry about some errors. The installation is ok if you see the followings:

[root@localhost my\_build\_dir]# make test

Running tests...

Test project /root/syscall\_intercept-master/my\_build\_dir

Start 1: asm\_pattern\_nosyscall

1/40 Test #1: asm\_pattern\_nosyscall .................. Passed 0.00 sec

Start 2: asm\_pattern\_pattern1

2/40 Test #2: asm\_pattern\_pattern1 ................... Passed 0.00 sec

Start 3: asm\_pattern\_pattern2

3/40 Test #3: asm\_pattern\_pattern2 ................... Passed 0.00 sec

Start 4: asm\_pattern\_pattern3

4/40 Test #4: asm\_pattern\_pattern3 ................... Passed 0.00 sec

Start 5: asm\_pattern\_pattern4

5/40 Test #5: asm\_pattern\_pattern4 ................... Passed 0.00 sec

Start 6: asm\_pattern\_pattern\_loop

6/40 Test #6: asm\_pattern\_pattern\_loop ............... Passed 0.00 sec

Start 7: asm\_pattern\_pattern\_loop2

7/40 Test #7: asm\_pattern\_pattern\_loop2 .............. Passed 0.00 sec

Start 8: asm\_pattern\_pattern\_symbol\_boundary0

8/40 Test #8: asm\_pattern\_pattern\_symbol\_boundary0 ... Passed 0.00 sec

Start 9: asm\_pattern\_pattern\_symbol\_boundary1

9/40 Test #9: asm\_pattern\_pattern\_symbol\_boundary1 ... Passed 0.00 sec

Start 10: asm\_pattern\_pattern\_symbol\_boundary2

10/40 Test #10: asm\_pattern\_pattern\_symbol\_boundary2 ... Passed 0.00 sec

Start 11: asm\_pattern\_pattern\_symbol\_boundary3

11/40 Test #11: asm\_pattern\_pattern\_symbol\_boundary3 ... Passed 0.00 sec

Start 12: asm\_pattern\_pattern\_nop\_padding0

12/40 Test #12: asm\_pattern\_pattern\_nop\_padding0 ....... Passed 0.00 sec

Start 13: asm\_pattern\_pattern\_nop\_padding1

13/40 Test #13: asm\_pattern\_pattern\_nop\_padding1 ....... Passed 0.00 sec

Start 14: asm\_pattern\_pattern\_nop\_padding2

14/40 Test #14: asm\_pattern\_pattern\_nop\_padding2 ....... Passed 0.00 sec

Start 15: asm\_pattern\_pattern\_nop\_padding3

15/40 Test #15: asm\_pattern\_pattern\_nop\_padding3 ....... Passed 0.00 sec

Start 16: asm\_pattern\_pattern\_nop\_padding4

16/40 Test #16: asm\_pattern\_pattern\_nop\_padding4 ....... Passed 0.00 sec

Start 17: asm\_pattern\_pattern\_nop\_padding5

17/40 Test #17: asm\_pattern\_pattern\_nop\_padding5 ....... Passed 0.00 sec

Start 18: asm\_pattern\_pattern\_nop\_padding6

18/40 Test #18: asm\_pattern\_pattern\_nop\_padding6 ....... Passed 0.00 sec

Start 19: asm\_pattern\_pattern\_nop\_padding7

19/40 Test #19: asm\_pattern\_pattern\_nop\_padding7 ....... Passed 0.00 sec

Start 20: asm\_pattern\_pattern\_nop\_padding8

20/40 Test #20: asm\_pattern\_pattern\_nop\_padding8 ....... Passed 0.00 sec

Start 21: asm\_pattern\_pattern\_nop\_padding9

21/40 Test #21: asm\_pattern\_pattern\_nop\_padding9 ....... Passed 0.00 sec

Start 22: asm\_pattern\_pattern\_double\_syscall

22/40 Test #22: asm\_pattern\_pattern\_double\_syscall ..... Passed 0.00 sec

Start 23: asm\_pattern\_pattern\_rets

23/40 Test #23: asm\_pattern\_pattern\_rets ............... Passed 0.00 sec

Start 24: asm\_pattern\_pattern\_jmps

24/40 Test #24: asm\_pattern\_pattern\_jmps ............... Passed 0.00 sec

