天河集群Tianhe2-JK系统

管理员手册

CSRC

2015 年10 月

目 录

I 系统概述 1

1 系统概述 3

1.1 高性能计算机系统结构 3
1.2 资源管理系统的组成 4
1.3 资源管理系统的特性 6
1.4 资源管理系统中的实体 7
1.4.1 节点 7
1.4.2 分区 8
1.4.3 作业 8
1.4.4 作业步 8
1.4.5 预约 9
1.4.6 触发器 9
1.5 常用概念 9
1.5.1 关联 9
1.5.2 节点列表表达式 10
1.5.3 处理器术语 11
1.5.4 时间格式 11

II 系统使用 13

2 资源状态查看 15

2.1 分区状态 15
2.1.1 状态查看 15
2.1.2 分区状态 16
2.1.3 分区属性 17
2.2 节点状态 18
2.2.1 状态查看 18
2.2.2 节点状态 19
2.2.3 节点标志 20
2.2.4 节点状态原因 21
2.2.5 节点状态监控机制 22

3 作业状态查看 **25**
3.1 作业状态 25
3.1.1 状态查看 25
3.1.2 历史作业信息 26
3.1.3 作业状态 27
3.1.4 作业标志 28
3.1.5 作业状态原因 29
3.2 作业步信息 30
3.3 预约信息 30

4 作业与资源分配 **31**
4.1 作业运行模式 31
4.1.1 交互作业 31
4.1.2 批处理作业 32
4.1.3 分配模式作业 33
4.2 作业的资源需求约束 35
4.2.1 资源数量 35
4.2.2 节点属性 38
4.3 作业运行参数 39
4.4 作业环境变量 41
4.5 作业的 Epilog 和 Prolog 42
4.5.1 PrologSlurmctld 42
4.5.2 EpilogSlurmctld 43
4.5.3 Prolog 43
4.5.4 Epilog 43

5 作业步与任务加载 **45**
5.1 作业步任务加载 45
5.1.1 作业步创建 45
5.1.2 任务状态 47
5.1.3 任务 I/O 48
5.1.4 信号传递 52
5.1.5 任务执行环境 53
5.1.6 任务布局 57
5.1.7 多程序作业步 63
5.1.8 作业步终止 64
5.1.9 任务的 Prolog 与 Epilog 64
5.1.10 批处理作业步 65
5.2 登录计算节点 65
5.3 MPI 程序加载 65
5.3.1 MPICH2 及其派生版本 65
5.3.2 Open MPI 66
5.3.3 其他 MPI 66

6 作业控制 **67**
6.1 取消作业 67
6.2 信号发送 68
6.3 重新排队 69
6.4 修改作业 70

7 作业检查点 **75**
7.1 作业步检查点 75
7.1.1 `yhrun` 加载任务 75
7.1.2 `yhrun_cr` 加载任务 77
7.1.3 自动检查点 78
7.2 批处理作业检查点 79
7.2.1 检查点 79
7.2.2 恢复执行 81
7.2.3 自动检查点与恢复 81
7.3 MPI 程序的检查点 83
7.3.1 YH-MPI 83
7.3.2 MVAPICH2 83
7.3.3 Open MPI 83
7.3.4 MPICH2 84

8 触发器 **85**
8.1 触发器事件 85
8.2 触发时机 86
8.3 触发器管理 86
8.3.1 设置触发器 86
8.3.2 查看触发器 88
8.3.3 取消触发器 88

III 系统管理 **89**

9 系统设置 **91**
9.1 资源管理系统配置 91
9.1.1 系统配置 91
9.1.2 记账存储配置 92
9.1.3 拓扑配置 92
9.1.4 修改配置文件 92
9.2 高可用支持 92
9.2.1 服务部署 92
9.2.2 故障与恢复流程 93
9.3 支撑环境 93

9.3.1 用户管理 93
9.3.2 时间同步 93
9.3.3 节点命名与地址解析 93
9.3.4 节点主机名 93
9.3.5 节点地址 94
9.4 数据备份 94

10 系统控制 **97**
10.1 分区控制 97
10.1.1 创建分区 97
10.1.2 修改分区 98
10.1.3 删除分区 98
10.2 节点控制 99
10.2.1 修改节点状态 99
10.3 作业控制 101
10.3.1 挂起与恢复 101
10.3.2 消息发送 103
10.4 配置控制 103
10.5 系统控制 104
10.5.1 系统启动 104
10.5.2 关闭系统 105
10.5.3 接管系统 105
10.5.4 Ping 控制进程 106
10.5.5 调整调试日志详细级别 106

