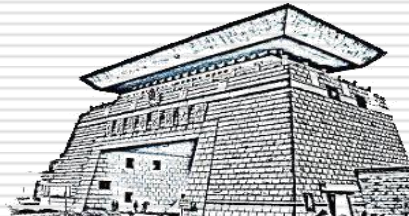
广州先导系统用户培训

资源管理系统

国防科学技术大学计算机学院



培训内容

- 系统概述
  - 天河高性能计算机结构
  - 资源管理系统组成
- 系统使用
  - 资源分配
  - 任务加载
  - 状态查看
  - 作业控制
- 系统上机流程简介

I. 系统概述

- 天河高性能计算机结构
- 资源管理系统组成
- 资源管理系统实体

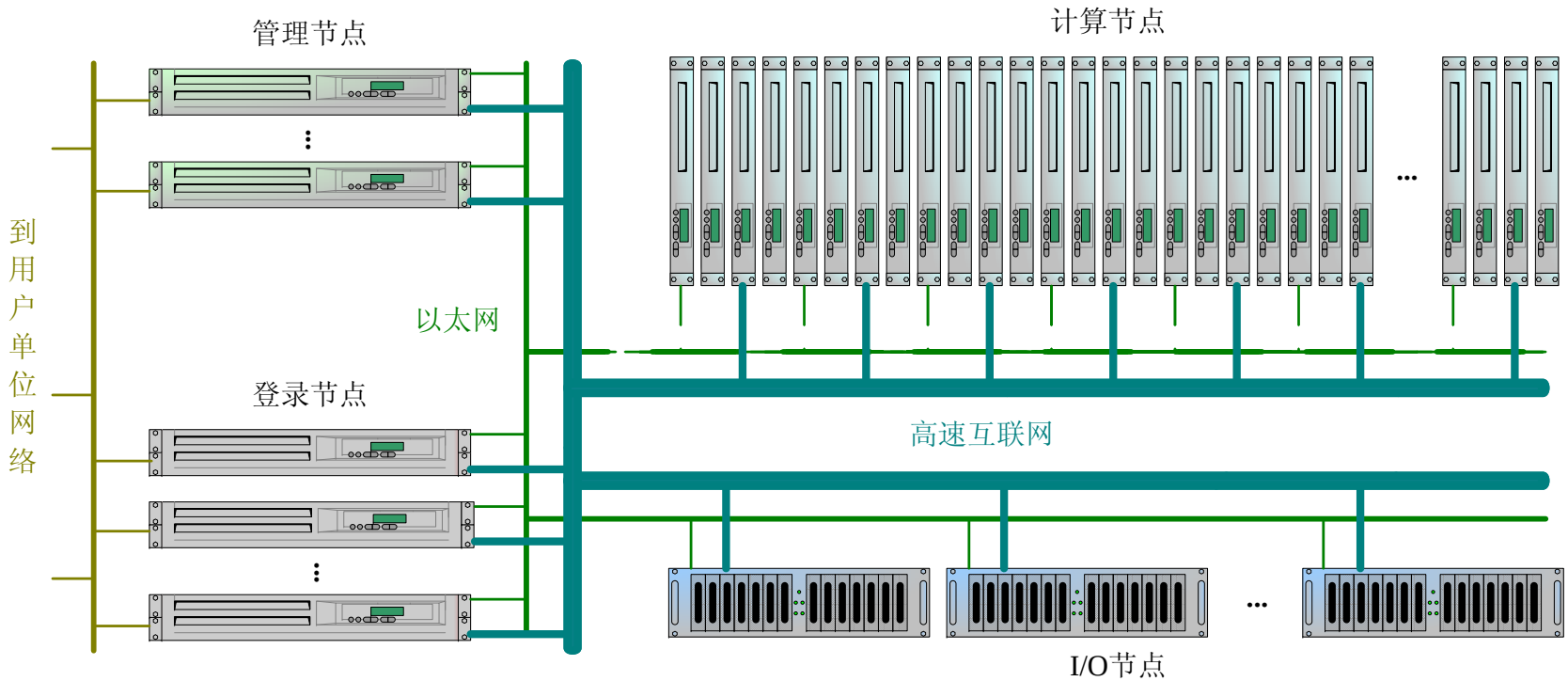
天河高性能计算机

组成

- 计算处理
- 互联通信
- I/O 存储
- 基础架构
- 监控诊断
- 操作系统
- 编译器
- 运行环境
- 应用软件

天河高性能计算机

逻辑结构



天河高性能计算机

逻辑结构

管理节点

- mn0，mn1，···
- 运行系统管理进程与支撑服务

登录节点

- ln0，ln1，···
- 用户登录，编辑、编译、提交作业、结果分析

天河高性能计算机

逻辑结构

计算节点

- cn0，cn1，···
- 主要的计算资源，执行用户程序

I/O 节点

- 元数据服务器：mds0，mds1
- 对象存储：ost0，ost1，...
- 提供存储服务
- 表现为全局共享文件系统

资源管理系统

- 操作系统的重要部分

提供高效的资源与作业管理

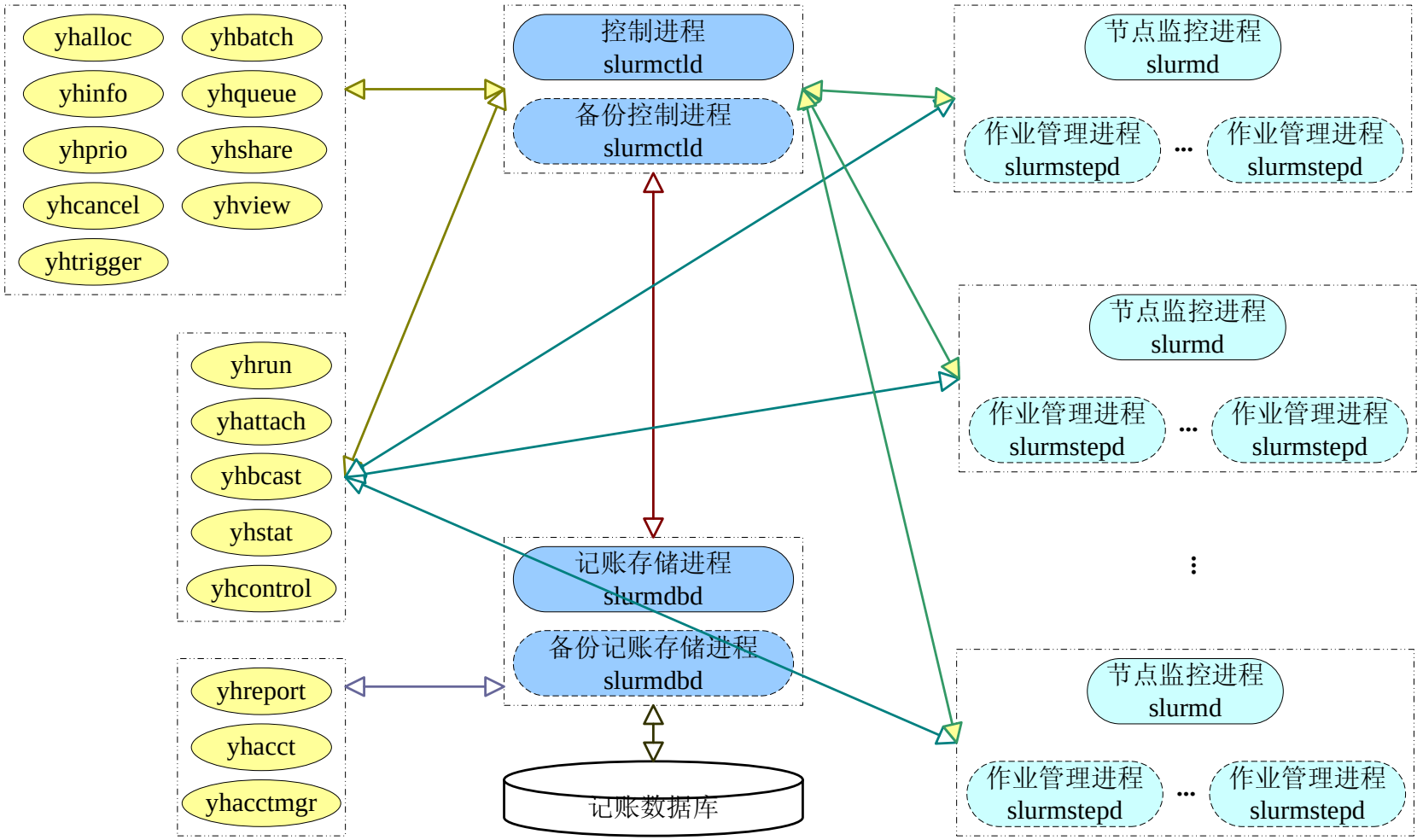
- 节点状态监控
- 分区管理
- 作业调度
- 资源预约
- 能耗管理
- 作业记账

是用户使用计算资源的接口

- 作业提交 / 运行
- 任务加载
- 作业控制
- 状态查看
- 事件触发器

资源管理系统

组成结构