Start 25: fork\_logging

25/40 Test #25: fork\_logging ...........................\*\*\*Failed 0.01 sec

Start 26: hook\_with\_shared

26/40 Test #26: hook\_with\_shared .......................\*\*\*Failed 0.01 sec

Start 27: hook\_with\_static

27/40 Test #27: hook\_with\_static .......................\*\*\*Failed 0.01 sec

Start 28: hook\_clone

28/40 Test #28: hook\_clone .............................\*\*\*Failed 0.01 sec

Start 29: filter\_none

29/40 Test #29: filter\_none ............................ Passed 0.29 sec

Start 30: filter\_positive

30/40 Test #30: filter\_positive ........................ Passed 0.29 sec

Start 31: filter\_negative

31/40 Test #31: filter\_negative ........................ Passed 0.01 sec

Start 32: filter\_negative\_substring0

32/40 Test #32: filter\_negative\_substring0 ............. Passed 0.01 sec

Start 33: filter\_negative\_substring1

33/40 Test #33: filter\_negative\_substring1 ............. Passed 0.01 sec

Start 34: clone\_thread

34/40 Test #34: clone\_thread ........................... Passed 0.30 sec

Start 35: prog\_pie\_intercept\_libc\_only

35/40 Test #35: prog\_pie\_intercept\_libc\_only ........... Passed 0.29 sec

Start 36: prog\_no\_pie\_intercept\_libc\_only

36/40 Test #36: prog\_no\_pie\_intercept\_libc\_only ........ Passed 0.31 sec

Start 37: prog\_pie\_intercept\_all

37/40 Test #37: prog\_pie\_intercept\_all ................. Passed 0.31 sec

Start 38: prog\_no\_pie\_intercept\_all

38/40 Test #38: prog\_no\_pie\_intercept\_all ..............\*\*\*Failed Required regular expression not found.Regex=[intercepted\_call

] 0.01 sec

Start 39: vfork\_logging

39/40 Test #39: vfork\_logging ..........................\*\*\*Failed Required regular expression not found.Regex=[in\_child\_created\_using\_vfork

] 0.01 sec

Start 40: syscall\_format\_logging

40/40 Test #40: syscall\_format\_logging .................\*\*\*Failed 0.01 sec

83% tests passed, 7 tests failed out of 40

Total Test time (real) = 1.94 sec

The following tests FAILED:

25 - fork\_logging (Failed)

26 - hook\_with\_shared (Failed)

27 - hook\_with\_static (Failed)

28 - hook\_clone (Failed)

38 - prog\_no\_pie\_intercept\_all (Failed)

39 - vfork\_logging (Failed)

40 - syscall\_format\_logging (Failed)

Errors while running CTest

make: \*\*\* [Makefile:106: test] Error 8

3.6 AppleS setup

AppleS setup consists two steps, i.e., compilation and configuration. First, the source code and the compiled syscall\_intercept library are deployed in a source directory (e.g., /root/apples). And then the AppleS artifact can be compiled. Second, configure AppleS for the target database by the file apples\_configuration.txt, which is deployed in the same directory of the database executable file (i.e., running directory). After that, one can manually start the database instance running with AppleS loaded by LD\_PRELOAD.

3.6.1 The tips about installing and configuring AppleS.

1) Install syscall\_intercept and put the compiled syscall\_intercept library in the source directory of AppleS (e.g., /root/apples). Please refer to 3.5.

