BESIII 离线软件在天河二号的测试技术指南

BESIII 离线软件系统（BESIII Offline Software System ，BOSS）是以中科院高能所为主导开发的粒子物理模拟、重建和分析的软件，它主要部署在高能所计算中心。本项目的目的是：

1. 将 BOSS 及其配套软件移植到天河二号上；
2. 验证 BOSS 运行得到的物理结果正确可靠；
3. 测量运行过程的内存、IO 指标，分析天河二号平台对 BOSS 是否有性能瓶颈；
4. 测量容器启动、数据库类型、睿频等因素对作业运行时间的影响；
5. 测量单核、单节点和多节点情况下作业的运行时间，与高能所计算中心比较；
6. 测量多节点情况下的加速比曲线，分析加速比曲线成因与提高方法；
7. 预测模拟 100 亿 J/psi 粒子在天河二号的运行情况，最终实现这一目标。

此技术指南描述了上述目标的实现方法。包含具体过程的测试方法、相关工具的使用方法、源代码、源代码存储目录等技术信息。

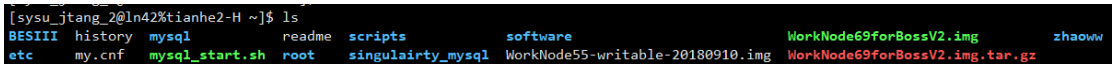
目录

一、BOSS 软件及其配套软件的移植

BOSS 采用 singularity 软件进行容器化部署到天河二号。mysql 尚不明确。singularity 是美国劳伦斯伯克利国家实验室开放的一款开源的操作系统级的虚拟化软件，类似于 docker，但不需要主机 root 权限并且可以利用主机的资源。

下面以 Debian 系为例说明如何使用（可能不准确，特别是关于 sudo 的使用）

1. 安装：
   sudo apt install singularity-container
2. 创建容器
   sudo singularity build --sandbox debian/ docker://debian
   容器以沙盒（sandbox）形式存在，或者说 debian 文件夹
3. 进入容器
   sudo singularity shell -w debian/
   实际上是进入了 debian 的文件夹
4. 在容器里面装软件
   例如安装 vim 编辑器：apt install vim （以 sudo 进入容器，此处不需 sudo）
   也可以拷贝本机文件到 debian/
   对于 BOSS，创建 /afs （from 马秋梅老师，可能我记不清是那些目录了）
   把 .ihep.ac.cn(→bes3→offline→ Boss cmthome-7.0.4 ExternalLib) 放到 /afs 下
   其他还有很多文件夹，不知道有没有用了。
5. 制作镜像
   sudo singularity build WorkNode69forBossV2.img debian/
   一旦做成镜像就不能再修改
6. 将镜像上传到天河二号上



还需要创建 BESIII 的文件夹，里面包含 cmthome-7.0.4 data mysql mysql.5.0 scripts workarea 等，这些是运行作业设置环境的。

From 郑伟老师邮件：

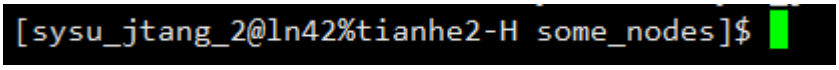
1) 先生成一个 Sincntific 6 的基础可写镜像
2) shell 进这个镜像在这个镜像基础上安装调试各种软件环境，建立一个和高能所现有计算环境相同的计算节点容器镜像。
3) 马老师把软件和数据拷贝到镜像中进行调试运行。
4) 最后打包成当前的镜像，并根据情况发布一个可写，或者只读的镜像。

如果知道 docker 镜像的建立也可以做一个 docker 镜像再转换为 singularity 的镜像。当前用的 mysql5.0 的 singualrity 镜像就是从 docker hub 上下载转换的。

7. 进入镜像
   只有 ln42 登录节点可以使用 singularity！如果未登录到 ln42，退出重新登

录。

查看登录节点的方法：看提示符：sysu_jtang_2@+登录节点名字



singularity shell -w --bind ${HOME}:/home WorkNode69forBossV2.img

此处-w 表示可写。--bind ${HOME}:/home 表示把本机$HOME 路径的文件夹挂载到容器内的/home 文件夹，这样在容器里面也能操作本机的文件。容器内，只有挂载的本机文件夹可以改动并且生效，其他文件夹可以改动但是下次进入容器就不会生效了。

8. 在容器里面执行一个 BOSS 模拟作业

先 source 环境：

source /home/BESIII/cmthome-7.0.4/setupCMT.sh
source /home/BESIII/cmthome-7.0.4/setup.sh
source /home/BESIII/workarea/7.0.4/TestRelease/TestRelease-00-00-86/cmt/setup.sh

运行作业：

cd /home/BESIII/workarea/7.0.4/TestRelease/TestRelease-00-00-86/run
boss.exe jobOptions_sim.txt

即可看到成功运行

9. 更改数据库类型

2019 年 8 月之前使用的是 mysql 数据库，之后为了测试需要，在 mysql 和 sqlite 之间互相切换。现在使用的是哪个数据库需要即时查看。更改数据库方法：

(1)进入容器：singularity shell -w --bind ${HOME}:/home WorkNode69forBossV2.img
(2)source 环境：（不然 cmt 无法运行）
source /home/BESIII/cmthome-7.0.4/setupCMT.sh
source /home/BESIII/cmthome-7.0.4/setup.sh
source /home/BESIII/workarea/7.0.4/TestRelease/TestRelease-00-00-86/cmt/setup.sh
(3)切换目录：
cd /BIGDATA1/sysu_jtang_2/BESIII/workarea/7.0.4/Database/DatabaseSvc/DatabaseSvc-00-00-26/share

(4)修改 DatabaseConfig.txt 左侧为 mysql 配置 右侧为 sqlite 配置

```
ApplicationMgr.DLLs += {"DatabaseSvc"};
ApplicationMgr.ExtSvc += {"DatabaseSvc"};

DatabaseSvc.DbType="mysql";
DatabaseSvc.Host="12.11.71.137";
DatabaseSvc.User="guest";
DatabaseSvc.Passwd="guestpass";
//DatabaseSvc.SqliteDbPath="/home/BESIII/data/database";
DatabaseSvc.ReuseConnection=true;
```

```
ApplicationMgr.DLLs += {"DatabaseSvc"};
ApplicationMgr.ExtSvc += {"DatabaseSvc"};

DatabaseSvc.DbType="sqlite";
//DatabaseSvc.Host="12.11.71.137";
//DatabaseSvc.User="guest";
//DatabaseSvc.Passwd="guestpass";
DatabaseSvc.SqliteDbPath="/home/BESIII/data/database";
DatabaseSvc.ReuseConnection=true;
```