yhalloc
yhbatch
yhinfo
yhqueue
yhprio
yhshare
yhcancel
yhview
yhtrigger
yhrun
yhattach
yhbcast
yhstat
yhcontrol
yhreport
yhacct
yhacctmgr
控制进程
slurmctld
备份控制进程
slurmctld
记账存储进程
slurmdbd
备份记账存储进程
slurmdbd
记账数据库
节点监控进程
slurmd
作业管理进程
slurmstepd
...
节点监控进程
slurmd
作业管理进程
slurmstepd
...
节点监控进程
slurmd
作业管理进程
slurmstepd
...

控制进程

- 运行在管理节点
- 是资源管理系统的控制中枢
- 记录节点状态
- 进行分区管理
- 进行作业管理、作业调度、资源分配

记账存储进程

- 运行在管理节点
- 将作业信息保存到数据库
- 记录用户、帐号、资源限制、QOS 等信息
- 用户认证和安全隔离

节点监控进程

- 运行在每个计算节点
- 监控节点状态，并向控制进程注册
- 接收来自控制进程与用户的请求并进行处理

作业管理进程

- 加载计算任务时由节点监控进程启动
- 管理一个作业步的所有任务
  - 启动计算任务进程
  - 标准 I/O 转发
  - 信号传递
  - 任务控制
  - 资源使用信息收集

命令工具

- yhacct：查看历史作业信息
- yhacctmgr：记账管理
- yhalloc：资源分配
- yhbatch：提交批处理作业
- yhcancel：取消作业
- yhcontrol：系统控制
- yhinfo：节点与分区状态查看
- yhqueue：队列状态查看
- yhrun：任务加载