2) Compile AppleS by the following commands:

[root@computing1 apples]# chmod +x com.sh

[root@computing1 apples]# ./com.sh

3) Put the AppleS configuration file (i.e., apples\_configuration.txt) in the running directory of the database (i.e., the same directory with mongodx and mysqld), the following is an example:

#The name of DB;

database: mongodb

#The version of DB;

version: [4.4.3]

#The working directory of AppleS, where there are a series of log files that tracks different types of running information of

#AppleS, user-level I/O statistics (e.g., user-level I/O unfairness and CV of request latency), and optimization settings obtained

#by P optimization.

working\_directory: /root/mongodb\_opt1

#AppleS doesn't make I/O statistics until the expected number of users access the DB;

expect\_users: 512

#Make AppleS stay inactive but still make I/O statistics;

disable\_apples: 0

#Make the length of ramp-up time, which isn't counted until AppleS intercepted the 1st network-IO syscall issued by the target DB;

eff\_delay: 60

#The control interval, which defines the responsiveness level of AppleS in us;

control\_interval: 100000

#Define the statistical interval for user I/O unfairness and latency variability in the number of control intervals;

measure\_rounds: 10

#Define the time unit of statistical interval for P optimization in the number of control intervals;

sampling\_rounds: 100

# Define the length of time without I/O statistics, used to skip the transient fluctuation, in the number of sampling\_rounds;

skip\_sampling: 6

#Define the length of statistical interval for P optimization in the number of sampling\_rounds;

sampling\_times: 6

#1: Start P optimization 0: Normal state;

opt\_run: 0

#Define the upper limit for user I/O unfairness in the unit of 1%\*10000 (e.g., 10% can be represented by 1000)

fairness\_ub: 0

#Define the upper limit for user I/O unfairness in the unit of 1\*1000 (e.g., 1.0 can be represented by 1000)

latencyVar\_ub: 0

#Define the running time for the experiment;

end\_time: 310

#Define the maximum number of issued requests for the experiment;

end\_rqs: 10000000

#The default P value, P optimization will override it;

P\_target: 20

#The default P value, L optimization will override it;

L\_target: 1

4) Make sure one has enabled AppleS by the command "echo 1 > als\_ctl.txt" in the running directory of the database before running it while AppleS can be manually disabled by the command "echo 2 > als\_ctl.txt" at the directory at any time.

5) Start the database instance running with AppleS loaded by LD\_PRELOAD by the following commands:

#For MongoDB (e.g., running directory is /usr/bin, source directory is /root/apples, data directory is /sdata1);

LD\_PRELOAD=/root/apples/sys\_xpslo\_mon.so /usr/bin/mongodx --dbpath /sdata1 --bind\_ip\_all

#For MySQL (e.g., running directory is /usr/sbin, source directory is /root/apples, data directory is /sdata2):

LD\_PRELOAD=/root/apples/sys\_xpslo\_mon.so /usr/sbin/mysqld --basedir=/usr --datadir=/sdata2/mysql --plugin-dir=/usr/lib64/mysql/plugin --user=mysql --log-error=/var/log/mysqld.log --pid-file=/var/run/mysqld/mysqld.pid --socket=/sdata2/mysql/mysqld.sock

3.7 Benchmark setup

3.6.1 The tips about installing and configuring YCSB.

1) The workload configuration is shown as follows:

# Read/update ratio: 50/50

# Default data size: 1 KB records (10 fields, 100 bytes each, plus key)

# Request distribution: zipfian

#About 150GB data;

recordcount=150000000

operationcount=10000000

workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=true

readproportion=0.5

updateproportion=0.5

scanproportion=0

insertproportion=0

requestdistribution=zipfian

3.6.2 The tips about installing and configuring TPC-C.

1) Create the database for TPC-C:

[root@computing2 tpcc-mysql-master]# mysqladmin -h 127.0.0.1 -P 3306 -u root -p123 create tpcc1000

