

# OpenGauss实验文档

## 一、实验介绍

多路服务器在各种服务中应用广泛，特别是金融行业通过多路服务器进行性能提升，支持更加大的并发交易场景。数据库随着从单机走向分布式数据库，需要充分利用底层硬件的多核能力、发挥CPU算力。本实验针对openGauss的NUMA场景进行性能调优，在调优的过程中，可以掌握一条SQL在数据库中的全生命周期执行过程、学习操作系统和数据库性能优化手段、NUMA-aware技术在数据库产品中使用，为进一步研究打下基础。

## 二、实验内容

### 环境搭建

#### 安装环境

本实验每组为每组提供两台aarch64架构的多NUMA服务器，已分别预装openEuler20.03LTS和centos7系统，分组情况以及每组的账号密码可从如下链接获得：[22秋高级云操作系统分组](#)。

yum配置环境时可直接在root下进行，普通用户使用sudo -E yum install xxx。

#### 安装OpenGauss

openGauss安装请参照华为官方文档：[opengauss安装教程](#)。选择**3.1.0**版本极简版或企业版安装。我们推荐功能更齐全，自定义程度更高的**企业版**。文档中的openGauss安装包下载链接失效，可用地址为<https://opengauss.org/zh/download>。请在openEuler系统上选择openGauss单节点安装。

## Part1 物理性能测试

### (1) fio测试

#### 工具介绍

在Lab2 Cloud Storage中，大家已经使用过fio工具测试磁盘性能，在本次实验中我们依然使用它来做基准测试，以获得磁盘I/O的极限性能。

## 安装步骤

- 下载地址

下载链接: <https://github.com/axboe/fio>

使用文档: <https://fio.readthedocs.io/en/latest/index.html>

- 编译安装

```
1 ./configure
2 make
3 make install
```

## 测试

- 测试场景和参数建议

在这部分中, 你需要对硬盘分别在read、write、randread和randwrite场景下的极限I/O性能进行测试, 因此将iodepth设置成较大值, ioengine设置成异步的io\_uring或者libaio以提升性能表现。SSD的读写速度较快, 因此size和runtime建议不要太小, 防止结果波动过大。

我们推荐你通过rwmixwrite, rwmixread等参数控制, 进行一些读写混合的场景测试, 以及适当更改blockSize (参数bs) 大小查看读写速度的变化。

- 注意事项

1. fio测试会对系统的硬盘进行读写, **会存在破坏数据的风险**, 因此在生产环境上要谨慎使用, 不要对生产环境的硬盘进行裸盘读写测试。如果需要进行测试, 最好将硬盘挂载到文件系统上, 创建一个文件, 然后对指定的文件进行读写。
2. 建议多测几次, 剔除不正常数据。
3. 测试时不要运行其他应用, 防止CPU资源不足影响测试结果。