资源管理系统

实体

节点

- 即指计算节点
- 包含处理器、内存、磁盘空间等资源
- 具有空闲、分配、故障等状态
- 使用节点名字标识

分区

- 节点的逻辑分组
- 提供一种管理机制，可设置资源限制、访问权限、优先级等
- 分区可重叠，提供类似于队列的功能
- 系统有一个默认分区
- 使用分区名字标识

资源管理系统

实体

作业

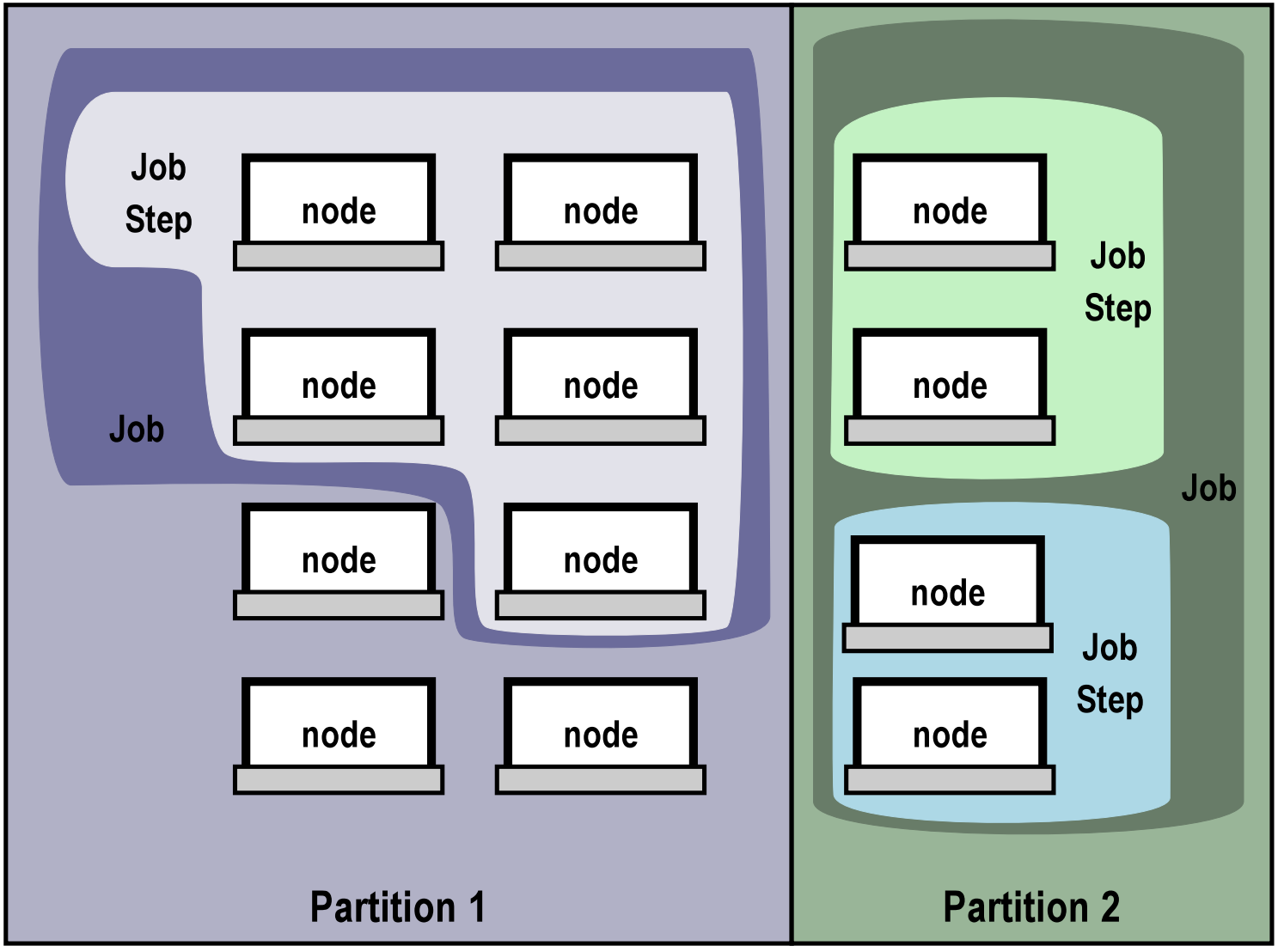
- 一次资源分配
- 位于一个分区中，作业不能跨分区
- 排队调度后分配资源运行
- 通过作业 ID 标识，如 123

作业步

- 通过 `yhrun` 进行的任务加载
- 作业步可只使用作业中的部分节点
- 一个作业可包含多个作业步，可并发运行
- 在作业内通过作业步 ID 标识，如 123.0

资源管理系统

实体


Job
Step
node
node
node
node
Job
node
node
node
node
Partition 1
node
Job
Step
node
Job
node
Job
Step
node
Partition 2

II. 系统使用

- 查看系统状态
- 分配资源
- 加载计算任务
- 作业控制

系统状态查看

内容

- 节点状态
- 分区状态
- 作业状态
- 作业步状态

节点状态

状态监控机制

- 节点状态由控制进程维护
- 控制进程使用三种机制检查节点状态
  - ping：仅检查通信状态
  - register：报告资源状态
    - `slurmd` 启动时主动进行
    - 节点多时，周期较长
  - health check：管理员定制脚本
- 命令工具从控制进程获取节点状态

节点状态

```
$ yhinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
work         up   infinite   1110  down* cn[0-451,494-1151]
work         up   infinite     42   idle cn[452-493]
2pao*        up   infinite      9  idle* cn[1161-1169]
2pao*        up   infinite      7  down* cn[1178-1179,1224-1225,1244-1245,1259
2pao*        up   infinite    112   idle cn[1152-1160,1170-1177,1180-1223,1226
```

节点详细信息

```
$ yhcontrol show node cn0
NodeName=cn0 Arch=x86_64 CoresPerSocket=1
    CPUAlloc=0 CPUErr=0 CPUTot=8 Features=(null)
    OS=Linux RealMemory=1 Sockets=8
    State=DOWN* ThreadsPerCore=1 TmpDisk=0 Weight=1
    Reason=Not responding [slurm@2010-03-15T15:17:11]
```

节点状态

状态值

基本状态

- UNKNOWN：未知，unk
- IDLE：空闲，idle
- ALLOCATED：已分配，alloc
- DOWN：故障，down

状态标志

- DRAIN：不再分配，drng/drain
- COMPLETING：有作业正在退出，comp
- NO_RESPOND：无响应，*

分区状态

状态查看

显示分区状态

```
$ yhinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
work         up   infinite   1110  down* cn[0-451,494-1151]
work         up   infinite     42   idle cn[452-493]
2pao*        up   infinite      7  down* cn[1178-1179,1224-1225,1244-1245,1259
2pao*        up   infinite    121   idle cn[1152-1177,1180-1223,1226-1243,1246
```

查看分区详细信息

```
$ yhcontrol show partition work
PartitionName=work
    AllocNodes=ALL AllowGroups=ALL Default=NO
    DefaultTime=NONE DisableRootJobs=NO Hidden=NO
    MaxNodes=UNLIMITED MaxTime=UNLIMITED MinNodes=1
    Nodes=cn[0-1151]
    Priority=1 RootOnly=NO Shared=NO
    State=UP TotalCPUs=9216 TotalNodes=1152
```

分区状态

分区属性

- 节点列表
- 状态：UP/DOWN
- 隐藏分区
- 访问权限
  - RootOnly
  - AllowGroups
- 资源限制
  - 节点范围
  - 运行时间
- 优先级
- 共享节点
- 默认分区

显示队列状态

```
$ yhqueue
JOBID PARTITION   NAME USER ST TIME NODES NODELIST(REASON)
 1463      2pao sbatch root  R 1:06    12 cn[1246-1257]
 1465      work   tjob test PD 0:00    66 (PartitionNodeLimit)
 1464      work  myjob root  R 0:32    23 cn[452-474]
```

- yhqueue 默认只显示排队、运行和退出过程中的作业
- 作业结束一段时间后，信息将从 slurmctld 中清除

显示作业详细信息

```
$ yhcontrol show job 123
JobId=1464 Name=myjog
    UserId=root(0) GroupId=root(0)
    Priority=2 Account=(null) QOS=normal
    JobState=RUNNING Reason=None Dependency=(null)
    TimeLimit=UNLIMITED Requeue=1 Restarts=0 BatchFlag=1 ExitCode=0:0
    SubmitTime=2010-03-16T08:24:34 EligibleTime=2010-03-16T08:24:34
    StartTime=2010-03-16T08:24:34 EndTime=NONE
    SuspendTime=None SecsPreSuspend=0
    Partition=work AllocNode:Sid=ln0:8116
    ReqNodeList=(null) ExcNodeList=(null)
    NodeList=cn[452-474]
    NumNodes=23 NumCPUs=23 CPUs/Task=1 ReqS:C:T=1:1:1
    MinCPUsNode=1 MinMemoryNode=0 MinTmpDiskNode=0
    Features=(null) Reservation=(null)
    Shared=OK Contiguous=0 Licenses=(null) Network=(null)
    Command=(null)
    WorkDir=/vol5
```