[root@computing2 tpcc-mysql-master]# mysql -h 127.0.0.1 -P 3306 -u root -p123 tpcc1000 < create\_table.sql

[root@computing2 tpcc-mysql-master]# mysql -h 127.0.0.1 -P 3306 -u root -p123 tpcc1000 < add\_fkey\_idx.sql

[root@computing2 tpcc-mysql-master]# mysql -h 127.0.0.1 -P 3306 -u root -p123

2) The database tpcc1000 was successfully created if you see the followings:

mysql> show databases;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| mysql |

| performance\_schema |

| sys |

| tpcc1000 |

+--------------------+

5 rows in set (0.01 sec)

mysql> use tpcc1000;

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed

mysql> show tables;

+--------------------+

| Tables\_in\_tpcc1000 |

+--------------------+

| customer |

| district |

| history |

| item |

| new\_orders |

| order\_line |

| orders |

| stock |

| warehouse |

+--------------------+

9 rows in set (0.00 sec)

3) Load data by using the following command;

nohup ./tpcc\_load -h127.0.0.1 -d tpcc1000 -u root -p "123" -w 1000 &

4. Evaluation workflow

This section includes two experiments (i.e., E1 and E2) that are conducted to evaluate AppleS's effectiveness on the two databases, MySQL and MongoDB, under excessive user parallelism and to validate the AppleS paper's key results and claims.

4.1 Experiment (E1): [MySQL 8.0.23+CentOS 8.3+Kernel 5.10.10] [30 human-minutes + 1 compute-hour]:

E1 aims to assess AppleS's capability to improve throughput for MySQL 8.0.23 with significantly enhanced user-level fairness and a low latency variability.

4.1.1 How to

E1 consists of 3 steps, i.e., baseline measurement, P optimization, and measurement under the AppleS control. Step 1: conduct 3 runs of TPC-C workload with 256 concurrent connections accessing MySQL 8.0.23 running with disabled AppleS, which only tracks user I/O statistics and records them in log files. Step 2: enable AppleS to conduct P optimization by setting "opt" at 1 and "disable" at 0. The P optimization is only required to run once for a specific database system and lasts about 6 minutes. Step 3: almost the same with Step 1 except for enabling AppleS by setting "disable" at 0. And then, one can compare between the measures obtained under the baseline and the AppleS-controlled case. The expected outcome would be over 20% throughput improvement, over 10 times user I/O fairness enhancement, and over 5 times lower latency variability.

Note: Before each run, please make sure that no mysqld process is still running. This means that you need to manually stop mysqld after the previous run by "pkill mysql".

4.1.2 Preparation

Create the database and load data for the TPC-C benchmark, which is only required to do once and lasts about 18 hours. Set the disk scheduler for the block device where the database files reside on as "mq-deadline" by using the command: "echo mq-deadline > /sys/block/sdf/queue/scheduler".

4.1.3 Execution

Configure apples\_configuration.txt for each step and deploy it in running directory before starting the step. The three apples\_configuration.txt files and the commands for starting AppleS and TPC-C workload for the three steps are shown below:

1) Step1: baseline measurement

(a) apples\_configuration.txt

database: mysql

version: [8.0.23]

working\_directory: /root/mysql\_baseline

expect\_users: 256

disable\_apples: 1

eff\_delay: 60

control\_interval: 100000

measure\_rounds: 10

sampling\_rounds: 100

skip\_sampling: 8

sampling\_times: 6

opt\_run: 0

fairness\_ub: 0

latencyVar\_ub: 0

end\_time: 335

end\_rqs: 10000000

P\_target: 20

L\_target: 1

(b) The commands for starting AppleS and TPC-C workload

[root@computing3 sbin]# echo 1 > als\_ctl.txt

[root@computing3 sbin]# LD\_PRELOAD=/root/apples/sys\_xpslo\_mon.so /usr/sbin/mysqld --basedir=/usr --datadir=/sdata2/mysql --plugin-dir=/usr/lib64/mysql/plugin --user=mysql --log-error=/var/log/mysqld.log --pid-file=/var/run/mysqld/mysqld.pid --socket=/sdata2/mysql/mysqld.sock