11 许可证管理 **109**

12 作业调度 **111**
12.1 概述 111
12.1.1 调度时机 111
12.1.2 调度流程 112
12.2 优先级排队 112
12.3 回填 113
12.4 公平份额 114
12.5 作业抢占 116
12.5.1 抢占方式 116
12.5.2 抢占模式 117
12.6 节点选择 117
12.6.1 节点权重 117
12.6.2 消耗性资源分配 117
12.6.3 拓扑感知 117

13 资源预约 **119**
13.1 概述 119
13.2 预约控制 119
13.2.1 创建预约 119
13.2.2 查看预约 122
13.2.3 修改预约 123
13.2.4 删除预约 123
13.3 使用预约 123
13.4 预约的记账 124

14 关联 **125**
14.1 概述 125
14.1.1 关联的概念 125
14.1.2 关联的属性 125
14.1.3 关联的控制 127
14.2 集群操作 128
14.3 帐号操作 131
14.4 用户操作 135

15 作业 QOS **141**
15.1 QOS 概念 141
15.1.1 资源限制 141
15.1.2 抢占与优先级 143
15.1.3 QOS 使用因子 143
15.2 QOS 操作 144
15.3 QOS 的使用 145

16 作业记账 **147**
16.1 历史作业查询 147

17 全局并行文件系统 **149**
17.1 全局文件系统服务常用管理操作 149
17.2 配置限额 150
17.3 查看文件系统状态 150
17.4 存储系统升降级 151
17.5 文件条带的设置和查询 152

图目录

1.1 高性能计算机系统逻辑结构 3
1.2 资源管理系统组成 5
1.3 作业与作业步 9

3.1 作业状态转换 27

5.1 作业步任务加载流程 45

14.1 份额 127
14.2 帐号的层次结构 131

表目录

第一部分

系统概述

第一章
系统概述

1.1 高性能计算机系统结构

高性能计算机一般由计算处理、互联通信、I/O 存储、监控诊断、基础架构、操作系统、编译器、运行环境、开发工具等多个软硬件子系统组成。从资源管理系统的角度，高性能计算机中的节点按逻辑功能可分为管理节点、登录节点、计算节点和 I/O 节点；节点之间通过网络联接。图 1.1所示为高性能计算机的逻辑结构。

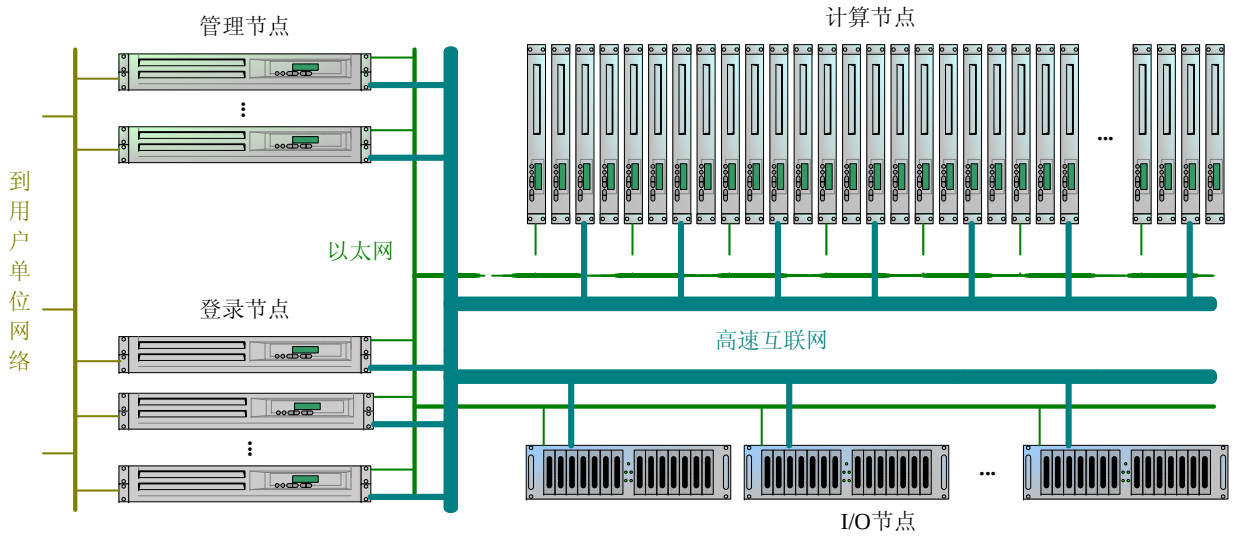


图 1.1: 高性能计算机系统逻辑结构

管理节点运行系统管理进程与相关的支撑服务。管理员在需要时登录管理节点，进行系统的管理与配置工作。普通用户不能登录管理节点。管理节点主机名字一般为 `mn0,mn1,...` 等。