- 测试结果

在得出的结果中需要关注IOPS

运行结果示例:

```

yzh@sdic:~/fio test$ sudo fio test.config
fio: failed internal cconv test
fio: time based requires a runtime/timeout setting
4K-RR-QD1J1: (g=0): rw=read, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=io_uring, iodepth=1
fio: time based requires a runtime/timeout setting
4K-RR-QD3ZJ1: (g=1): rw=read, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=io_uring, iodepth=32
fio-3.23-dirty
Starting 2 threads
Jobs: 1 (f=1): [ (1),R(1)][94.1%][r=526MiB/s][r=135k IOPS][eta 00m:01s]
4K-RR-QD1J1: (groupid=0, jobs=1): err= 0: pid=6071: Thu Oct 28 03:51:55 2021
read: IOPS=19.4k, BW=75.6MiB/s (79.3MB/s)(1024MiB/13536msec)
clat (usec): min=46, max=14729, avg=51.21, stdev=67.02
lat (usec): min=46, max=14729, avg=51.32, stdev=67.02
clat percentiles (usec):
| 1.00th=[ 49], 5.00th=[ 49], 10.00th=[ 49], 20.00th=[ 50],
| 30.00th=[ 50], 40.00th=[ 50], 50.00th=[ 50], 60.00th=[ 51],
| 70.00th=[ 51], 80.00th=[ 51], 90.00th=[ 52], 95.00th=[ 57],
| 99.00th=[ 59], 99.50th=[ 81], 99.90th=[ 88], 99.95th=[ 94],
| 99.99th=[ 273]
bw ( KIB/s): min=69888, max=79205, per=100.00%, avg=77654.78, stdev=2099.45, samples=18
tops
lat (usec) : 50=36.53%, 100=63.43%, 250=0.02%, 500=0.01%
lat (msec) : 2=0.01%, 4=0.01%, 10=0.01%, 20=0.01%
cpu
: usr=99.95%, sys=0.00%, ctx=1694, majf=0, minf=0
IO depths : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
submit : 0=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
issued rwts: total=262144,0,0,0 short=0,0,0,0 dropped=0,0,0,0
latency : target=0, window=0, percentile=100.00%, depth=1
4K-RR-QD3ZJ1: (groupid=1, jobs=1): err= 0: pid=6075: Thu Oct 28 03:51:55 2021
read: IOPS=130k, BW=507MiB/s (532MB/s)(1024MiB/2018msec)
clat (usec): min=46, max=7580, avg=245.81, stdev=203.26
lat (usec): min=46, max=7589, avg=245.89, stdev=203.26
clat percentiles (usec):
| 1.00th=[ 68], 5.00th=[ 97], 10.00th=[ 113], 20.00th=[ 131],
| 30.00th=[ 147], 40.00th=[ 161], 50.00th=[ 178], 60.00th=[ 208],
| 70.00th=[ 302], 80.00th=[ 388], 90.00th=[ 449], 95.00th=[ 502],
| 99.00th=[ 709], 99.50th=[ 750], 99.90th=[ 832], 99.95th=[ 4359],
| 99.99th=[ 6980]
bw ( KIB/s): min=502912, max=535888, per=100.00%, avg=520684.00, stdev=16972.57, samples=4
tops
lat (usec) : 50=0.07%, 100=5.84%, 250=59.61%, 500=29.29%, 750=4.67%
lat (usec) : 1000=0.45%
lat (msec) : 4=0.01%, 10=0.06%
cpu
: usr=99.90%, sys=0.00%, ctx=253, majf=0, minf=0
IO depths : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=100.0%, >=64=0.0%
submit : 0=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.1%, 64=0.0%, >=64=0.0%
issued rwts: total=262144,0,0,0 short=0,0,0,0 dropped=0,0,0,0
latency : target=0, window=0, percentile=100.00%, depth=32

Run status group 0 (all jobs):
READ: bw=75.6MiB/s (79.3MB/s), 75.6MiB/s-75.6MiB/s (79.3MB/s-79.3MB/s), io=1024MiB (1074MB), run=13536-13536msec

Run status group 1 (all jobs):
READ: bw=507MiB/s (532MB/s), 507MiB/s-507MiB/s (532MB/s-532MB/s), io=1024MiB (1074MB), run=2018-2018msec

Disk stats (read/write):
dm-0: ios=498936/38, merge=0/0, ticks=67828/20, in_queue=67848, util=97.66%, aggrios=383486/32, aggrmerge=140802/6, aggrticks=38725/19, aggrin_queue=38745, aggrutil=97.63%
sda: ios=383486/32, merge=140802/6, ticks=38725/19, in_queue=38745, util=97.63%

```

## • 配置文件（供参考）

运行指令：

```
1 fio fio_bench.config
```

fio\_bench.config

```

1 [global]
2 name=NVMe-benchmark
3 filename=/dev/nvme1n1      #硬盘路径，请看注意事项
4 ioengine=io_uring
5 sqthread_poll=1
6 direct=1
7 numjobs=1
8 runtime=180
9 ramp_time=60
10 thread=1
11 time_based=1
12 randrepeat=0

```

```
13 norandommap=1
14 group_reporting=1
15 size=100G                #size建议不要过小，数据会不准确
16 stonewall
17
18 [4K-R]
19 bs=4k
20 rw=read
21 iodepth=32
22 stonewall
23
24 [4K-W]
25 bs=4k
26 rw=write
27 iodepth=32
28 stonewall
29
30 [4K-RR]
31 bs=4k
32 rw=randread
33 iodepth=32
34 stonewall
35
36 [4K-RW]
37 bs=4k
38 rw=randwrite
39 iodepth=32
40 stonewall
```

## 具体要求

你在实验报告中需要提交fio测试时的参数截图和运行结果的截图。

## (2) 其他测试

以上fio测试仅对平台SSD的IO性能进行了测试，我们推荐你对实验平台性能进行更全面的测试，如网络性能、跨NUMA节点内存访问性能等，请大家自行设计实验。

## 具体要求

你在实验报告中需要写明：

- 实验设计思路

- 测试工具
- 测试结果截图及分析

## Part2 基础环境下的测试（baseline）

### （1）跑TPC-C得出tpmC值

对于openGauss的TPCC性能调优，华为在官方文档中给出了基于arm平台调优的具体步骤和细节：[TPCC性能调优指导](#)。你可以参考其中环境的配置，请按照自己的实验思路自行调整。