[root@computing tpcc-mysql-master]# ./tpcc\_start -h127.0.0.1 -P3306 -dtpcc1000 -uroot -p 123 -w1000 -c256 -r60 -i10 -l300

2) Step2: P optimization

(a) apples\_configuration.txt

database: mysql

version: [8.0.23]

working\_directory: /root/mysql\_opt

expect\_users: 256

disable\_apples: 0

eff\_delay: 60

control\_interval: 100000

measure\_rounds: 10

sampling\_rounds: 100

skip\_sampling: 8

sampling\_times: 6

opt\_run: 1

fairness\_ub: 0

latencyVar\_ub: 0

end\_time: 20

end\_rqs: 10000000

P\_target: 20

L\_target: 1

(b) The commands for starting AppleS and TPC-C workload

[root@computing3 sbin]# echo 1 > als\_ctl.txt

[root@computing3 sbin]# LD\_PRELOAD=/root/apples/sys\_xpslo\_mon.so /usr/sbin/mysqld --basedir=/usr --datadir=/sdata2/mysql --plugin-dir=/usr/lib64/mysql/plugin --user=mysql --log-error=/var/log/mysqld.log --pid-file=/var/run/mysqld/mysqld.pid --socket=/sdata2/mysql/mysqld.sock

[root@computing tpcc-mysql-master]# .tpcc\_start -h127.0.0.1 -P3306 -dtpcc1000 -uroot -p 123 -w1000 -c256 -r60 -i10 -l450

Note: We strongly sugggest you run the step 2 three times to prevent transient I/O fluctuation from affecting P optimization by ruling out the most deviated settings.

For example, you run the step 2 three times and got three lines settings in the file bslo\_opt\_log1.txt shown as the followings:

,11,1,8,128,32, 7411.64, 5143.94, 5670.29

,9,1,8,128,32, 8427.51, 5176.22, 5502.28

,11,1,8,128,32, 7406.85, 5290.27, 5908.03

The 1st number of each line indicates the optimized P value (i.e., 11, 9, 11). Two of them are the same, i.e., 11. Thus, one would like use line 1 or 3. AppleS always reads the last line of the file bslo\_opt\_log1.txt. If the file looks like the following:

,8,1,8,128,32, 7411.64, 5143.94, 5670.29

,9,1,8,128,32, 8427.51, 5176.22, 5502.28

,11,1,8,128,32, 7406.85, 5290.27, 5908.03

In this case, one would like to use the 2nd line (with the median value 9) and thus remove the last line.

3) Step3: measurement under the AppleS control

(a) apples\_configuration.txt

database: mysql

version: [8.0.23]

working\_directory: /root/mysql\_opt

expect\_users: 256

disable\_apples: 0

eff\_delay: 60

control\_interval: 100000

measure\_rounds: 10

sampling\_rounds: 100

skip\_sampling: 6

sampling\_times: 6

opt\_run: 0

fairness\_ub: 0

latencyVar\_ub: 0

end\_time: 335

end\_rqs: 10000000

P\_target: 20

L\_target: 1

(b) The commands for starting AppleS and TPC-C workload

[root@computing3 sbin]# echo 1 > als\_ctl.txt

[root@computing3 sbin]# LD\_PRELOAD=/root/apples/sys\_xpslo\_mon.so /usr/sbin/mysqld --basedir=/usr --datadir=/sdata2/mysql --plugin-dir=/usr/lib64/mysql/plugin --user=mysql --log-error=/var/log/mysqld.log --pid-file=/var/run/mysqld/mysqld.pid --socket=/sdata2/mysql/mysqld.sock

[root@computing tpcc-mysql-master]# ./tpcc\_start -h127.0.0.1 -P3306 -dtpcc1000 -uroot -p 123 -w1000 -c256 -r60 -i10 -l300

4.1.4 Results

Each run lasts 360 seconds, including 60-seconds ramp-up time and 300-seconds running time. The throughput measured under different cases can be directly observed from the benchmark results while CV of request latency and user I/O unfairness can be collected from the last two columns of the $a^{th}$ line (a = running\_time/control\_interval, e.g., if running\_time=300 seconds and control\_interval=0.1 second, a = 3000) in bslo\_cmd\_log1.txt. Each measure is the average over the three runs.