登录节点是用户使用高性能计算机的访问点，供用户登录进行程序编辑、编译，作业提交、运行，结果查看、分析等。登录节点的主机名字一般为 `ln0,ln1,...` 等。

计算节点提供高性能计算能力，运行用户作业的计算任务。资源管理系统所管理的计算资源就是指系统中的计算节点。计算节点的主机名字一般为 cn0,cn1,... 等。

I/O 节点提供存储服务，在使用上表现为一个或多个全局共享的文件系统。登录节点和计算节点在相同的目录挂载这些文件系统。I/O 节点的主机名字一般为 mds0,mds1,...（元数据服务器）以及 ost0,ost1,...（对象存储服务器）等。

系统中的节点之间通过高速互联网联接，提供高带宽、低延迟的计算和 I/O 通信链路。一般地，管理节点、登录节点和 I/O 节点之间还通过以太网进行联接，供系统的管理数据通信、调测试、监控诊断等使用。在某些系统中，计算节点也通过以太网与其它节点联接。管理节点和登录节点通过以太网联接到用户单位的内部网络，供管理员和用户远程访问。

以上描述的是高性能计算机的逻辑结构。对于具体的系统，可以存在物理上的一个节点对应于多个逻辑节点的情况。例如，对于规模较小的系统，可能管理节点和登录节点是同一个物理节点；登录节点也可以同时作为计算节点等。

1.2 资源管理系统的组成

资源管理系统是用户使用高性能计算机中计算资源的接口。用户通过资源管理系统提供的工具查看系统状态，提交作业，加载执行计算任务，查看历史作业信息等。只有通过资源管理系统分配资源，用户才能够访问计算节点。

资源管理系统提供给管理员对系统进行配置管理的机制和工具。管理员可根据需要对系统的分区、调度策略、用户与帐号的资源限制等进行配置修改，为用户预约资源，从记账数据中分析系统使用情况等。

资源管理系统主要由控制进程、节点监控进程、作业管理进程、记账存储进程、命令工具等组成。图 1.2所示为资源管理系统的组成结构。

控制进程 slurmctld 是资源管理系统的中枢服务，负责资源状态维护、资源分配、作业调度、作业管理控制等。控制进程一般运行在管理节点 mn0 上。在某些条件下可以在 mn1 上运行备份控制进程；当 mn0 上的主控控制进程失效时，mn1 上的备份控制进程将接管资源管理服务。

节点监控进程 slurmd 运行在每个计算节点上，负责收集节点上的资源状态并向控制进程报告。slurmd 接收来自控制进程与用户命令的请求，进行作业步任务加载、作业取消等操作。

作业管理进程 slurmstepd 由 slurmd 在加载作业步任务或批处理作业时派生。slurmstepd 进程管理本节点上一个作业步的所有任务，负责计算任务启动、标准 I/O 转