(5)编译相关软件包
cd ../cmt
cmt br cmt config
cmt br gmake

source setup.sh
cd ../../../ReadDBBase/ReadDBBase-00-00-02/cmt/
cmt br cmt config
cmt br gmake
source setup.sh

(6)完成
注意：在高能所集群上使用 sqlite 不能提交后台作业（boss.condor、hep_sub）!
原因是 sqlite 不支持网络文件系统。

10. 如何在天河二号上面提交作业
首先说明天河二号提交作业的命令（什么叫提交作业就不解释了）
天河二号以节点为使用单位，每次至少用一个节点。每个节点有两颗计算的 CPU 和三颗协处理器（协助计算，可以忽略），真正使用的就两颗计算 CPU。计算的 CPU（型号：Intel Xeon E5-2692 v2）每颗 12 核，一个节点共 24 核。
天河二号提交作业命令有两个：yhrun 、yhbatch（yh 应该是代表银河的意思）
yhrun：前台提交作业，作业结果直接输出在屏幕，一旦提交就要等待作业完成才能退。

例子：yhrun -n 2 -N 1 -w cn7010 -c 10 -p rhenv ./a.out
选项：-n 任务数，表示后面的脚本 ./a.out 被同时执行几次
-N 使用节点数，表示一共使用几个节点
-c 使用核数，运行一次 ./a.out 使用多少个核
-p 使用的分区，一般是两个分区（bigdate 和 rhenv），但是 singularity 安装在 rhenv，所以运行 BOSS 作业只能选 -p rhenv
-w 指定使用的节点，用来验证同一节点运行两次作业用时是否相同（可能不同..）

但是必须满足：$n \geq N \quad \&\& \quad n \cdot c \leq 24 \cdot N$

./a.out 要有可执行权限，设定方法：chmod +x ./a.out

例子：yhbatch -N 5 -w cn[7010-7014] -p rhenv yhbatch_sub.sh
执行此条指令，屏幕返回 slurm-20192366.out 类似的，表示作业提交到后台了，然后就可以在终端做其他事情了。
选项：和 yhrun 基本一致，但是要注意几点：
(1)无论这里-n 写几，yhbatch_sub.sh 会且只会执行一次， -n 在这里没有实际意义
(2)yhbatch_sub.sh 要有可执行权限，设定方法：chmod +x yhbatch_sub.sh
(3)系统先根据-N 分配节点，然后在分配的第一个节点上执行 yhbatch_sub.sh 命令。如何使用这些节点（核）取决于 yhbatch_sub.sh 怎么写。
例如：yhbatch_sub.sh 包含:
yhrun -n 1 -c 24 sub1.sh
yhrun -n 2 -c 12 sub2.sh

使用 yhq 查看上交的作业和作业目前的状态，使用 yhi 查看天河二号节点使用情况。

$ yhq

```
[sysu_jtang_2@ln42%tianhe2-H ~]$ yhq
             JOBID PARTITION     NAME         USER ST       TIME  NODES NODELIST(REASON)
           2025324     rhenv yhbatch. sysu_jtang_2  R    2:56:03     20 cn[6981-6983,6997-7013]
           2025393     rhenv yhbatch. sysu_jtang_2  R    1:11:05     30 cn[6992-6995,7024-7030,7034-7036,7232-7235,7239-7250]
```

表示：现在有两个作业，编号分别是 2025324、2025393，它们在 rhenv 分区计算，交作业的脚本名字是 yhbatch.sh 提交作业的用户是 sysu_jtang_2，状态（ST）是运行，运行了多久（TIME），使用的节点数量（NODES），使用了那些节点（NODELIST）

$ yhi

```
[sysu_jtang_2@ln42%tianhe2-H ~]$ yhi
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
bigdata*     up   infinite     20   comp cn[7442-7446,7551-7552,7904-7907,8577-8579,8621-8623,8631-8633]
bigdata*     up   infinite      4   drng cn[7838,7934,8123,8519]
bigdata*     up   infinite   1269  alloc cn[7296-7299,7301,7304-7305,7309-7325,7327-7341,7343-7349,7351-7374,7376-7400,7402-7435,7437-7438,7447-7457
,7459-7467,7469-7480,7482-7515,7517-7518,7520-7526,7529,7531-7548,7550,7553-7578,7580-7596,7598-7620,7622-7651,7660-7668,7670-7687,7689-7705,7707,77
09-7775,7777-7779,7781-7810,7812-7822,7824-7833,7839-7849,7851-7861,7863-7903,7908-7923,7925-7931,7933,7935,7949-7952,7957,7959,7965-7993,7995-7999,
8004-8032,8037,8039,8044,8050,8057-8058,8060-8062,8064-8069,8071-8075,8077-8089,8091,8093-8102,8104-8111,8118-8121,8125-8126,8128-8146,8148-8154,815
6-8164,8169-8178,8184-8201,8204-8218,8220-8234,8236-8239,8244-8246,8248-8254,8260-8270,8274-8277,8279-8311,8314-8321,8323-8327,8334-8336,8338,8340-8
342,8344-8346,8348-8357,8359,8362-8375,8378-8390,8394-8395,8401-8402,8404-8411,8413-8416,8421-8433,8435,8437,8440-8442,8445-8449,8454-8455,8457-8460
,8463,8465-8473,8475-8478,8480,8498,8503-8511,8513,8515-8517,8520-8523,8525-8530,8533-8540,8547,8549-8555,8557-8560,8562-8564,8567,8581-8583,8585-85
92,8595,8610-8613,8615-8618,8626-8628,8641-8643,8646-8650,8652,8654-8667,8669-8679,8682,8685,8688-8689,8693-8698,8709-8725,8727-8733,8741-8743,8746-
8749,8752-8753,8760,8765-8766,8790,8792,8796,8801-8805,8816-8822,8830-8834,8836-8845,8849-8853,8861-8865,8867-8877,8879,8888-8902,8908-8911,8913-892
1,8926-8927,8934-8939,8942-8943,8954-8959,8990-8991,9010,9012-9015,9018,9044,9130,9176-9181,9183-9186,9188-9197,9200-9212]
bigdata*     up   infinite     95   idle cn[7834-7837,8000-8003,8113-8116,8179-8182,8240-8243,8256-8259,8271-8273,8396-8400,8418,8420,8434,8436,8462
,8464,8474,8479,8482-8486,8490,8497,8518,8541,8571,8573,8575,8596,8607,8645,8651,8681,8683,8691,8699,8701,8707,8751,8756,8762,8767-8770,8772-8776,87
80,8783,8791,8823,8855-8858,8883-8886,8903-8907,8998-9001]
rhenv        up   infinite      1   drng cn7275
rhenv        up   infinite     53  alloc cn[6981-6983,6987,6992-6995,6997-7013,7024-7030,7034-7036,7232-7235,7237,7239-7250,7279]
rhenv        up   infinite     24   idle cn[6978-6979,6990,7016,7018,7021,7038-7039,7180,7183,7210,7253-7254,7256-7257,7259-7260,7262-7263,7267,7271
,7276,7280,7292]
```