Note: There exists a minor difference of the starting time for I/O statistics between AppleS (from syscall signal) and the TPC-C benchmark. We use the end\_time setting to correct it and guarantee the running time identified by the AppleS is larger than 300 seconds.

4.2 Experiment (E2): [MongoDB 4.4.3+Cgroup (V2)] [30 human-minutes + 1 compute-hour]:

E2 aims to verify AppleS's capability to cooperate with Cgroup (V2) for MongoDB 4.4.3 to gain further improvement on user-level fairness and latency variability.

4.1.1 How to

E2 consists of 5 steps, i.e., baseline measurement, P optimization, measure under the AppleS control, measure under the control of Cgroup (V2), and measure under the control of both AppleS and Cgroup. Step 1: conduct 3 runs of YCSB workload with 512 concurrent connections accessing MongoDB 4.4.3 running with disabled AppleS. Step 2: the same with the Step 2 of E1. Step 3: almost the same with Step 1 except for enabling AppleS by setting "disable" at 0. Step 4: almost the same with Step 1 except for setting /cgroup2/cg1/cgroup.procs as the PID of MongoDB to enable the Cgroup (V2) control. Step 5: almost the same with Step 4 except for enabling AppleS. And then, one can compare among the measures obtained under the baseline, the AppleS-controlled case, the Cgroup-controlled case, and the dual-controlled case. The expected outcome is that AppleS can work with Cgroup to achieve a high user-level fairness (over 30 times improvement than the baseline) and low latency variability (over 2 times improvement than the baseline).

Note: Before each run, please make sure that no mongod or mongodx process is still running. This means that you need to manually stop mongod or mongodx after the previous run by "CTRL+C".

4.1.2 Preparation

Create the database and load data for the YCSB benchmark, which is only required to do once and lasts about 16 hours. Set the disk scheduler for the block device where the database files reside on as "mq-deadline" by using the command: "echo mq-deadline > /sys/block/sde/queue/scheduler". And then, create cgroup directory and enable its functionality by using the following commands:

[root@computing /]# mount -t cgroup2 none /cgroup2

[root@computing /]# cd /cgroup2

[root@computing cgroup2]# cat cgroup.subtree\_control

io memory pids

[root@computing cgroup2]# ps ax -L -o 'pid tid cls rtprio comm' |grep RR

1657 1722 RR 99 rtkit-daemon

[root@computing cgroup2]# pkill rtkit-daemon

[root@computing cgroup2]# echo "+cpu" > /cgroup2/cgroup.subtree\_control

[root@computing cgroup2]# cat cgroup.subtree\_control

cpu io memory pids

[root@computing cgroup2]# mkdir cg1

[root@computing cgroup2]# cd cg1

[root@computing cg1]# ls

cgroup.controllers cgroup.max.descendants cgroup.threads cpu.weight io.low memory.current memory.low memory.oom.group memory.swap.high pids.max

cgroup.events cgroup.procs cgroup.type cpu.weight.nice io.max memory.events memory.max memory.stat memory.swap.max

cgroup.freeze cgroup.stat cpu.max io.bfq.weight io.stat memory.events.local memory.min memory.swap.current pids.current

cgroup.max.depth cgroup.subtree\_control cpu.stat io.latency io.weight memory.high memory.numa\_stat memory.swap.events pids.events