历史作业信息

```
$ yhacct
JobID    JobName  Partition  AllocCPUS      State ExitCode
-------- ---------- --------- --------- --------- --------
 1449   hostname       2pao            0   COMPLETE      0:0
 1450     ft.B.8       work            0   COMPLETE      0:0
 1451     ft.B.8       work            0   COMPLETE      0:0
 1452     ft.B.8       work            0  CANCELLED      0:0
 1453        env       2pao            0   COMPLETE      0:0
 1454    memlock       2pao            0    PENDING      0:0
 1457   hostname       2pao            0   COMPLETE      0:0
 1458      STACK       2pao            0    PENDING      0:0
 1459   hostname       2pao            0   COMPLETE      0:0
 1462       bash       2pao            0    PENDING      0:0
```

作业状态

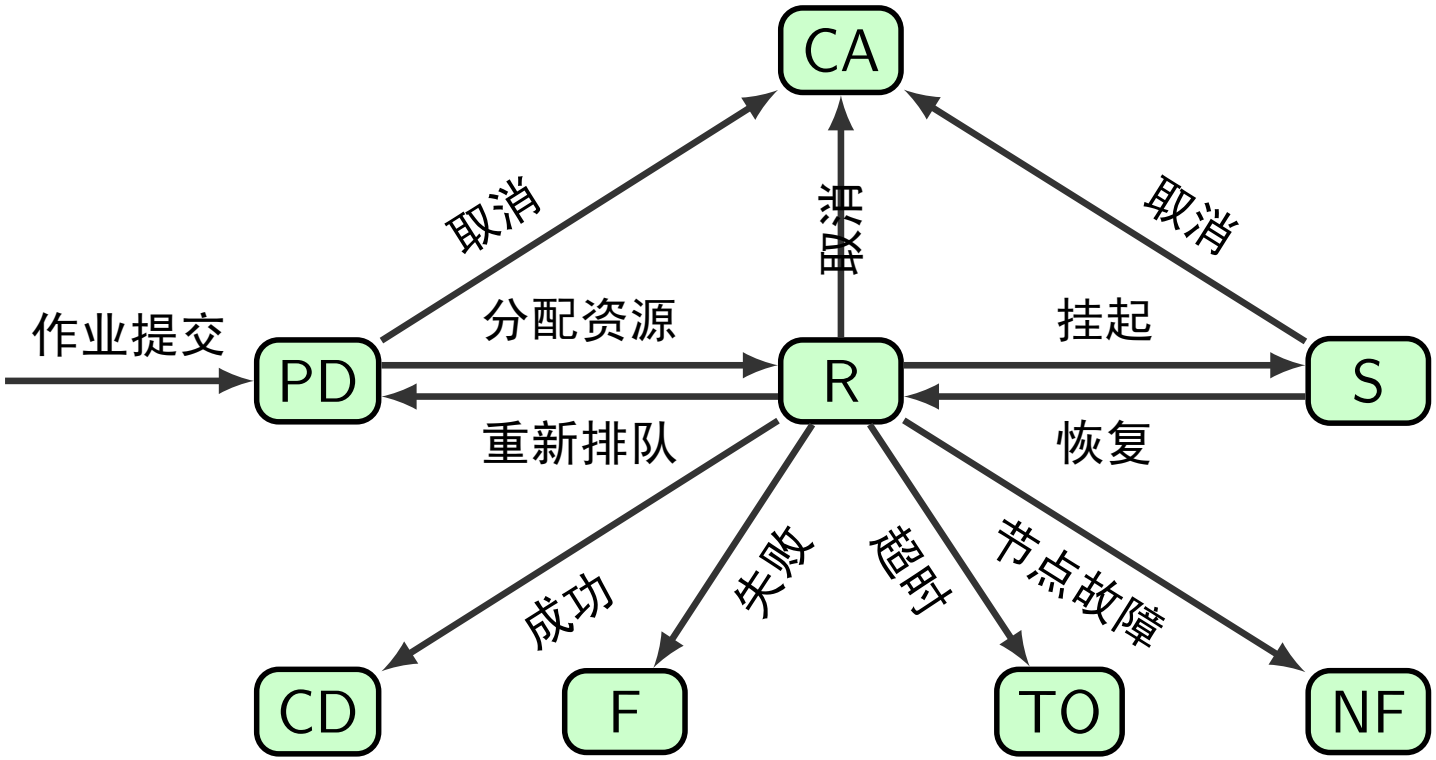
状态值

- PENDING：排队，PD
- RUNNING：运行，R
- SUSPENDED：挂起，S
- COMPLETED：成功结束，CD
- FAILED：失败结束，F
- CANCELLED：被取消，CA
- TIMEOUT：超时，TO
- NODE_FAIL：因节点故障而运行失败，NF

CD、F、CA、TO、NF 都是运行结束的状态

作业状态

状态转换



作业状态

状态标志

- COMPLETING：正在退出，CG
- CONFIGURING：分配给作业的节点正在启动，CF

- 作业 CG 与节点 comp 相对应

作业状态

状态原因

排队状态

- Priority：优先级不够高
- Dependency：作业的依赖关系未满足
- Resources：当前可用资源不能满足作业需求
- PartitionNodeLimit：作业请求的节点数超过分区的作业节点数限制
- PartitionTimeLimit：作业请求的运行时间超过分区的作业运行时间限制
- PartitionDown：作业所在的分区处于 DOWN 状态
- JobHeld：作业被阻止调度

作业状态

状态原因

排队状态

- BeginTime：作业请求的启动时间还未到达
- AssociationJobLimit：关联的作业限制已满
- AssociationResourceLimit：关联的资源限制已满
- AssociationTimeLimit：关联的运行时间限制已满
- Reservation：作业请求的预约时间未到
- ReqNodeNotAvail：作业请求的节点不可用

结束状态

- PartitionDown：作业所在的分区被删除
- NodeDown：分配给作业的节点进入 DOWN 状态
- BadConstraints：作业的资源约束无效
- SystemFailure：系统故障
- JobLaunchFailure：作业加载故障
- NonZeroExitCode：作业的退出代码非 0
- TimeLimit：作业超出运行时间限制
- InactiveLimit：作业超出不活跃时间限制
- InvalidBankAccount：作业的计费帐号无效