[数据库和服务端绑核](#)中，gs\_guc设置的参数含义请查看[GUC参数说明](#)。

使用BenchmarkSQL工具进行TPCC测试。BenchmarkSQL是一款经典的开源数据库测试工具，内嵌了TPCC测试脚本，可以对EnterpriseDB、PostgreSQL、MySQL、Oracle以及SQL Server等数据库直接进行测试。

在客户端安装BenchmarkSQL测试工具并部署环境，步骤如下：

- 下载BenchmarkSQL标准测试工具[benchmarksql-5.0](#)。
- 将目录lib/postgresql下面的\*.jar 替换为openGauss适配的jar包。

```
1 $ pwd
2 /your path/benchmarksql-5.0/lib/postgres
3 $ ls
4 postgresql.jar #openGauss jdbc驱动。
5 postgresql-9.3-1102.jdbc41.jar.bak # 自带jar备份。
```

openGauss适配的JDBC版本包获取路径为[openGauss-x.x.x-JDBC.tar.gz](#)，请下载openeuler\_aarch64版本。

- 进入benchmarksql-5.0根目录，输入ant命令进行编译。

```
1 $ cd /your path/benchmarksql-5.0/
2 $ ant
```

- 创建benchmarkSQL配置文件，使用benchmarkSQL前需要配置数据库相关的信息，包括数据库账号、密码、端口、数据库名称。

```
1 $ cd /your path/benchmarksql-5.0/run
```

```
2 $ cp props.pg props.opengauss.1000w
3 $ vim props.opengauss.1000w
```

从props.pg拷贝一份配置文件并按如下修改。

```
1 db=postgres
2 driver=org.postgresql.Driver
3 // 修改连接字符串，包含IP、端口号、数据库，其中8.92.4.238为数据库服务端的千兆网卡IP。
4 conn=jdbc:postgresql://8.92.4.238:21579/tpcc1000?prepareThreshold=1&batchMode=on&fetchsize=10
5 // 设置数据库登录用户和密码。
6 user=bot
7 password=Gaussdba@Mpp
8
9 warehouses=1000 // 数量可自行调整
10 loadWorkers=200
11
12 // 设置最大并发数量，跟服务端最大work数对应。
13 terminals=812
14 //要为每个终端运行指定事务--runMins必须等于零
15 runTxnsPerTerminal=0
16 //要运行指定的分钟 - runTxnsPerTerminal必须等于零
17 runMins=5
18 //每分钟总事务数
19 limitTxnsPerMin=0
20
21
22 //在4.x兼容模式下运行时，设置为True。
23 //设置为false以均匀使用整个配置的数据库。
24 terminalWarehouseFixed=false
25
26 //以下五个值相加之和为100。
27 //45、43、4、4和4的默认百分比与TPC-C规范匹配。
28 newOrderWeight=45
29 paymentWeight=43
30 orderStatusWeight=4
31 deliveryWeight=4
32 stockLevelWeight=4
33
34 //创建文件夹以收集详细结果数据。
35 //通过注释取消此内容。
36 resultDirectory=my_result_%tY-%tm-%td_%tH%M%S
37 osCollectorScript=./misc/os_collector_linux.py
38 osCollectorInterval=1
39 //收集OS负载信息。
```

```
40 osCollectorSSHAddr=osuer@10.44.133.78
41 osCollectorDevices=net_enp3s0 blk_nvme0n1 blk_nvme1n1 blk_nvme2n1 blk_nvme3n
1
```

数据库名称和端口、ip、用户名等需要按照数据库服务端配置来设置，其余参数配置，比如\warehouse、terminals等等，需要你根据环境的实际情况和实验需要进行修改。

e. 导入数据前的准备。

需要替换benchmarkSQL中的文件，路径为benchmarksql-5.0/run/sql.common/tableCreates.sql。详细操作可以参考[openGauss数据库性能调优](#)。

6. 导入数据。

首先，连接数据库，创建数据库和用户数据。

```
1 create user bot identified by 'Gaussdba@Mpp' profile default;
2 alter user bot sysadmin;
3 create database tpcc1000 encoding 'UTF8' template=template0 owner bot;
```