[root@computing cg1]# echo threaded > cgroup.type

4.1.3 Execution

Configure apples\_configuration.txt for each step and deploy it in running directory before starting the step. The apples\_configuration.txt and workloadax files and the commands for starting AppleS, YCSB workload, and Cgroup for the three steps are shown below:

1) Step1: baseline measurement

(a) apples\_configuration.txt

database: mongodb

version: [4.4.3]

working\_directory: /root/mongodb\_baseline

expect\_users: 512

disable\_apples: 1

eff\_delay: 60

control\_interval: 100000

measure\_rounds: 10

sampling\_rounds: 100

skip\_sampling: 6

sampling\_times: 6

opt\_run: 0

fairness\_ub: 0

latencyVar\_ub: 0

end\_time: 310

end\_rqs: 10000000

P\_target: 20

L\_target: 1

(b) workloadax

recordcount=150000000

operationcount=10000000

workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=true

readproportion=0.5

updateproportion=0.5

scanproportion=0

insertproportion=0

requestdistribution=zipfian

(c)The commands for starting AppleS, YCSB workload, and Cgroup

[root@computing bin]# echo 1 > als\_ctl.txt

[root@computing bin]# LD\_PRELOAD=/root/apples/sys\_xpslo\_mon.so ./mongodx --dbpath /sdata1 --bind\_ip\_all

[root@computing3 YCSB-master]# ./bin/ycsb run mongodb -s -threads 512 -p "mongodb.maxconnections=1026" -p maxexecutiontime=360 -P workloads/workloadax > runSync.txt

2) Step2: P optimization

(a) apples\_configuration.txt

database: mongodb

version: [4.4.3]

working\_directory: /root/mongodb\_opt

expect\_users: 512

disable\_apples: 1

eff\_delay: 60

control\_interval: 100000

measure\_rounds: 10

sampling\_rounds: 100

skip\_sampling: 6

sampling\_times: 6

opt\_run: 1

fairness\_ub: 0

latencyVar\_ub: 0

end\_time: 20

end\_rqs: 10000000

P\_target: 20

L\_target: 1

(b) workloadax

recordcount=150000000

operationcount=10000000

workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=true

readproportion=0.5

updateproportion=0.5

scanproportion=0

insertproportion=0

requestdistribution=zipfian

(c)The commands for starting AppleS, YCSB workload, and Cgroup

[root@computing bin]# echo 1 > als\_ctl.txt

[root@computing bin]# LD\_PRELOAD=/root/apples/sys\_xpslo\_mon.so ./mongodx --dbpath /sdata1 --bind\_ip\_all

[root@computing3 YCSB-master]# ./bin/ycsb run mongodb -s -threads 512 -p "mongodb.maxconnections=1026" -p maxexecutiontime=360 -P workloads/workloadax > runSync.txt

Note: We strongly sugggest you run the step 2 three times to prevent transient I/O fluctuation from affecting P optimization by ruling out the most deviated settings.

For example, you run the step 2 three times and got three lines settings in the file bslo\_opt\_log1.txt shown as the followings:

,11,1,8,128,32, 7411.64, 5143.94, 5670.29

,9,1,8,128,32, 8427.51, 5176.22, 5502.28

,11,1,8,128,32, 7406.85, 5290.27, 5908.03

The 1st number of each line indicates the optimized P value (i.e., 11, 9, 11). Two of them are the same, i.e., 11. Thus, one would like use line 1 or 3. AppleS always reads the last line of the file bslo\_opt\_log1.txt. If the file looks like the following:

,8,1,8,128,32, 7411.64, 5143.94, 5670.29

,9,1,8,128,32, 8427.51, 5176.22, 5502.28

,11,1,8,128,32, 7406.85, 5290.27, 5908.03

In this case, one would like to use the 2nd line (with the median value 9) and thus remove the last line.