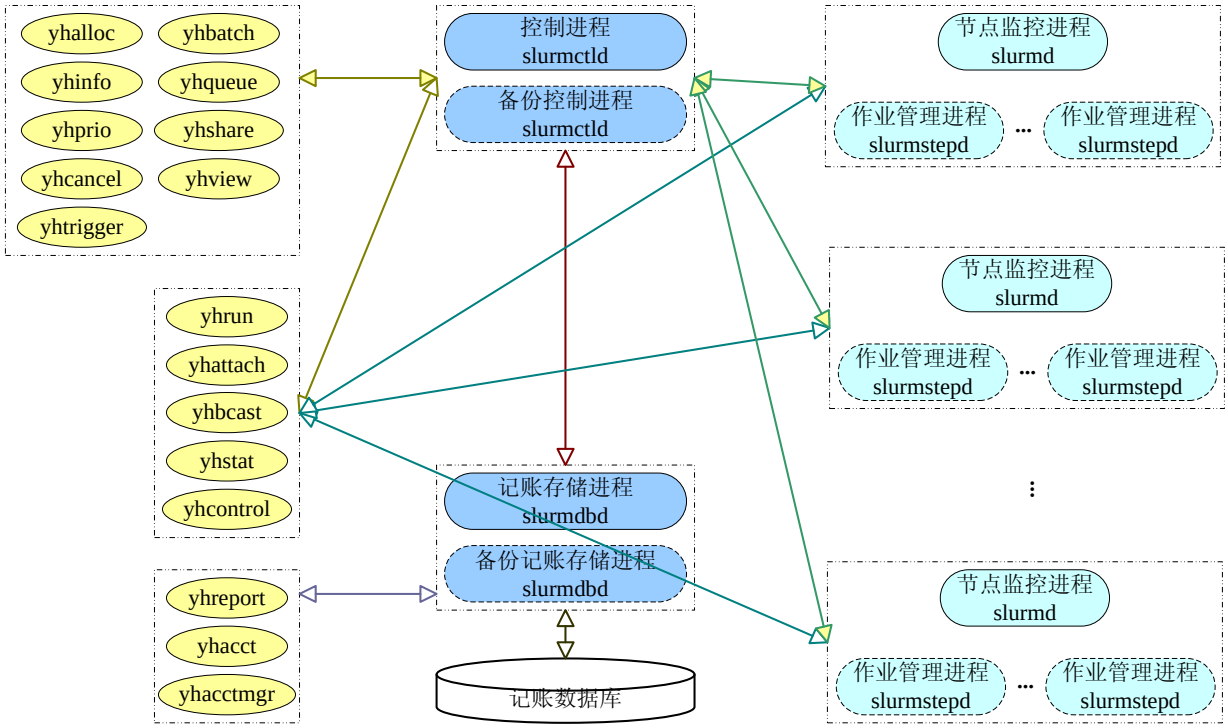


图 1.2: 资源管理系统组成

发、信号传递、记账信息收集以及并行环境初始化等功能。

记账存储进程 `slurmdbd` 一般运行在管理节点 mn0 上，是命令工具与控制进程访问记账数据库的中间层。它提供访问记账数据与关联信息的统一接口，并起到用户认证与安全隔离的作用。

命令工具包括一组供用户和管理员使用的命令，主要有：

- `yhacct`：查看记账数据，查询历史作业信息；
- `yhacctmgr`：记账数据管理，包括用户与计费帐号、资源限制、QOS、负载特性等数据的管理；
- `yhalloc`：分配资源，用于运行分配模式作业；
- `yhattach`：附接 I/O，查看已运行作业步的标准输出/标准错误；
- `yhbatch`：提交批处理作业；
- `yhbcast`：广播文件，利用高效的并发通信机制将文件广播到计算节点；
- `yhcancel`：作业取消与信号发送，取消作业，或给作业的任务发送信号；

- yhcontrol：系统控制，包括重读配置文件，关闭资源管理系统，资源预约，修改实体信息，显示实体的详细信息等；
- yhdiag：系统诊断，显示资源管理系统内部的一些统计数据。
- yhinfo：节点与分区状态查看；
- yhprio：作业优先级查看；
- yhqueue：作业队列状态与作业步信息查看；
- yhreport：系统利用情况报告；
- yhrun：交互作业运行与任务加载；
- yhrun_cr：带检查点支持的任务加载；
- yhshare：查看用户/帐号的份额；
- yhstat：作业步的资源使用情况查看；
- yhtrigger：触发器管理；

管理员可以利用这些命令工具进行 Shell 脚本开发，定制所需功能。资源管理系统也提供 Perl API 进行功能扩展。

1.3 资源管理系统的特性

资源管理系统的主要功能包括：

- 资源分配：支持细粒度的资源选择，可将处理器和内存作为消耗性资源进行分配；网络拓扑感知的优化资源分配；支持资源的预约使用。
- 作业调度：基于综合优先级的作业排队，可定制优先级权重实现不同调度策略；支持 Backfill 调度；支持份额；支持基于分区优先级或 QOS 的作业抢占。
- 任务加载：提供快速的大规模并行任务加载命令，支持并行运行环境的其它任务加载方式。
- 作业容错：可支持基于检查点的自动作业容错，周期性对批处理作业进行检查点，并在节点故障时自动恢复。
- 触发器功能：在节点或作业状态变化时触发事件，可使用定制脚本对事件进行处理。

- 分区使用：可将系统中的计算节点划分为多个分区，实施不同的资源限制和访问控制；分区可重叠。
- 资源限制：基于分区设置作业的资源使用限制；基于关联对用户与帐号等的资源使用进行限制。
- 能耗管理：节点空闲时进入低功耗状态甚至关闭，降低系统能耗。

资源管理系统的主要特点有：

- 高效性：资源管理系统提供了高效的资源管理与作业任务加载。系统实现采用轻量级通信协议，节点间通信采用可扩展性好的层次拓扑逻辑结构。实现上使用位串、独立读写锁等高效的数据结构，支持高并发访问。
- 灵活性：系统提供诸多配置参数，可由管理员根据需要进行调整。可配置作业调度的优先级权重，定制不同的调度策略。支持利用插件进行扩展和功能增强。
- 可靠性：控制进程支持双节点备份。支持基于检查点的自动作业容错。
- 易用性：丰富的命令工具，对系统进行状态查看和作业控制等操作。支持多种作业运行模式。
- 易管理性：基于分区的访问控制和资源限制，基于关联的资源限制，提供作业 QoS 功能。基于数据库的系统记帐与使用分析统计。
- 安全性：采用可靠的用户认证机制，防止身份冒充。基于加密证书的资源分配认证，防止未授权的计算资源访问。