作业步状态

- 作业步仅在运行时存在
- 仅由 yhrun 加载的任务产生作业步

查看状态

```
$ yhqueue -s
STEPID          NAME PARTITION     USER      TIME NODELIST
1466.0         sleep      2pao     root      0:07 cn[1246-1255]

$ yhcontrol show step 1466.0
StepId=1466.0 UserId=0 StartTime=2010-03-16T08:41:49 TimeLimit=UNLIMITED
    Partition=2pao Nodes=cn[1246-1255] Tasks=10 Name=sleep Network=(null)
    ResvPorts=(null) Checkpoint=0 CheckpointDir=/vol5
```

作业与资源分配

内容

- 作业运行模式
- 作业的资源需求
- 作业的运行参数
- 作业的环境变量

作业 = 资源分配请求

- 提交：申请资源
- 排队：等待资源
- 运行：分配资源（无论是否执行程序）
- 挂起：暂时释放资源
- 结束：释放资源

作业运行模式

- 交互模式
- 批处理模式
- 分配模式

- 只是用户使用方式区别
- 管理、调度、记账时同等对待

交互模式作业

1. 在终端提交资源分配请求
2. 等待资源分配
3. 加载计算任务
4. 运行中，任务 I/O 传递到终端
5. 可与任务进行交互：I/O，信号
6. 任务执行结束后，资源被释放

- 一个作业（一次资源分配）包含一个作业步（一次任务加载）

交互模式作业

- 通过 yhrun 命令运行

例

```
$ yhrun -n 4 cpi
Enter the number of intervals: (0 quits) 1
pi is approximately 3.2000000000000002, Error is 0.0584073464102071
wall clock time = 0.000047
Enter the number of intervals: (0 quits) 2
pi is approximately 3.1623529411764704, Error is 0.0207602875866773
wall clock time = 0.000022
Enter the number of intervals: (0 quits) 3
pi is approximately 3.1508492098656031, Error is 0.0092565562758100
wall clock time = 0.000014
Enter the number of intervals: (0 quits) 0
$
```

批处理模式作业

1. 用户编写作业脚本
2. 提交作业
3. 作业排队等待资源分配
4. 在首节点加载执行作业脚本
5. 脚本执行结束，释放资源
6. 用户在输出文件中查看运行结果

- 脚本中可通过 `yhrun` 加载计算任务
- 一个作业可包含多个作业步

批处理模式作业

作业脚本

- 作业脚本为文本文件，首行以“#!”开头，指定解释程序
- 可包含各种合适命令
- 可用 yhrun 在分配的节点上加载任务
- 脚本在计算节点上执行
- 默认地，脚本的输出写到文件中

例

```
#!/bin/bash
cp data input
yhrun a.out
cp output result
yhrun b.out
```

批处理模式作业

- yhbatch 命令提交作业

例

```
$ cat job.sh
#!/bin/sh
yhrun -n 8 hostname
$ yhbatch -N 2 job.sh
Job 123 submitted
$ yhqueue
JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  123      work   job.sh     root   R       0:01      2 cn[0-1]
$ cat slurm-123.out
cn0
cn0
cn0
...
```

分配模式作业

1. 提交资源分配请求
2. 作业排队等待资源分配
3. 执行用户指定的命令
4. 命令执行结束，释放资源

- 交互模式作业与批处理模式作业的结合
- 一个作业可包含多个作业步
- 可通过 yhrun 加载计算任务
- 可与任务进行交互
- 命令在用户提交作业的节点上执行

分配模式作业

- yhalloc 命令提交分配模式作业

例

```
$ yhalloc -N 16 /bin/bash
yhalloc: Granted job allocation 1481
$ yhrun -n 16 a.out
...
$ yhrun -n 32 b.out
...
$ exit
exit
yhalloc: Relinquishing job allocation 1481
$
```

作业的资源需求

资源数量

节点数量

- `-N,--nodes` 选项指定
- *min*[-*max*]
- 如未指定，则根据其他需求，分配足够的节点

处理器数量

- 由几个参数决定
  - 作业要加载的任务数 `-n,--ntasks`，默认每个节点一个
  - 每个任务需要的处理器数 `-c,--cpus-per-task`，默认为 1
- 系统将根据参数计算，分配足够处理器数目的节点

节点与处理器数目约束

```
$ yhrun -n 8 -l hostname
0: cn1246
2: cn1246
6: cn1246
1: cn1246
3: cn1246
7: cn1246
4: cn1246
5: cn1246
```

节点与处理器数目约束

```
$ yhrun -N 4 -n 8 -l hostname
0: cn1246
2: cn1247
4: cn1248
6: cn1249
1: cn1246
3: cn1247
5: cn1248
7: cn1249
```

物理内存

- --mem 指定每个节点上使用的物理内存
- --mem-per-cpu 指定所分配的每个处理器需要的物理内存
- 资源分配时考虑节点上可用的内存数量
- 超出内存限制的作业将被终止

指定物理内存数目

```
$ yhrun -N 4 --mem 2048 -n 8 a.out
```

运行时间

- `-t,--time` 选项指定
- 超出时间限制的作业将被终止
- 应尽可能准确估计：调度时用此估计时间进行 backfill 判断

例

```
$ yhrun -N 4 -n 8 -t 100 a.out
```

作业的资源需求

节点属性

分区

- `-p,--partition` 选项指定
- 从指定分区中分配节点
- 使用指定分区的资源限制 / 访问权限进行检查
- 作业必须位于一个分区中，不能跨分区

预约

- `--reservation` 指定
- 从指定的预约中分配节点
- 作业须在预约结束之前终止，否则超时

作业的资源需求

节点属性

指定节点

- `-w,--nodelist`：分配给作业的节点中至少包含指定节点
- `-F,--nodefile`：分配给作业的节点中至少包含指定文件中列出的节点
- `-x,--exclude`：分配给作业的节点中不要包含指定节点

指定节点

```
$ yhrun -w cn[0-3] -N 8 a.out

$ yhrun -x cn[0-3] -N 8 a.out
```

作业的资源需求

节点属性

处理器特征

- `--mincpus`：分配给作业的节点上至少具有的 CPU 数目
- `--sockets-per-node`：分配给作业的节点，每节点至少的 socket 数
- `--cores-per-socket`：分配给作业的节点，每 socket 至少的 core 数
- `--threads-per-core`：分配给作业的节点，每 core 至少的 thread 数
- `-B,--extra-node-info`：同时指定三者，$S[:C[:T]]$