3) Step3: measure under the AppleS control

(a) apples\_configuration.txt

database: mongodb

version: [4.4.3]

working\_directory: /root/mongodb\_opt

expect\_users: 512

disable\_apples: 0

eff\_delay: 60

control\_interval: 100000

measure\_rounds: 10

sampling\_rounds: 100

skip\_sampling: 6

sampling\_times: 6

opt\_run: 0

fairness\_ub: 0

latencyVar\_ub: 0

end\_time: 310

end\_rqs: 10000000

P\_target: 20

L\_target: 1

(b) workloadax

recordcount=150000000

operationcount=10000000

workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=true

readproportion=0.5

updateproportion=0.5

scanproportion=0

insertproportion=0

requestdistribution=zipfian

(c)The commands for starting AppleS, YCSB workload, and Cgroup

[root@computing bin]# echo 1 > als\_ctl.txt

[root@computing bin]# LD\_PRELOAD=/root/apples/sys\_xpslo\_mon.so ./mongodx --dbpath /sdata1 --bind\_ip\_all

[root@computing3 YCSB-master]# ./bin/ycsb run mongodb -s -threads 512 -p "mongodb.maxconnections=1026" -p maxexecutiontime=360 -P workloads/workloadax > runSync.txt

4) Step4: measure under the control of Cgroup (V2)

(a) apples\_configuration.txt

database: mongodb

version: [4.4.3]

working\_directory: /root/mongodb\_baseline

expect\_users: 512

disable\_apples: 1

eff\_delay: 60

control\_interval: 100000

measure\_rounds: 10

sampling\_rounds: 100

skip\_sampling: 6

sampling\_times: 6

opt\_run: 0

fairness\_ub: 0

latencyVar\_ub: 0

end\_time: 310

end\_rqs: 10000000

P\_target: 20

L\_target: 1

(b) workloadax

recordcount=150000000

operationcount=10000000

workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=true

readproportion=0.5

updateproportion=0.5

scanproportion=0

insertproportion=0

requestdistribution=zipfian

(c)The commands for starting AppleS, YCSB workload, and Cgroup

[root@computing bin]# echo 1 > als\_ctl.txt

[root@computing bin]# LD\_PRELOAD=/root/apples/sys\_xpslo\_mon.so ./mongodx --dbpath /sdata1 --bind\_ip\_all

[root@computing apples] ./set\_cgroup\_mongodb.sh

[root@computing3 YCSB-master]# ./bin/ycsb run mongodb -s -threads 512 -p "mongodb.maxconnections=1026" -p maxexecutiontime=360 -P workloads/workloadax > runSync.txt

5) Step5: measure under the control of both AppleS and Cgroup

(a) apples\_configuration.txt

database: mongodb

version: [4.4.3]

working\_directory: /root/mongodb\_baseline

expect\_users: 512

disable\_apples: 0

eff\_delay: 60

control\_interval: 100000

measure\_rounds: 10

sampling\_rounds: 100

skip\_sampling: 6

sampling\_times: 6

opt\_run: 0

fairness\_ub: 0

latencyVar\_ub: 0

end\_time: 310

end\_rqs: 10000000

P\_target: 20

L\_target: 1

(b) workloadax

recordcount=150000000

operationcount=10000000

workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=true

readproportion=0.5

updateproportion=0.5

scanproportion=0

insertproportion=0

requestdistribution=zipfian

(c)The commands for starting AppleS, YCSB workload, and Cgroup

[root@computing bin]# echo 1 > als\_ctl.txt

[root@computing bin]# LD\_PRELOAD=/root/apples/sys\_xpslo\_mon.so ./mongodx --dbpath /sdata1 --bind\_ip\_all

[root@computing apples] ./set\_cgroup\_mongodb.sh

[root@computing3 YCSB-master]# ./bin/ycsb run mongodb -s -threads 512 -p "mongodb.maxconnections=1026" -p maxexecutiontime=360 -P workloads/workloadax > runSync.txt