1.4 资源管理系统中的实体

为便于对本手册的理解，本章对资源管理系统中的实体与常用概念进行概要介绍。

1.4.1 节点

如无特殊说明，本手册中节点（node）即指计算节点，是资源分配的基本单位。节点上的资源包括处理器、内存、磁盘空间等等。用户的计算任务在节点上运行。

在资源管理系统中，节点通过节点名标识。节点名是一个字符串，如“cn123”。节点名一般与节点的主机名一致，但也允许两者不相同。节点可能处于空闲、已分配、宕机、无响应、低能耗等状态。

1.4.2 分区

分区（partition）是对系统中的节点进行的逻辑分组。分区提供了一种系统使用和管理机制，可由管理员设置可访问分区的用户、分区中作业的资源限制、分区的优先级等等。分区通过分区名进行标识，分区名是一个字符串，如“work”。

不同的分区可以重叠，即一个节点可以位于多个分区中。从提交作业的角度看，分区类似于其它系统中的“队列”的概念。系统可以有一个默认分区，未指定分区的作业将被提交到默认分区中。

1.4.3 作业

作业（job）是指一次资源分配。用户在使用计算节点之前，必须首先向资源管理系统提出资源分配申请，由系统进行排队调度。用户所提交的资源分配请求在资源管理系统中就对应于一个作业。在作业优先级足够高，且用户请求的资源可用时，资源管理系统将合适的节点分配给作业，用户才能访问所分配的资源。

一个作业分配的资源必须完全位于一个分区内，资源管理系统不会为作业跨分区分配资源。作业通过作业 ID 进行标识。作业 ID 是一个整数，如“123”.

在提交到系统中后，作业可能经历一系列的状态，包括排队、运行、挂起、完成、失败、取消、超时等等。

1.4.4 作业步

作业步（job step）是指通过资源管理系统的 `yhrun` 命令进行的一次任务加载。作业步的任务受资源管理系统管理。在分配资源后，如果用户不使用资源管理系统提供的机制加载计算任务，而是使用其他方式加载，如通过 SSH 运行程序或通过 mpiexec 加载并行任务，则不会产生作业步*，所加载的任务也不受资源管理系统管理控制。

一个作业中可以有多个作业步。即，用户分配一次资源，可以在所分配的计算节点上多次加载执行任务。作业步通过作业步 ID 标识。作业步 ID 由作业 ID 和作业内的作业步序号组成，如“123.1”。作业的第一个作业步的序号为 0，第二个作业步的序号为 1，依此类推。

作业步可以只使用作业所分配的部分节点。一个作业中的多个作业步可以并发执行。图 1.3所示为节点、分区、作业与作业步等实体的示意。

*Open MPI 编译的并行程序在使用 `mpiexec` 加载时，支持使用 `yhrun` 作为进程启动机制。这依赖于 Open MPI 编译时所使用的配置以及 `mpiexec` 的参数。如果使用 `yhrun` 作为进程启动机制加载 Open MPI 程序，则加载程序时将产生一个作业步。但这种情况下 `yhrun` 加载的是 Open MPI 的 orted 程序，而不是直接加载计算程序。可参见第 5.3.2节。