比如 rhenv 分区，此时无法使用 1 个节点 cn7275，被占据 53 个，分别是……，空闲 24 个，分别是……

下面提交作业的脚本适合理解原理：
cd ~/zhaoww/myfirstsub
现在里面应该有 rhopi.dec sim.txt sub_this.sh write_job.sh
提交作业：yhbatch -n 1 -N 1 -c 24 -p rhenv sub_this.sh
没有意外的话，应该可以成功的。
sub_this.sh：

```
1 #!/bin/bash
2
3 ############## 单核单进程作业 ########################################
4
5 export TIMEFORMAT=$'\nreal\t%3lR\tuser\t%3lU\tsys\t%3lS\t cpu %P'
6 export EXE="singularity exec --bind ${HOME}:/home /BIGDATA1/sysu_jtang_2/WorkNode69forBossV2.img /home/zhaoww/myfirstsub/write_job.sh"
7
8 time($EXE)
9
10 wait
```

提交作业的 shell 脚本，第一行都加上 #!/bin/bash
line5：设置时间格式
line6：令：EXE=到容器里面执行 write_job.sh 脚本里面的内容
line8：执行 EXE 并计时
line10：等待脚本的指令完成后再退出

write_job.sh

```
1 source /home/BESIII/cmthome-7.0.4/setupCMT.sh
2 source /home/BESIII/cmthome-7.0.4/setup.sh
3 source /home/BESIII/workarea/7.0.4/TestRelease/TestRelease-00-00-86/cmt/setup.sh
4
5 cd /home/zhaoww/myfirstsub
6 boss.exe sim.txt
7
```

由于在容器里面了，所以直接 source 环境，执行 boss.exe

为什么这里的路径/home/zhaoww/myfirstsub，而不是
/BIGDATA1/sysu_jtang_2/zhaoww/myfirstsub

因为 source 环境和 boss.exe 都是在容器里面执行的，我们要用容器里面的地址（由于挂载的原因）

如果不好记，那就都在~/zhaoww/里面交作业，只需要更改
/home/zhaoww……后面的地址就行了。

上面是占用一个节点（24 核）但却只用了一个核，显然很浪费。如何使用 24 核？

```
1 #!/bin/bash
2
3 export TIMEFORMAT=$'\nreal\t%3lR\tuser\t%3lU\tsys\t%3lS\t cpu %P'
4 export NUMJOB=24
5 export SLEEPTIME=0.1
6
7 for num in `seq 1 1 $NUMJOB`
8 do
9     export EXE="singularity exec --bind ${HOME}:/home /BIGDATA1/sysu_jtang_2/WorkNode69forBossV2.img /home/zhaoww/myfirstsub/write_job.sh"
10    time($EXE) &
11    sleep $SLEEPTIME
12 done
13
14 wait
15
```

line7：从 1 到 24 的循环
line10：注意最后的&符号，表示每次执行完一个 EXE 就换一个核，这样换 24 次，就把 24 个核用完了。而且这 24 个核的 EXE 是几乎同时开始的，实现了作业并行。
用这种方法可以推广到多节点并行。

二、验证 BOSS 运行得到的物理结果正确可靠

我们把软件部署到了天河二号上，首先要保证的是物理结果的正确性。如果相同的脚本在高能所集群运行一个结果，在天河二号运行又是一个结果，那肯定是不行的。

为了验证结果的正确性，我们需要进行验证

验证内容：模拟出图的 *.root 和 重建分析以后的 *.root；

验证方法：对应 branch 的数值相减，看时候都是 0；

理想结果：所有 branch 相减都是 0.

1. 模拟出图
   使用 https://docbes3.ihep.ac.cn/viewvc/cgi-bin/viewvc.cgi/BESIII/BossCvs/Simulation/McTestAlg/
   先拷贝来最新版本：
   cd /scratchfs/bes/zhaoww/boss-7.0.4/workarea
   cmt co -r McTestAlg-00-00-17 McTestAlg
   这样拷贝过来的包是有问题的，需要修改，下面的操作是修改之后才可以的。
   cd ./Simulation/McTestAlg/McTestAlg-00-00-17/share
   修改输入文件、输出文件、输出等级和事例数（在 line20 之后）
   boss.exe mct.txt 即可出*.root
2. 使用相同的脚本在高能所和天河二号上分别模拟、重建和分析，将模拟得到的*.rtraw 使用 McTestAlg 出图。
3. 使用如下脚本对比两者的*.root 文件是否相同。

```
{
   double ivx0 ;       //声明一个double变量，命名为branch的名字
   double tvx0 ;
   TCanvas c1;
   TFile *f1 = new TFile("/scratchfs/bes/zhaoww/boss-7.0.4/sim_validation/v2_root_code/IHEP.root");
   TFile *f2 = new TFile("/scratchfs/bes/zhaoww/boss-7.0.4/sim_validation/v2_root_code/TH-2.root");
   TTree *T1 = (TTree*)f1->Get("vxyz");
   TTree *T2 = (TTree*)f2->Get("vxyz");

   T1->SetBranchAddress("vy0",&ivx0);      // "vx0",branch名字，&vx0，声明的名字
   T2->SetBranchAddress("vy0",&tvx0);

   TH1F *h = new TH1F("vy0","vy0 distribution",10000,-120,60);

   Long64_t nentries1 = T1->GetEntries();
   Long64_t nentries2 = T2->GetEntries();

   if(nentries1==nentries2)  {
       for (Long64_t i=0;i<nentries1;i++)
       {
         T1->GetEntry(i);      //每个entry对应的vx0是不一样的
         T2->GetEntry(i);
         h->Fill(ivx0-tvx0);        // entry范围 0~19236
         h->Draw();
       }
       c1.Print("vy0.pdf"); }
    else
       cout<<"nentries1!=nentries2"<<endl;

}
```