处理器特征

```
$ yhrun -B 4:4 -n 32 a.out
```

作业的运行参数

作业名字

- 默认：加载的程序 / 批处理脚本文件名 / 执行的命令
- -J,--job-name：指定名字

作业名字

```
$ yhbatch -N 4 job.sh
$ yhbatch -N 8 -J myjob job.sh
$ yhqueue
JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
 1234      work   job.sh     test   R      00:05      4 cn[0-3]
 1235      work    myjob     test   R      00:03      8 cn[12-19]
$ yhcancel -J myjob
```

作业的运行参数

启动时间

- `--begin`：作业在指定时间之后才能运行

依赖关系

- `-d,--dependency`：指定作业的依赖关系
  - `after:`*jobid*：在指定作业开始之后
  - `afterok:`*jobid*：在指定作业成功结束之后
  - `afternotok:`*jobid*：在指定作业不成功结束之后
  - `afterany:`*jobid*：在指定作业结束之后
- 满足依赖关系的作业才能运行
- 不可能满足依赖关系的作业将被取消

作业的运行参数

工作目录

- -D,--chdir：指定任务 / 脚本 / 命令的工作目录
- 默认：yhrun/yhbatch/yhalloc 的工作目录

例

```
$ yhbatch -N 4 -D /home/test/devel/bin job.sh
```

作业的运行参数

节点故障容忍

- 默认：节点失效时将终止作业
  - 失效指节点变为 DOWN 状态
  - 主要针对 MPI 程序的执行，及时释放资源
- `-k,--no-kill`：容忍节点故障
  - 程序自身容错
  - 正在执行的作业步失败后，继续运行后续作业步

例

```
$ yhbatch -N 128 -t 1000 -k job.sh
```

作业的环境变量

- 系统在运行计算任务 / 作业脚本 / 命令时，会为其设置一些环境变量，以反应其资源分配情况
- 在批处理和分配模式作业中，可根据环境变量获得资源分配情况

作业环境变量

- SLURM_JOB_ID：作业 ID
- SLURM_JOB_NUM_NODES：作业分配的节点数
- SLURM_JOB_NODELIST：作业分配的节点列表
- SLURM_JOB_CPUS_PER_NODE：作业分配的每节点 CPU 数
- HOSTNAME：对于批处理作业，此变量被设置为批处理脚本所执行节点的节点名（不是节点主机名）
- SLURM_NPROCS：要加载的任务数
- 等等

作业步与任务加载

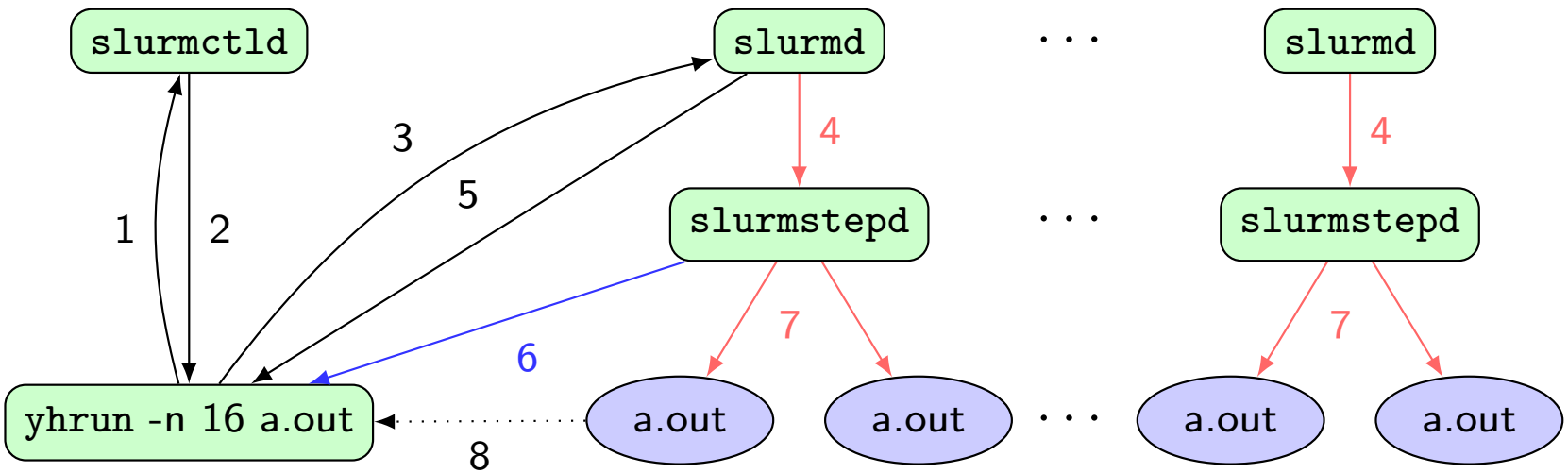
内容

- 作业步任务
  - 加载流程
  - 任务状态查看
  - 任务 I/O 控制
  - 信号传递
  - 执行环境
  - 任务布局
  - 多程序作业步
  - 作业步终止
- 登录计算节点
- MPI 程序加载

作业步任务

加载流程

- 命令：yhrun [*options*] *program* [*args*]



1. 创建作业步　2. 响应：任务布局 / 证书　3. 加载任务　4. 派生作业管理进程
5. 加载响应　6. 建立 I/O 连接　7. 派生任务　8. PMI 连接

- 任务加载后，对一秒钟内第一次 Ctrl+C，yhrun 显示任务状态

任务的可能状态

- failed to start：加载失败
- running：正在运行
- exited：退出，exit code 为 0
- exited abnormally：退出，exit code 非 0
- unknown：还未收到节点的加载响应

显示任务状态

```
$ yhrun -n 8 ft.B.8
NAS Parallel Benchmarks 3.3 -- FT Benchmark
No input file inputft.data. Using compiled defaults
...
T =    1     Checksum =    5.177643571579D+02    5.077803458597D+02
T =    2     Checksum =    5.154521291263D+02    5.088249431599D+02
yhrun: interrupt (one more within 1 sec to abort)
yhrun: tasks 0-7: running
yhrun: sending Ctrl-C to job 1434.0
forrtl: error (69): process interrupted (SIGINT)
Image              PC                Routine            Line        Source
ft.B.8             000000000040863B  Unknown               Unknown  Unknown
...
```

I/O 传递

- 默认地，程序的标准输入 / 输出 / 错误从 / 到 yhrun
- 可通过 -i,--input/-o,--output/-e,--error 选项控制
  - all：默认，传递所有任务 I/O
  - none：不传递
  - *taskid*：仅传递指定任务的 I/O
  - *filename*：将任务的 I/O 写入文件
    - 支持变量替换，节点名 / 任务号 / 局部任务号等