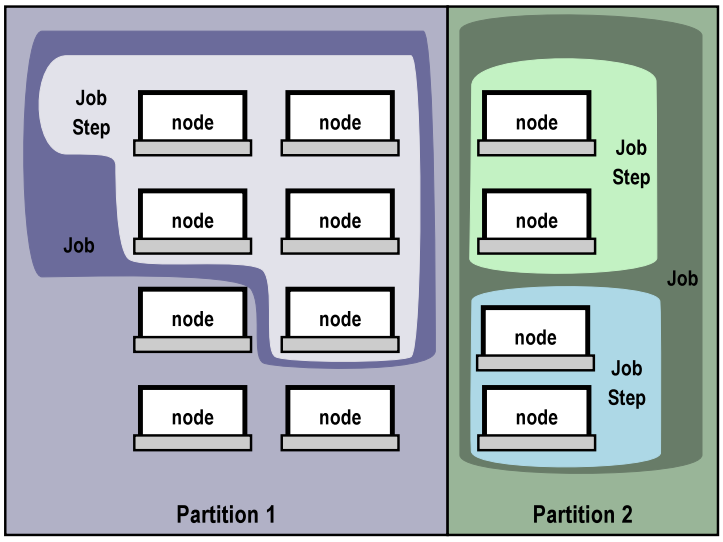


图 1.3: 作业与作业步

1.4.5 预约

预约类似于对资源的一种提前分配。预约有三要素：

- 资源需求：如所需节点数目，节点特性等；
- 访问限制：哪些用户或帐号可以使用预约的节点；
- 起止时间：即预约的访问限制生效的时间；

资源预约成功后，用户提交作业时可指定使用预约的节点。预约可以是周期性的，以一天或一周为周期。

管理员可以使用系统维护预约，避免进行维护中的节点被分配给作业。

1.4.6 触发器

触发器是指在系统发生特定事件时，系统执行用户或管理员设置的程序进行记录、处理的机制。所支持的事件包括节点状态变化、作业状态变化、重读配置文件等。用户指定的程序，在事件发生后由控制进程在管理节点上执行。

1.5 常用概念

1.5.1 关联

关联是系统实施资源限制的一个基础概念。所谓关联，是由高性能计算机系统名、帐号名、用户名和分区名构成的四元组。每个作业都有对应的关联，因为作业都是由用户使

用某个帐号提交到系统的一个分区中。管理员对帐号、用户等设置的资源限制，在实现上最终以关联的属性进行记录。关于关联的详细内容可参见第 14章。

1.5.2 节点列表表达式

在进行作业提交、创建预约、改变节点状态、系统配置等动作时，可能需要指定一组节点。当需要指定的节点数目很多时，逐个列出节点的名字将变得繁琐而易错。为此，系统提供了节点列表表达式，用于配置文件、命令行参数、系统状态输出等。

非正式地，节点列表表达式的语法定义为：

nodelist_exp := *node_exp*[,*node_exp*...]

node_exp := *nodename* | [*prefix1*][*rangelist_exp1*][[*prefix2*][*rangelist_exp2*]]

rangelist_exp := *range_exp*[,*range_exp*...]

range_exp := *number* | *number1*-*number2*

即，节点列表表达式为逗号分隔的节点表达式列表。每个节点表达式可以是一个节点名字，或者是节点范围列表；一个节点表达式中最多可包含两个范围列表表达式。范围表达式可以是一个数值，或为连字符分隔的两个数值，表示该两个数值之间的全部值。合法的节点列表表达式如“cn0,cn[2-5,13,16-21],cn[28-33]”。

使用 `yhcontrol` 命令可以展开或生成节点列表表达式。

节点列表表达式的生成与展开

```
$ yhcontrol show hostlist cn0,cn1,cn2,cn3,cn6,cn7
cn[0-3,6-7]

$ yhcontrol show hostnames cn[0-3,6-7]
cn0
cn1
cn2
cn3
cn6
cn7
```

1.5.3 处理器术语

资源管理系统支持多核处理器的管理与分配。在本手册中，“CPU”或“处理器”泛指逻辑处理器，即在节点操作系统中所看到的处理器。根据具体系统的配置不同，这可能是一颗物理处理器，一个处理器核，或者一个处理器线程。如果需要具体指称，则手册中将使用“Socket”，“Core”，“Thread”，表示物理处理器，处理器核，以及处理器线程。

1.5.4 时间格式

资源管理系统在指定与输出时间时，主要有以下几种格式：

- 秒数：主要用于配置超时时间，如消息传递超时等。
- 分钟数：主要用于作业运行时间限制等。
- *days*-*hours*:*minutes*:*seconds*：用于显示作业的运行时间等。
- YYYY-MM-DDTHH:MM:SS：ISO8601 格式，主要用于显示特定时刻。
- 特定时刻的名字：
  - today：当日
  - tomorrow：次日
  - midnight：0 时 0 分 0 秒
  - noon：12 时 0 分 0 秒
  - teatime：16 时 0 分 0 秒
  - now：当前时刻

第二部分

系统使用

第二章
资源状态查看

资源管理系统提供了一系列的命令，用于对系统中的分区、节点、作业等实体的状态进行查看。

2.1 分区状态

2.1.1 状态查看

查看分区状态的命令是 yhinfo。

```
$ yhinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
work*        up   infinite    128  down* cn[1024-1151]
work*        up   infinite   1023  alloc cn[0-453,455-1023]
work*        up   infinite      1   idle cn454
```

可以通过参数定制 yhinfo 的输出；详情请参见第??章：

```
$ yhinfo --format "%10P%20N"
PARTITION NODELIST
work*     cn[0-1151]
```

yhcontrol 命令可显示分区的详细信息：

```
$ yhcontrol show partitions
PartitionName=work
   AllocNodes=ALL AllowGroups=ALL Default=YES
```

```
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
```