更为精确的脚本如下：

区别在于对于 entries 的每一个值的处理。

（不好解释 emmm….. Linus Torvalds: “**Talk is cheap**. **Show me the code**.”）

```
{
   double ivx0 ;       //声明一个double变量，命名为branch的名字
   double tvx0 ;
   TCanvas c1;
   TFile *f1 = new TFile("/scratchfs/bes/zhaoww/boss-7.0.4/sim_validation/v2_root_code/IHEP.root");
   TFile *f2 = new TFile("/scratchfs/bes/zhaoww/boss-7.0.4/sim_validation/v2_root_code/TH-2.root");
   TTree *T1 = (TTree*)f1->Get("geff");
   TTree *T2 = (TTree*)f2->Get("geff");
   T1->SetBranchAddress("vx0",&ivx0);      // "vx0",branch名字，&vx0，声明的名字
   T2->SetBranchAddress("vx0",&tvx0);
   TH1F *h = new TH1F("vx0","vx0 distribution",4,-1,3);
   Long64_t nentries1 = T1->GetEntries();
   Long64_t nentries2 = T2->GetEntries();

   if(nentries1==nentries2)  {
       for (Long64_t i=0;i<nentries1;i++)
       {
         T1->GetEntry(i);      //每个entry对应的vx0是不一样的
         T2->GetEntry(i);
               if (ivx0==tvx0)
               {  h->Fill(1); }
               else
               {  h->Fill(0); }
       }
       h->Draw();
       c1.Print("vx0.pdf"); }
   else
       cout<<"nentries1!=nentries2"<<endl;
}
```

验证过程代码的存储位置：高能所/scratchfs/bes/zhaoww/boss-7.0.4/sim_validation

内含：

rhopi_1w_lxslc6　rhopi_1w_TH2　模拟重建分析的文件

root_code　v2_root_code　v3_root_code　v4_root_code 逐步精确的四个版本的

验证方法

三、测量运行过程的内存、IO 指标

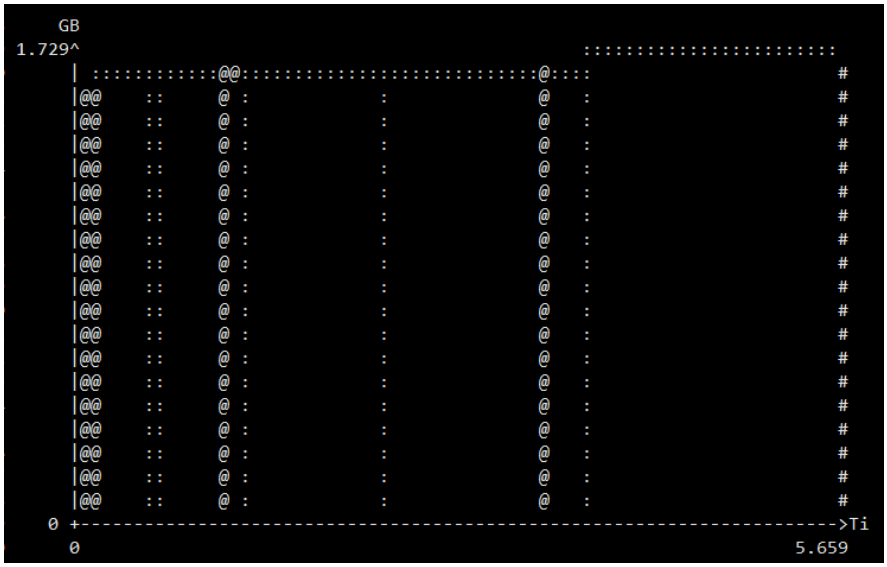
1. 内存

内存的测量我尝试了两种方法：使用 valgrind --tool=massif ，使用/usr/bin/time
源代码位置：天河二号 ~/zhaoww/massif ~/zhaoww/callgrind 也有部分
高能所 /besfs/users/zhaoww/valgrind
文件夹 valgrind 内含有 memcheck、callgrind、massif 的联系，按需取用吧

1.1 massif 的用法 (没用的不用看了)

命令：valgrind --tool=massif –pages-as-heap=yes boss.exe sim.txt
--pages-as-heap=yes 统计所有内存
-- time-unit=<i|ms|B> [default: i]
设置 Massif 分析的时间单位，有三种：
按照程序运行的指令数；
按照时间（毫秒）；
按照申请及释放的字节数变化。

输出：massif.out.PID
查看：ms_print massif.out.PID
查看就会出现这样的：



“:”表示普通快照、“@”表示详细快照、“#”表示峰值快照
每个垂直条表示快照，即在某一时间点内存使用量的测量。
如果下一快照相距不止一列，则从快照顶部到下一个快照列之前绘制水平字符行。
底部的文本显示了这个程序的 25 个快照，即每个堆分配/释放一个快照，再加上几个额外快照。

```
Number of snapshots: 79
 Detailed snapshots: [4, 10, 14, 20, 34, 44, 54, 58, 68, 73 (peak)]

--------------------------------------------------------------------------------
  n        time(i)         total(B)   useful-heap(B) extra-heap(B)    stacks(B)
--------------------------------------------------------------------------------
  0              0           12,288          12,288            0            0
  1     21,258,103       49,930,240      49,930,240            0            0
  2     39,145,802       54,792,192      54,792,192            0            0
  3     67,278,751       61,968,384      61,968,384            0            0
  4     93,880,591      161,312,768     161,312,768            0            0
```

<1>快照数 n
<2>时间 time（i）
<3>总内存消耗量；
<4>可用堆内存的字节数，即程序申请内存时，指定的数量；
<5>额外堆内存的字节数，包括管理内存增加的字节；
<6>栈的大小，默认关闭