然后，执行脚本导入数据。

```
1 ./runDatabaseBuild.sh props.opengauss.1000w
```

g. 运行TPCC测试程序。

```
1 numactl -C 0-19,32-51,64-83,96-115 ./runBenchmark.sh props.opengauss.1000w
```

运行TPCC测试程序后，可以得到类似于下图的测试结果。

```
Measured tpmC (NewOrders) = 1529000.28
Measured tpmTOTAL = 3397782.88
Session Start      = 2020-05-30 09:30:24
Session End        = 2020-05-30 17:30:25
Transaction Count  = 1630943994
```

## 具体要求

在搭建了openGauss数据库服务的物理机上，使用BenchmarkSQL进行TPCC性能测试。请提供顺利运行测试后，程序运行的截图和结果。该部分作为环境搭建部分的评分指标之一。



## (2) 单核多核测试结果分析（扩展性）

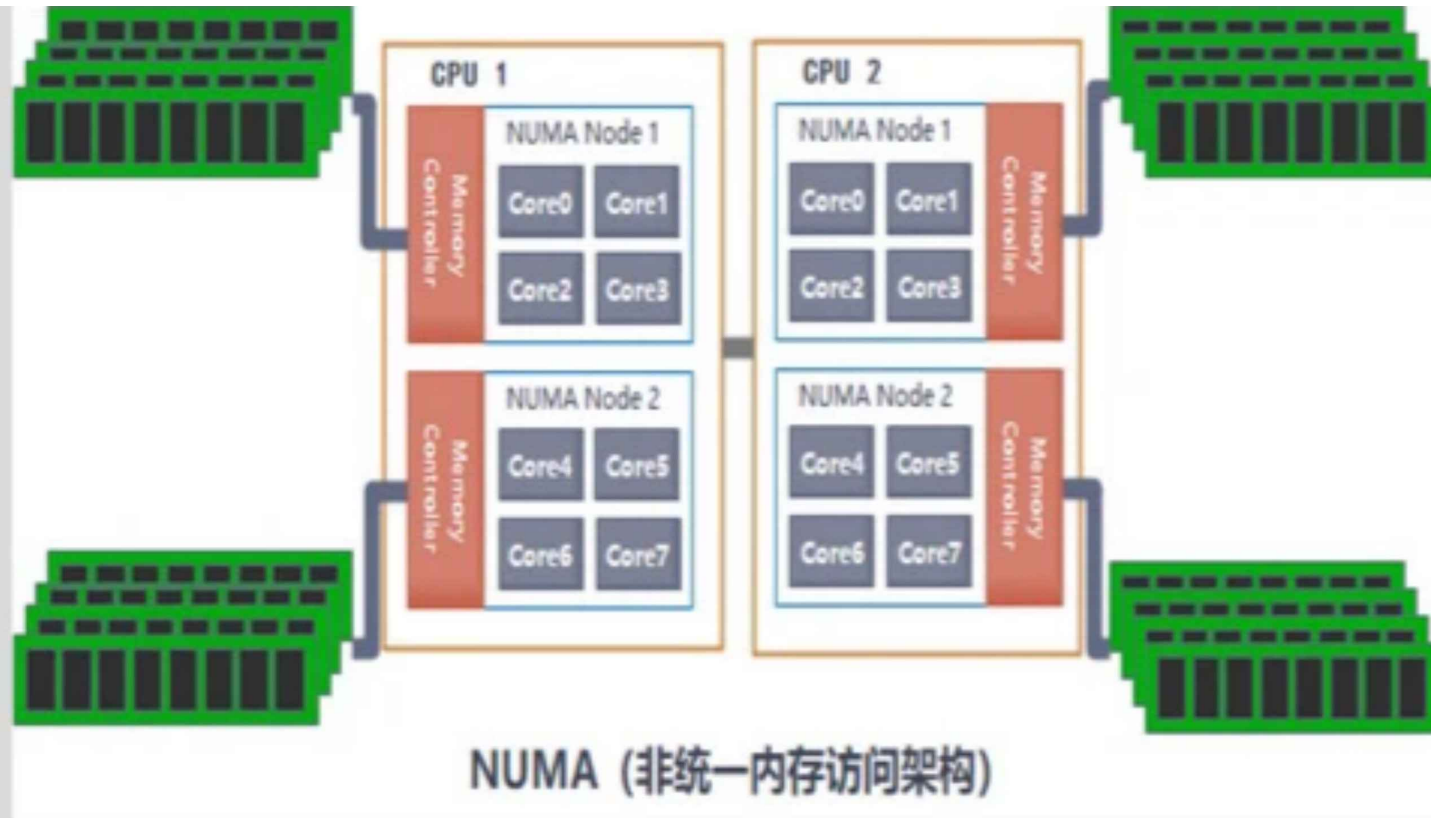
- 请你根据使用的CPU核数从少到多进行TPC-C测试（测试组数越多更容易得到完整趋势），画出性能变化趋势图（折线图或拟合曲线图）。请注意你测试时分配给openGauss使用的核应当位于不同的numa节点上。
- 结合性能变化趋势图，分析在numa架构的情况下性能随核心数增加的可扩展性。根据Part1测试的极限性能结果，分析硬盘I/O速度等硬件性能是否为你当前实验测试的瓶颈所在。