2.1.2 分区状态

分区可以有四种状态：UP，DOWN，INACTIVE，DRAIN。这四种状态对应于两个标志的组合：SUBMIT（可向分区中提交作业）和 SCHED（可调度分区中的作业运行）。

- UP：正常状态，分区具有 SUBMIT 和 SCHED 标志。
- DOWN：分区仅具有 SUBMIT 标志。用户可向分区中提交作业，但分区中的作业不会被调度运行。
- INACTIVE：分区无 SUBMIT 和 SCHED 标志。
- DRAIN：分区仅具有 SCHED 标志。用户不能向分区中提交作业，但分区中已有的作业可以被调度运行。

此外，分区还具有如下标志：

- Default：分区是系统的默认分区。系统中只能有一个默认分区。
- Hidden：分区是隐藏分区。默认地，查看系统状态的命令不显示隐藏分区及隐藏分区中的节点和作业。
- DisableRootJobs：分区不允许 root 用户提交作业。
- RootOnly：分区仅允许 root 用户提交作业。

管理员可以把分区状态临时设置为 DOWN 状态，以暂时停止分区中作业的继续调度；可以把分区临时设置为 DRAIN 状态，以暂时停止向分区中提交作业。管理员也可以通过设置分区为 RootOnly 以暂时停止普通用户向分区中提交作业。设置为 DOWN 状态不影响分区中已经运行的作业。

2.1.3 分区属性

除状态外，分区还具有如下属性：

- AllowGroups

  可以访问分区即向分区中提交作业的用户组。

- AllocNodes

  可以从哪些节点向分区中提交作业。例如，管理员可以设置只能从 ln0 和 ln1 向分区 work 中提交作业。

- Alternate

  备选分区。当用户作业提交到的分区不具有 SUBMIT 标志时，系统将尝试把作业提交到其备选分区中；若备选分区也不具有 SUBMIT 标志，则继续尝试备选分区的备选分区，依此类推。

- MaxTime

  提交到分区中作业的最长运行时间。

- DefaultTime

  提交到分区中作业的默认运行时间。

- MaxNodes

  提交到分区中作业最多分配的节点数。

- MinNodes

  提交到分区中作业最少分配的节点数。

- Nodes

  分区中包含的节点。

- Priority

  分区的优先级。分区的优先级影响其中作业的调度。具体请参见第 12章。

- Shared

  提交到分区中的作业是否共享节点。此属性可取值为：

  – EXCLUSIVE：提交到分区的作业不和其它作业共享节点，总是将整个节点的资源全部分配给作业。

- FORCE[:i]：提交到分区的作业总是可以和其它作业共享节点，即使用户提交作业时指定了 `--exclusive` 选项；节点的资源可以被过度使用。i 为大于 1 的整数，指定最多可以几个作业共享一个节点。i 的缺省值为 4。
- YES[:i]：提交到分区的作业可以和其它作业共享节点，节点的资源可以被过度使用。i 为大于 1 的整数，指定最多可以几个作业共享一个节点。i 的缺省值为 4。
- NO：提交到分区的作业可以和其它作业共享节点，但是节点的资源不能被过度使用。

当分区设置 Shared 为 YES 或 NO 时，如果用户提交作业时指定了 `--exclusive` 选项，则作业仍不共享节点。

2.2 节点状态

2.2.1 状态查看

与查看分区状态相同，查看节点状态的命令是 `yhinfo`。

```
$ yhinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
work*        up   infinite    128  down* cn[1024-1151]
work*        up   infinite   1023  alloc cn[0-453,455-1023]
work*        up   infinite      1   idle cn454
```

可以以面向节点的角度显示：

```
$ yhinfo --Node
NODELIST            NODES PARTITION STATE
cn[0-453,455-1023]   1023     work* alloc
cn454                   1     work* idle
cn[1024-1151]         128     work* down*
```

`yhcontrol` 命令可显示节点的详细信息：

```
$ yhcontrol show nodes cn[0-1]
NodeName=cn0 Arch=x86_64 CoresPerSocket=4
   CPUAlloc=8 CPUErr=0 CPUTot=8 Features=(null)
```

```
   OS=Linux RealMemory=30000 Sockets=2
   State=ALLOCATED ThreadsPerCore=1 TmpDisk=0 Weight=1
   BootTime=2010-05-04T08:09:07 SlurmdStartTime=2010-05-04T08:15:26
   Reason=(null)

NodeName=cn1 Arch=x86_64 CoresPerSocket=4
   CPUAlloc=8 CPUErr=0 CPUTot=8 Features=(null)
   OS=Linux RealMemory=30000 Sockets=2
   State=ALLOCATED ThreadsPerCore=1 TmpDisk=0 Weight=1
   BootTime=2010-05-04T08:09:08 SlurmdStartTime=2010-05-04T08:15:29
   Reason=(null)
```