1.2 /usr/bin/time
说明：此命令是 time（路径/bin/time）的扩充版，使用时要加上路径
命令：/usr/bin/time -f "\nmax=%M \t Kbytes" boss.exe sim.txt
更多：\n 换行 \t 制表符
E=real time U=user time S=sys time P=cpu using
M=最大驻留集 K=平均内存大小
W=进程从主内存内交换次数 w 等待次数，如 IO 等待
I=输入 O=输出

```
real=44:08.60           use=2575.59             sys=4.40                cpu=97%
max=490664                      mem=0           KBytes
swap=0                  waitnum=2998
input=1416                      output=96080
```

来自：/besfs/users/zhaoww/valgrind/massif/usr_time/sub.sh.err.41940652.0

1.3 为什么 massif 与/usr/bin/time 差别那么大？
From 邹佳恒老师邮件
1，/usr/bin/time -f %M 得到的是物理内存，符合我们的要求；
2，valgrind --tool=massif --pages-as-heap=yes 得到的是虚拟内存，所以更大；
测试脚本见：/besfs/users/zhaoww/mem_zoujh 里面有详细注释
实际使用是看物理内存，目前 500MB×24=12GB < 88GB
所以天河二号的内存不会限制 BOSS 作业运行。

2. 天河二号的 IO 情况
如何准确理解 IO，我也不是很清楚。在估算过程中，简单地把 IO 当成对硬盘的读写速度。
天河二号/BIGDATA1/sysu_jtang_2/zhaoww/jpsi/some_nodes
此文件夹是测试天河二号上多节点模拟 J/psi 粒子的。统计一个（或者几个）节点模拟 1 万 J/psi 生成的*.rtraw 和相关文件的总大小，统计平均用时，计算出单节点模拟 1 万 J/psi 粒子时，单作业的磁盘写入速度约为 8.5Kbytes/s 。

根据天河二号技术支持团队的邮件
From 天河二号技术支持团队
单节点理论上可以达到 1GB/s（峰值），BIGDATA1 存储总体 IO 测试中可以达到 10GB/s（峰值）。但具体应用中这个数据会受多方面因素影响。

按照总体 8GB/s 计算，可以同时支持约 4 万个作业同时写入磁盘。

四、测量容器启动、数据库类型、睿频等因素对作业运行时间的影响

1. 测量容器启动
天河二号：/BIGDATA1/sysu_jtang_2/zhaoww/cores/part_cores
此目录下含有：contains_diff　　cores_diff
contains_diff：
主要测量容器启动的时间，次级目录：
1_contain　使用单节点单核，开一次容器，容器里面运行 24 次作业
24_contain　使用单节点单核，开一次容器执行一个作业，这样串行 24 次
open_contain 测量容器空载、容器+source 环境、容器+source 环境+模拟 5 事例用时

这几个目录的测量比较混乱，而且结果也不具有说服力，建议新开目录重新测。

cores_diff
测量单节点使用 23 核、22 核、21 核的运行时间

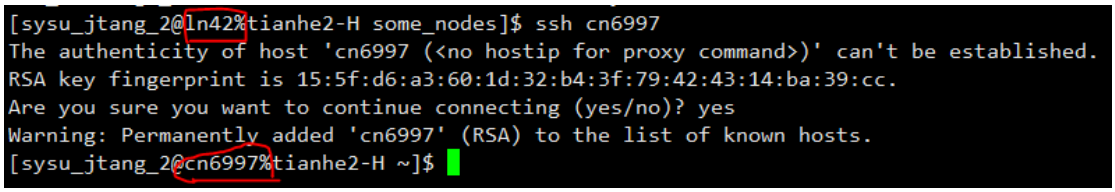
2. 数据库类型对运行时间影响
本来计划在高能所测量的，但是高能所使用 sqlite 无法提交作业，只能在天河二号上面测量。
单核：/BIGDATA1/sysu_jtang_2/zhaoww/cores/single_core/
此目录下的 7k，9k，11k，15k，20k，40k，60k 次级目录
每个次级目录都有 slurm-sqlite 和 slurm-mysql，分别代表使用 sqlite 和 mysql 运行相同脚本的输出日志。

单节点：/BIGDATA1/sysu_jtang_2/zhaoww/cores/single_node
此目录下的 8k，10k，30k，50k 次级目录
每个次级目录都有 slurm-sqlite 和 slurm-mysql，分别代表使用 sqlite 和 mysql 运行相同脚本的输出日志。

3. 睿频

```
[sysu_jtang_2@ln42%tianhe2-H some_nodes]$ yhq
             JOBID PARTITION     NAME         USER ST       TIME  NODES NODELIST(REASON)
           2026270     rhenv yhbatch. sysu_jtang_2 PD       0:00     40 (Resources)
           2026254     rhenv yhbatch. sysu_jtang_2  R      10:17     35 cn[6997-7013,7024-7029,7239-7250]
```

这里使用着的节点是 cn[6997-7013,7024-7029,7239-7250]，我们可以登录到其中一个节点，命令：ssh cn6997。登陆之后可以查看此节点此时的状态。



注意此时已经由 ln42，变成了 cn6997
使用 watch -n 0 "cat /proc/cpuinfo | grep -i mhz" 动态查看 CPU 频率
使用 Ctrl+Z 退出查看
使用 exit 返回 ln42