例

```
$ yhrun -o result -n 4 hostname
$ yhrun -i 0 -n 16 a.out
```

任务标号

- 区分输出 / 错误信息由哪个任务给出
- `-l,--label` 选项

例

```
$ yhrun -N 2 hostname
cn1247
cn1246

$ yhrun -l -N 2 hostname
0: cn1246
1: cn1247
```

伪终端

- `--pty` 选项，可在伪终端中运行 0 号任务
- 将仅从 0 号任务传递 I/O
- 不能指定 `-i/-o/-e`

伪终端示例

```
$ yhrun -w cn0 ls
anaconda-ks.cfg
install.log
install.log.syslog
l.tar
lustre-1.8.1.1

$ yhrun --pty -w cn0 ls
anaconda-ks.cfg  install.log  install.log.syslog  l.tar  lustre-1.8.1.1
```

伪终端示例

```
$ yhrun -w cn0 top
yhrun: error: cn0: task 0: Exited with exit code 1
top: failed tty get

$ yhrun --pty -w cn0 top
top - 10:51:00 up  1:54,  2 users,  load average: 0.00, 0.02, 0.08
Tasks: 179 total,   1 running, 178 sleeping,   0 stopped,   0 zombie
Cpu(s):  1.3%us,  0.5%sy,  0.1%ni, 97.4%id,  0.5%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  32947432k total,  7666408k used, 25281024k free,   166060k buffers
...
```

附接 I/O

- 在作业步运行中，可用 yhattach 命令附接到任务 I/O
- 常用于查看批处理作业中的作业步的输出

示例

```
$ yhqueue -s
STEPID          NAME PARTITION     USER      TIME NODELIST
1452.0       ft.B.8      work     root      0:02 cn452
$ yhattach 1452.0
Layout type          :              1D
T =     1     Checksum =    5.177643571579D+02    5.077803458597D+02
T =     2     Checksum =    5.154521291263D+02    5.088249431599D+02
^C
$
```

yhrun 的信号处理

- 加载任务前：标准行为
- 加载任务后
  - SIGINT：一秒内首次将显示任务状态
  - SIGQUIT：强制终止作业步
  - SIGCONT：忽略
  - SIGUSR1/SIGUSR2/SIGALRM：传递到所有任务

发送信号

- `yhcancel` 可以向所有任务发送信号

环境变量

- yhrun 的环境变量将传递到任务进程
- 系统为任务进程设置环境变量，以反应其分配资源、执行环境等

传递环境变量

```
$ abc=123 yhrun env | grep abc
abc=123

$ export xyz=456
$ yhrun env | grep xyz
xyz=456
```

系统设置环境变量

```
$ yhrun env | grep SLURM
SLURM_PRIO_PROCESS=0
SLURM_SUBMIT_DIR=/root
SLURM_JOB_ID=1453
SLURM_NODELIST=cn1258
SLURM_TASKS_PER_NODE=1
SLURM_SRUN_COMM_PORT=46027
SLURM_STEP_ID=0
SLURM_STEP_NODELIST=cn1258
SLURM_STEP_NUM_NODES=1
SLURM_STEP_NUM_TASKS=1
SLURM_NNODES=1
...
```

作业步任务

执行环境

资源限制

- 可以将 yhrun 的资源限制（ulimit）传递到任务进程
- --propagate 选项设置传递的资源限制
- 默认的传递项在配置文件中设置

- NONE
- ALL
- CPU
- FSIZE
- DATA
- STACK
- CORE
- RSS
- NPROC
- NOFILE
- MEMLOCK
- AS

示例

```
$ yhrun --propagate=MEMLOCK a.out
```

工作目录

- 默认为 yhrun 进程的工作目录
  - 前提：计算节点上目录存在、可访问
- 通过 --chdir 选项设置

示例

```
$ pwd
/home/test
$ yhrun pwd
/home/test
$ yhrun -D /tmp pwd
/tmp
```

作业步任务

任务布局

- 任务布局指任务在所分配节点上的分布
  - 任务数分布
  - 任务号分布
- 支持多种布局方式：`-m,--distribution`
  - 循环布局
  - 块布局
  - 基于面的布局
  - 任意布局
- 可用于性能调优
  - 节点的负载
  - 任务间通信

循环布局

- `-m cyclic`
- 在分配的处理器数目范围内，尽可能将任务在节点间平均分配

示例

设作业步使用 4 个节点，cn[0,3] 四个处理器，cn[1-2] 两个处理器

cn0	cn1	cn2	cn3
0	1	2	3
4	5	6	7
8			9
10			11
12	13	14	15

先按可用处理器在节点间轮转分配任务

处理器全部分配，按节点轮转分配任务

...

块布局

- `-m block`
- 各节点任务数与循环布局相同，仅任务编号不同
  - 由横着排变成竖着排

示例

16 个任务

cn0	cn1	cn2	cn3
0	5	8	11
1	6	9	12
2			13
3			14
4	7	10	15

10 个任务

cn0	cn1	cn2	cn3
0	3	5	7
1	4	6	8
2			9

基于面的布局

- `-m plane=`*size*
- 是一种块循环布局，块大小为 *size*
- 不考虑负载

示例：4 节点×4 处理器，加载 16 任务

循环布局

cn0	cn1	cn2	cn3
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

块布局

cn0	cn1	cn2	cn3
0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

面布局，块大小为 3

cn0	cn1	cn2	cn3
0	3	6	9
1	4	7	10
2	5	8	11
12	15		
13			
14			

任意布局

- -m arbitrary
- --nodefile 指定节点列表文件
  - 每个节点一行
  - 行数与要加载的任务数相同

示例

```
$ cat nodefile
cn0     cn2     cn0     cn3
cn1     cn4     cn0     cn2

$ yhrun -n 8 -m arbitrary -F nodefile a.out
```

作业步任务

多程序作业步

- 支持 MPMD 程序的运行，即不同任务号执行不同程序
- `--multi-prog` 选项
- `yhrun` 最后跟配置文件，而不是可执行程序命令

配置文件格式

- 按行组织，每行分为若干空白分隔的字段
- 第一字段：任务号部分
  - 逗号分隔的任务号列表
  - 可包含范围 *min-max*
  - “*”表示其余所有任务
- 第二字段：要执行的程序
- 其余字段：执行程序的参数

作业步任务

多程序作业步

配置文件格式

- 在程序和参数部分，支持变量替换
  - %t：任务号
  - %o：在本行的偏移

示例

```
$ cat mp.conf
0         a.out abc
1         b.out %t
2,5,7-9   c.out %o
*         d.out
```