## Part3 numa-aware调优

### 背景介绍

传统数据库基于多进程模型，在NUMA架构下一般使用进程绑核的方式,或者通过对操作系统NUMA的配置进行控制修改内存分配方式来进行性能提升。Linux内核本身已经适配了NUMA特性，对应用提供了一些API用于更细粒度话的NUMA控制，即NUMA-aware技术，应用可以按照特性对内存和CPU使用特点进行NUMA优化。OLTP类型数据库因为其低时延高并发的业务特定，更需要在多核下进行NUMA-aware的业务造或数据结构优化以达到最优性能。

OpenGauss数据库基于PostgreSQL，是基于多进程模型设计的。对于每个新加入的客户端，OpenGauss都会fork一个新worker进程对客户端请求进行处理。各worker进程的内存并不共享，但为提供主进程和worker进程间通信、锁共享、维持日志写入等功能，主进程和worker进程间会通过共享内存来维护一些数据结构。这些共享的数据结构中有些在数据库运行过程中会被频繁访问，在NUMA场景下，若内存的分配位置不进行NUMA-aware的优化，则会导致跨核的内存访问，引发可观的访问延迟，严重影响整体系统性能。





## numa-aware调优思路

请你针对OpenGauss中**PGPROC**这一共享数据结构进行NUMA-aware优化。

**PGPROC**是OpenGauss中一个重要的结构体，在运行时维护了worker进程的线程ID、线程当前状态(等待的信号量)、会话ID、数据库用户OID等信息。该对象的所有实例均由主线程维护，在worker进程内全局唯一，**存储在共享内存中**。对于每个客户端连接，OpenGauss都会fork一个新的worker进程，并由主进程分配一个PGPROC结构体给新的worker进程。在worker进程执行涉及事务的相关操作时（如新建事务、事务提交、获取事务快照等），worker进程都需要访问/更新PGPROC结构体内数据，在有海量事务执行的OLTP场景下会被频繁访问。

```
struct PGPROC {
    /* proc->links MUST BE FIRST IN STRUCT (see ProcSleep,ProcWakeup,etc)
    SHM_QUEUE links; /* list link if process is in a list */

    PGSemaphoreData sem; /* ONE semaphore to sleep on */
    int waitStatus;      /* STATUS_WAITING, STATUS_OK or STATUS_ERROR */
                        /* STATUS_BLOCKING for update block session */

    Latch procLatch; /* generic latch for process */

    LocalTransactionId lxid; /* local id of top-level transaction current
                        * being executed by this proc, if running;
                        * else InvalidLocalTransactionId */
    ThreadId pid;          /* Backend's process ID; 0 if prepared xact
    /*
    * session id in mySessionMemoryEntry
    * stream works share SessionMemoryEntry with their parent sessions,
    * so sessMemorySessionid is their parent's as well.
    */
    ThreadId sessMemorySessionid;
    uint64 sessionid; /* if not zero, session id in thread pool*/
    GlobalSessionId globalSessionId;
```

若worker进程在访问、更新PGPROC结构体内数据时，触发了NUMA远程访问，将严重影响性能。请你通过分析源码的方式，学习PGPROC在OpenGauss数据库内部的使用方式，并尝试通过**内存分区、线程绑定等NUMA-aware的方法**，在主进程中优化PGPROC结构体的内存分配与存储结构，以此提高OpenGauss的OLTP性能。

## 具体要求

你的工作应该包含三部分：

- 分析PGPROC数据结构及其相关操作函数在TPC-C负载下的执行特点。
  - 请你学习主进程中PGPROC类型对象的分配方式，并分析其在内存分布上有什么特点，在NUMA场景下可能会导致怎样的性能问题。