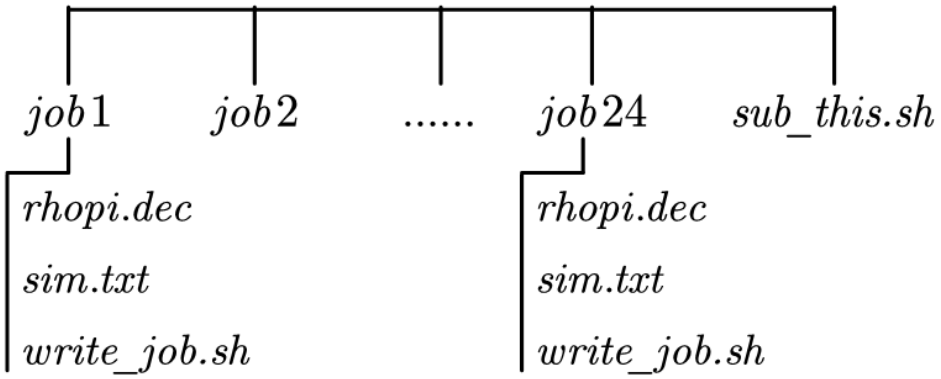
计算节点 CPU 主频 2.2GHz，超过此值即可认为是睿频。

五、测量单核、单节点和多节点情况下作业的运行时间，与高能所计算中心比较

1. 单核
高能所 /besfs/users/zhaoww/core

天河二号 /BIGDATA1/sysu_jtang_2/zhaoww/cores/single_core
天河二号上提交作业一般准备一下四个文件即可：
rhopi.dec sim.txt sub_this.sh write_job.sh
sim.txt 不要改动 run ID，因为就这一个 run ID 可以运行成功
sub_this.sh 修改为在容器里指向 write_job.sh 的地址
write_job.sh 前三行不变，后面修改为容器里指向 sim.txt 的地址

2. 单节点
天河二号 /BIGDATA1/sysu_jtang_2/zhaoww/cores/single_node/
原理简单，但是说起来就复杂了。单节点要并行 24 个模拟作业，为了避免互相影响，我就把 24 个脚本放到了 24 个目录里面。
每个目录命名 job1、job2……job24，每个目录里面含有：
rhopi.dec sim.txt write_job.sh
sim.txt 随机数要保证 24 个作业都不同
write_job.sh 修改为在容器里指向 sim.txt 的地址
sub_this.sh 和 job1-24 同一级，里面依次调用 24 个 write_job.sh
把这个说清楚太费劲了，看图吧，或者到相应目录找个例子看看

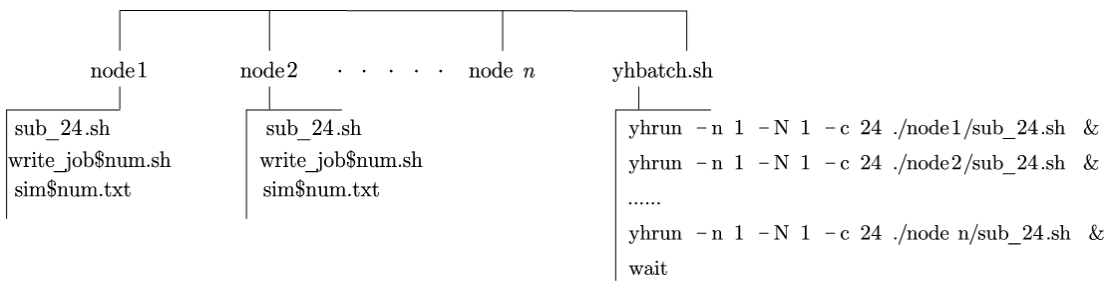


3. 多节点

天河二号 /BIGDATA1/sysu_jtang_2/zhaoww/cores/some_nodes

以后大多数时间就是提交多节点作业，所以对多节点的情况要尽量熟悉。

目录结构如下：



node1……node *n* ： 每个 node 等价于单个节点的，这里为了方便，每个 node *n* 里面就没有分 24 个次级目录了。注意每个 sim.txt 的随机数都是不同的，最好的办法就是用作业使用的第几个核数做随机数种子。

那么问题来了，如果要实验 70 个节点并行，怎么生成这么多文件夹？难道手工创建？

经过我从数个节点提交作业的经验，逐步精简做成了一个半自动化的生成脚本。至于这个脚本是怎么实现的，我也不明白了…….希望后续可以共同完善。

使用方法：

$ cd /BIGDATA1/sysu_jtang_2/zhaoww/cores/some_nodes

$ mkdir 70 #创建测试多节点的文件夹，用节点数命名

$ cp -r gen_dir_loop.sh nodeT yhbatch.sh ./70 #复制三个模板

$ cd ./70/nodeT 修改 nodeT 里面的 sub_this_24.sh write_job.sh 两个脚本指向的地址，使之符合 70 这个目录结构（这一步应该可以合并到 gen_dir_loop.sh）

$ cd /BIGDATA1/sysu_jtang_2/zhaoww/cores/some_nodes/70

$ bash gen_dir_loop.sh 70

修改 yhbatch.sh 里面循环的次数，使之符合节点数。即可

提交作业：yhbatch -N 70 -p rhenv yhbatch.sh

六、测量多节点情况的加速比曲线，分析加速比曲线成因与提高方法

1. 定义与公式

由于我们的作业是分配到不同核同时进行，所以是伪并行，不能按照加速比一般定义计算。于是我就仿照一般定义做了我们的定义：

$$\text{加速比} = S_p = \frac{N_p}{N_1} = \frac{p\text{节点每秒处理事例数}}{\text{单节点每秒处理事例数}}$$

$$\text{效率} = E_p = \frac{N_p}{pN_1} = \frac{p\text{节点每秒处理事例数}}{p\cdot\text{单节点每秒处理事例数}}$$

理想情况依然是 $S_p=p$，$E_p=1$。

类似的，也可以定义核加速比：

$$\text{核加速比} = S_c = \frac{N_c}{N_1} = \frac{c\text{核每秒处理事例数}}{\text{单核每秒处理事例数}}$$

$$\text{核效率} = E_{24} = \frac{N_c}{c\cdot N_1} = \frac{c\text{核每秒处理事例数}}{c\cdot\text{单核每秒处理事例数}}$$