各任务执行的程序

```
$ yhrun --multi-prog -n 16 mp.conf
```

- 0：a.out abc
- 1：b.out 1
- 2：c.out 0
- 3,4,6,10-15：d.out
- 5：c.out 1
- 7：c.out 2
- 8：c.out 3
- 9：c.out 4

作业步任务

作业步终止

- 作业步所有任务执行结束后，作业步终止
- 可指定当有任务退出时，强制终止作业步前等待的时间
  - `-W,--wait` 选项
  - 系统配置默认的等待时间
- 可指定当有任务异常，强制终止作业步
- `yhcancel` 可取消作业步的执行

有任务提前退出时，30 秒后终止作业步

```
$ yhrun -W 30 -n 128 a.out
```

取消作业步

```
$ yhcancel 123.0
```

登录计算节点

- 用户分配资源后，可使用 SSH 登录所分配的节点
- 可用于查看节点上进程状态、运行程序等

MPI 程序加载

MPICH2 及派生 MPI

- 资源管理系统中集成了 PMI 实现
- 对使用 PMI 接口进行进程管理的 MPI 实现，可利用 `yhrun` 直接加载
  - 包括 MPICH2、MVAPICH2、YH-MPI 等
  - 编译配置 MPI 时需指定 `--with-pmi=slurm`

MPI 程序加载

Open MPI

- Open MPI 支持使用 `yhrun` 加载任务
- 编译配置时需指定 `--with-slurm`
- 加载的是 orted，而不是直接加载任务
- 产生作业步，但名字为 orted

MPI 程序加载

通用加载方法

- 利用资源管理系统分配资源
  - `yhalloc`
  - `yhbatch`
- 使用 MPI 自带的 mpiexec 加载计算任务
  - 进行修改或包装
  - 从环境变量中获取分配的节点、要加载的任务等

作业控制

内容

- 取消作业
- 发送信号
- 挂起与恢复
- 重排队
- 修改作业

作业控制

作业取消

- yhcancel 命令取消作业 / 作业步
- 排队作业：标记为 CANCELLED 状态
- 运行 / 挂起作业：终止所有作业步；标记为 CANCELLED 状态；回收资源

示例

```
$ yhcancel 123 456
$ yhcancel 789.1
# yhcancel -u test
# yhcancel -p debug -t pd
```

- 系统将定期重复发送 SIGKILL 到作业步任务，直到其退出
- 显示为 CG 状态的作业已经结束，不用再取消，yhcancel 也不能去掉其 COMPLETING 标志

作业控制

信号发送

- `yhcancel` 可给作业 / 作业步发送信号
  - 作业步：发送到所有任务
  - 作业：发送到所有正在运行的作业步的所有任务
  - `-b,--batch` 指定发送信号到批处理脚本
- `-s,--signal` 指定信号名字或数值

- HUP
- INT
- QUIT
- ABRT
- KILL
- ALRM
- TERM
- USR1
- USR2
- CONT
- STOP
- TSTP
- TTIN
- TTOUT

示例

```
$ yhcancel -s usr1 123.0
$ yhcancel -u test -s stop
$ yhcancel -b -s 12 456
```

作业控制

挂起与恢复

- 挂起：暂时释放处理器资源
  - 向任务发送 SIGTSTP 和 SIGSTOP 信号
  - 节点被释放，可以分配给其它作业
- 恢复：节点再次分配给作业；向任务发送 SIGCONT 信号

作业控制

挂起与恢复

示例

```
$ yhqueue
JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  123      work   job.sh     root   R       0:02      2 cn[0-1]
$ yhcontrol suspend 123
$ yhqueue
JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  123      work   job.sh     root   S       0:02      2 cn[0-1]
$ yhcontrol resume 123
$ yhqueue
JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  123      work   job.sh     root   R       0:05      2 cn[0-1]
```

作业控制

重新排队

- 可将运行或挂起的批处理作业重新排队
  - 释放资源，置为 PENDING 状态
  - 不会重新分配作业 ID
  - 提交时间设为重排队时间

示例

```
$ yhqueue
JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  123      work   job.sh     root   R       0:02      2 cn[0-1]
$ yhcontrol requeue 123
$ yhqueue
JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  123      work   job.sh     root  PD       0:00      2 (Priority)
```

作业控制

修改作业

- 作业提交后可对其参数等进行修改
- 不同状态下可修改的参数不同
- 修改参数时进行类似提交时的权限与资源限制检查
- 通过 `yhcontrol` 命令进行

作业控制

修改作业

改变作业排队的分区

```
$ yhqueue
JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  123      work   job.sh     root  PD       0:00      2 (Resource)
$ yhcontrol update jobid=123 partition=debug
$ yhqueue
JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  123     debug   job.sh     root  PD       0:00      2 (Resource)
```

改变作业的名字

```
$ yhqueue
JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  123      work   job.sh     root   R       0:02      2 cn[0-1]
$ yhcontrol update jobid=123 name=myjob
$ yhqueue
JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  123      work    myjob     root   R       0:05      2 cn[0-1]
```

改变已运行作业的运行时间限制

```
$ yhqueue --format '%i %P %l %t %.5M %N'
JOBID PARTITION TIMELIMIT ST  TIME NODELIST
  123      work     60:00  R  52:12 cn[0-3]
$ su
...
# yhcontrol update jobid=123 timelimit=100
$ yhqueue --format '%i %P %l %t %.5M %N'
JOBID PARTITION TIMELIMIT ST  TIME NODELIST
  123      work  01:40:00  R  52:25 cn[0-3]
```

III. 上机流程

1. 申请帐号
2. 登录到系统的登录节点
3. 编辑 / 编译程序
4. 提交运行作业
5. 查看系统与作业状态
6. 分析作业结果

上机流程

用户登录

- 申请用户帐号
  - HOME 目录一般位于全局共享文件系统
- 登录到登录节点
  - Linux：`ssh` *`username`*`@171.131.1.67`
  - Windows：SecurityCRT/Putty 等
- 图形登录
  - Linux：X，gdm
  - Windows：Xmanager/Xwin 等
- 退出登录
  - exit
  - Ctrl-D

上机流程

编辑 / 编译程序

- 编辑器：vim/emacs/gedit...
- 编译器
  - 串行：gcc/g77/gfortran/icc/ifort
  - MPI：mpicc/mpif77/mpif90...
  - OpenMP：ompc/ompf90/icc

内容回顾

- 系统概述
- 系统使用
  - 资源分配
  - 任务加载
  - 状态查看
  - 作业控制
- 系统上机流程简介

问题？