- 请你学习PGPROC类型对象相关操作函数的功能，并在TPC-C负载下，通过**火焰图**分析PGPROC相关的操作函数执行的CPU时间占比，指出相关操作中性能瓶颈所在。
- 设计并实现NUMA-aware的PGPROC结构体分配方式。请在报告中描述清楚你的优化思路与方式、使用了哪些NUMA相关API、修改了哪些函数。openGauss中已经自带了numa优化的实现，可在源代码中通过宏定义ifdef \_\_USE\_NUMA找到所有numa相关函数定义及调用的地方。你可以在该版本上继续优化，也可以另起炉灶设计新的算法。重新编译openGauss请参照官方文档[编译指导书](#)，注意在**configure**时是否要添加**-D\_USE\_NUMA**参数。
- 对经过优化后的系统进行TPCC测试，与不使用numa优化（包括自带的）的系统进行比较。

## 参考

- openGauss中NUMA相关功能与参数设置
  - [src\bin\scripts\bind\\_net\\_irq.sh](#)：网卡按numa node绑核脚本
  - [numa\\_distribute\\_mode](#): NUMA GUC参数控制,开启则表示将部分共享数据和线程分布到不同的NUMA节点下，减少远端访存次数，提高性能
- PGPROC相关实现原理
  - 源代码位置：
    - <https://github.com/postgres/postgres/blob/master/src/include/storage/proc.h>
    - <https://github.com/postgres/postgres/blob/master/src/include/storage/procarray.h>
    - <https://github.com/opengauss-mirror/openGauss-server/blob/master/src/gausskernel/storage/lmgr/proc.cpp>
- NUMA API
  - openGauss已经具备numa能力，引入了numa lib作为三方件，代码中直接应用numa.h头文件即可，numa api可以参考linux：<https://linux.die.net/man/3/numa>
  - 常用的API如下三个：
    - [numa\\_run\\_on\\_node](#): 将当前任务及子任务运行在指定的Node
    - [numa\\_set\\_localalloc](#): 将调用者线程的内存分配策略设置为本地分配
    - [numa\\_alloc\\_onnode](#): 在指定的NUMA Node上申请内存
- NUMA-aware相关研究（仅供参考）
  - [Enabling NUMA-aware Main Memory Spatial Join Processing: An Experimental Study](#)
  - [Compact NUMA-aware Locks](#)
  - [DeLoc: A Locality and Memory-Congestion-Aware Task Mapping Method for Modern NUMA Systems](#)

- [NUMA-aware CPU core allocation in cooperating dynamic applications](#)
- [NUMA Awareness: Improving Thread and Memory Management](#)
- [Scalable NUMA-aware Blocking Synchronization Primitives](#)

### 三、评分标准

- 材料提交要求
  - 每个小组只需组长提交实验材料
  - 提交内容需要包括一份实验报告"组号\_report.{docx/doc/pdf}"、一个包含截图的文件夹"组号\_data"、和一个包含修改过的代码文件的文件夹（加上README文件简要描述代码修改的位置）。请将它们放在一起以压缩包形式提交。
  - 需要包含所有fio测试、tpc-c测试及其他测试的配置和测试结果的截图，若使用配置文件或脚本可以文件形式提交
- 评分细则（满分100分）
  - Part1（基础）：20分
    - fio测试结果截图：10分
    - 其他测试：10分
  - Part2（基础）：35分
    - TPC-C测试结果截图：10分
    - 单核和多核下TPC-C性能趋势图：15分
    - 性能分析：10分
  - Part3（拓展）：总分45分
    - 分析PGPROC数据结构及其相关操作函数在TPC-C负载下的执行特点
      - 分析PGPROC结构在内存分布上的特点以及NUMA场景下的性能问题：5分
      - 在TPC-C负载下，画出数据库运行时的火焰图：5分
      - 尝试分析性能瓶颈所在：5分
    - 根据实验平台调整参数或修改代码来提升数据库性能，并结合资料阐述你的优化思路：15分
    - 添加numa-aware机制后的优化效果（和不使用numa优化的系统相比）：15分
- 禁止抄袭他人的实验数据、实验内容。助教们会对实验测试结果的真实性进行普查，请确保能复现实验结果
- 报告内除实验主体内容外，还需要小组成员和贡献度情况（阐述明确工作内容）