简单化简公式：

假设使用了 p 节点，每个节点 24 核，每个作业模拟 eve 个事例

并行情况下所有作业平均用时 t_p 秒

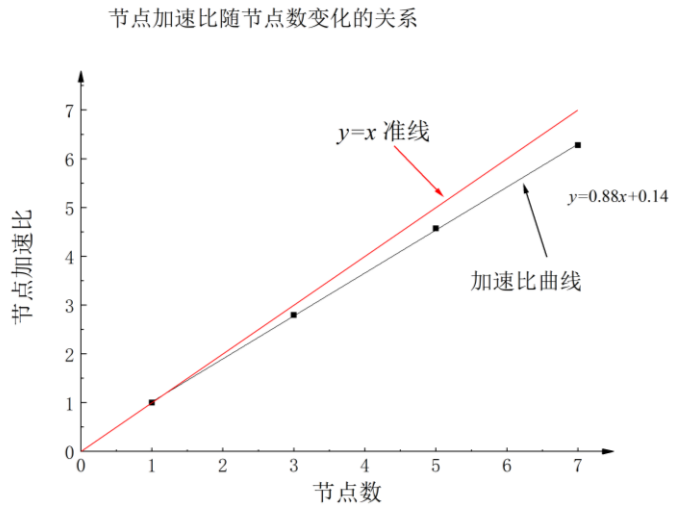
$$N_p = p\text{节点每秒处理事例数} = \frac{24\times p\times eve}{t_p}$$

$$N_1 = \text{单节点每秒处理事例数} = \frac{24\times eve}{t_1}$$

$$S_p = \frac{N_p}{N_1} = \frac{\frac{24\times p\times eve}{t_p}}{\frac{24\times eve}{t_1}} = \frac{pt_1}{t_p},\quad E_p = \frac{S_p}{p} = \frac{t_1}{t_p}$$

加速比曲线就是 S_p 与 p 的关系，化简公式说明测出 p 节点用时即可。

2. 分析加速比曲线



随着 p 增大，加速比曲线 S_p 与 p 有可能不是一直线性，p 非常大时曲线的斜率可能逐渐减少小。

根据加速比曲线可以预测使用一定量的节点并行作业的用时。这对于我们准确预测模拟 100 亿 J/psi 的用时非常重要。

所以现在的方向就是准确测量加速比曲线及其走向。我目前测量到了 7 个节点，与成百上千节点比，这太小太小了。以后的目标：

1. 充分利用 rhenv 分区现有的节点，测量 80 节点以下并行的加速比曲线；
2. 申请使用更多可用节点，测量加速比曲线走向，观测是否有平缓曲线出现；
3. 分析加速比曲线偏离理想曲线（$S_p=p$）的原因，优化提高加速比；
4. 根据修正后的加速比曲线预测模拟 100 亿 *J/psi* 粒子所需的节点数、总时间、失败率和数据传输时间等参数；
5. 根据分配到的节点数量模拟 100 亿 *J/psi* 粒子（约需要 8000 节点），进行模拟作业，传输数据会中科院高能所。

七、预测模拟 100 亿 J/psi 粒子在天河二号的运行情况

上面的测量都是基于模拟 1 万 rhopi 粒子，这是因为此过程时间较短，节约机时。但是最终目标是模拟 100 亿 J/psi，所以建议：

1. 探讨不同模拟脚本的加速比曲线是否相同
2. 如果相同，那就用 rhopi 粒子的结论预测 J/psi
3. 如果不相同，及时切换到 J/psi 的加速比测量

八、有用的 shell 命令等杂记

1.查看 IO 状态

iostat -d -k -x 1 5

2. 查看文件个数

   ls sim_* | grep “^-” | wc -l

3. 查看文件有多少行

   wc -l sim.txt

4. 画直方图

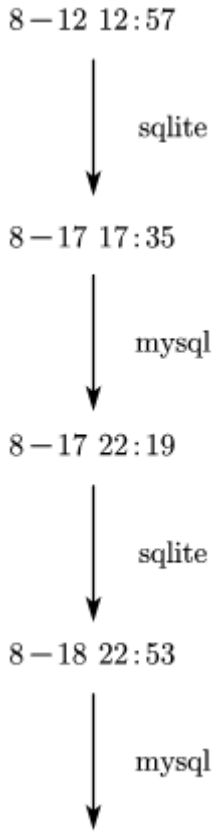
   TH1F h(“右上角长方形”, “标题”，bin 数目，开始，停止)；

   支持科学计数法，3e6　　5e10

5. 查看 CPU 频率

watch -n 0 “cat /proc/cpuinfo | grep -i mhz ”

6. TH-2 数据库切换情况



7. 一般多节点 slurm-2019828.out 会有几百万行，如何挑出来运行时间？

sed -n -e ‘/real/p’ slurm-*.out >> time.txt 将所有含有 real 的行写入 time.txt

计算这些时间的平均值：

cat time.txt | cut -c 6-8 后面的数字表示显示第多少列。

6 和 8 列：6,8

6 到 8 列：6-8

把需要的列显示在屏幕上，然后复制粘贴到 Excel，使用函数计算就行了。

下面的写不下去了，有时间再认真整理吧~~~

8. SQL 语句

增删改查（CURD）

查询 select：

select 字段，字段，* from 表名 where 项 order by 项 limit 项 ;

where 字段 < / = / >

where 字段 in (1,3,7)
where 字段 between 2 and 4
where 字段 1=小明 and/or 字段 2=password
order by 字段 desc/asc
Limit 0 起，3 共

查询总条数：

select count(*)/max(字段)/min(字段)/avg(字段)/sum(字段) as 字段内名
from 表名

添加：

insert into 表名 （字段 1，字段 2，字段 3） values （值 1，值 2，值 3）

修改：

update 表名 set 字段 1=值 1，字段 2=值 2 where 字段 3=值 3

删除：

delete from 表名 where 字段名=字段值

https://www.bilibili.com/video/av23464269

9. mysql 用法

mysql基本操作

- 安装
  参考https://www.cnblogs.com/hyfblog/p/10561379.html 在Windows下面安装。
  Linux安装时root密码搞不定。
- 登录
  mysql -uroot -p
- 库与表基本操作

	查看	新建	使用	删除
库	Show databases	create database 库名	use 库名	drop database 库名
表	show tables	create table 表名	describe 表名	drop table 表名

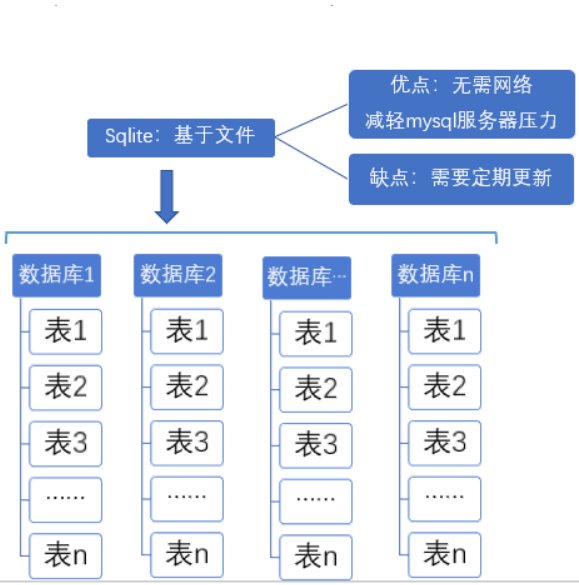
- 数据类型
  tinyint,small int,mediuimt,int,bigint
  float(总位数，小数位数) double (M,D)
  char,varchar
  datetime,date,time,year

- 创建一张表

```
create TABLE (库名).表名 (name VARCHAR(20),
                          sex CHAR(20),
                          age tinyint(3),
                          birth DATE ) ;
```

https://www.cnblogs.com/hyfblog/p/10561379.html

10. sqlite 用法



Sqlite基本命令：

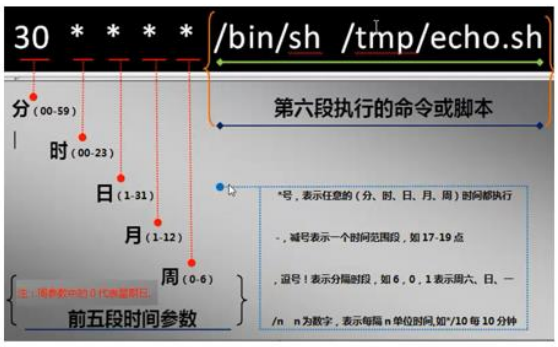
- 创建或打开一个数据库
  sqlite3 mydatabase.db
- 系统命令：
  .help .quit
  .databases 查看所有数据库
  .table 查看当前数据库所有表
  .schema 查看表格结构图
- SQL语句：
  增删改查CURD
  insert;
  delete;
  update;
  select;

11. crontab 用法

cron与crontab：定时执行命令

```
-bash-4.1$ vi cronsub          #写一个
-bash-4.1$ crontab cronsub  #交上去
-bash-4.1$ crontab -l          #看一看
* * * * * echo 'hello world'>>
/scratchfs/bes/zhaoww/crontab/success
-bash-4.1$ ls                  #再看看
cronsub    **
-bash-4.1$ ls                  #成功了
cronsub  success
-bash-4.1$ crontab -r      #删掉吧
-bash-4.1$ crontab -l      #没有了
no crontab for zhaoww
```

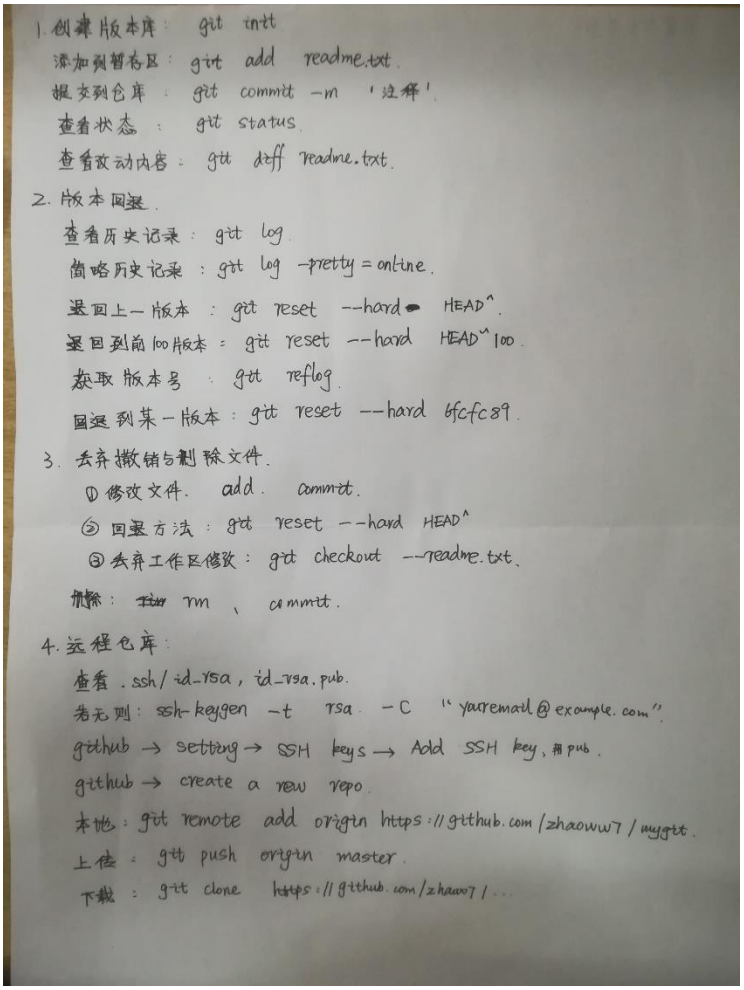
crontab -u //设定某个用户的cron服务
crontab -l //列出某个用户cron服务的详细内容
crontab -r //删除某个用户的cron服务
crontab -e //编辑某个用户的cron服务



https://www.cnblogs.com/zhoul/p/9931664.html
https://blog.csdn.net/gjggj/article/details/72922036

12. valgrind 工具集 memcheck、callgrind、massif 用法

13. git 用法



14. cvs 在 BOSS 的应用

拷贝： cmt co -r TestRelease-00-86 TestRelease

编译： cmt br cmt config

cmt br gmake

source setup.sh

15. docker 用法

见 https://github.com/zhaoww7/IHEPlearn/pre_in_ihep

九、老师的咨询邮箱

中科院高能所：

软件组：张瑶老师：zhangyao@ihep.ac.cn

马秋梅老师：maqm@ihep.ac.cn

王亮亮老师：llwang@ihep.ac.cn

J/psi 粒子模拟：孙永昭老师：sunyz@ihep.ac.cn

计算中心：邹佳恒老师：zoujh@ihep.ac.cn

郑伟老师：zhengw@ihep.ac.cn

中山大学：

唐健老师：tangjian5@mail.sysu.edu.cn

赵问问（本科生）：zhaoww2013@126.com

天河二号：

陈璟锟老师：jingkun.chen@nscc-gz.cn

技术支持团队：techsupport@nscc-gz.cn

其他：

俄罗斯：zhemchugov@jinr.ru

十、开源网页

在高能所周四组会汇报了五次，相关 PPT：https://github.com/zhaoww7/IHEPlearn