2.2.2 节点状态

节点可能处于的状态包括：

- UNKNOWN：未知状态

  仅见于系统初启阶段，在控制进程还未能与节点监控进程通信，获得节点的状态之前存在。此状态在 `yhinfo` 的输出中为“unk”。

- DOWN：宕机状态

  处于此状态的节点不能分配给作业使用。节点进入 DOWN 状态的可能原因包括：

  – 节点上资源数量少于系统配置的数量
  – 通信故障导致的节点持续不响应
  – 节点运行作业的 Prolog/Epilog 程序出错
  – 管理员直接设置

  节点进入 DOWN 状态时，在节点上运行的作业将被终止，作业状态标记为 NODE_FAIL（除非用户提交作业时设置了节点容错参数，参见第 4.3节）。

  此状态在 `yhinfo` 的输出中为“down”。

- IDLE：空闲状态

  节点状态正常，且没有分配给任何作业。此状态在 `yhinfo` 的输出中为“idle”。

- ALLOCATED：分配状态

  节点状态正常，且已经分配到一个或多个作业。此状态在 `yhinfo` 的输出中为“alloc”。

2.2.3 节点标志

除状态之外，与节点相关的还有一些标志，这些标志与节点的状态一起，完整地表示节点的信息。标志与状态相对独立，即节点可以在多种状态下都具有某个标志。可能的标志有：

- DRAIN：排空标志

  具有此标志的节点不再被分配到作业，但已经在节点上运行的作业不受影响。在 `yhinfo` 的输出中，具有此标志的节点上如果有作业运行，则显示为“drng”，否则显示为“drain”。

- COMPLETING：正在退出标志

  表示节点上有作业处于退出过程中，正在释放资源。在作业运行结束时（包括作业运行成功、失败、节点故障、超时、被取消等各种情形），其分配的节点被释放，同时这些节点被设置 COMPLETING 标志。具有 COMPLETING 标志的节点不能被分配到需要独占节点的作业。在控制进程确认作业的所有进程都已经在节点上退出，即节点上的资源已被完全释放后，节点的 COMPLETING 标志被清除。

  在 `yhinfo` 的输出中，如果一个节点上有多个作业，且有的处于运行状态，有的处于退出过程中，则节点状态显示为“alloc+”；如果节点上所有的作业都处于退出过程中，则节点状态显示为“comp”。如果 `yhinfo` 显示有节点具有 COMPLETING 标志，则 `yhqueue` 命令可以看到对应的具有 COMPLETING 标志的作业。

  COMPLETING 标志的存在是为了确保作业退出时完全释放资源，是系统的正常处理流程。在正常情况下，即通信正常，作业未残留不可杀死的进程等，COMPLETING 标志是暂时性的。如果一个节点长期具有此标志，则表明此节点有故障，需要管理员干预处理：要么节点上残留了作业进程，要么节点与管理节点的通信有问题，使得控制进程不能及时获得作业进程退出的通知。

- NO_RESPOND：无响应标志

  表示节点监控进程与控制进程的通信有故障。为了维护节点的状态，控制进程周期性地 ping 所有的非 DOWN 状态的计算节点，以测试节点是否响应。系统的一些其他动作也涉及到控制进程与节点上监控进程之间的通信。若这些通信失败，节点将被设置 NO_RESPOND 标志。如果在系统配置的周期中，与节点的通信连续失败，则节点将被置为 DOWN 状态。

  在 `yhinfo` 的输出中，具有此标志的节点的状态后带有“*”符号。

- POWER_SAVE：节能标志

  表示此节点目前被控制进程设置为节能状态。在 yhinfo 的输出中，具有此标志的节点的状态后带有“~”符号。

- FAIL：失效标志

  仅由管理员设置。具有此标志的节点不会被分配给作业。在 yhinfo 的输出中，具有此标志的节点上如果有作业运行，则显示为“failg”，否则显示为“fail”。

- POWER_UP：启动标志

  表示节点正在从节能状态恢复。具有此标志的节点在 yhinfo 的输出中显示为“power_up”。

- MAINT：系统维护标志

  表示节点正在进行系统维护，即节点处于系统维护性预约中，且当前时刻在该预约的起止时间内。具有此标志的节点在 yhinfo 的输出中显示为“maint”。

2.2.4 节点状态原因

节点处于某些状态或被设置某些标志时，可以有一个字符串描述其处于当前状态或具有当前标志的原因。除 yhcontrol 命令查看节点详细信息时显示节点处于当前状态的原因外，可以通过“yhinfo -R”命令查看节点处于不可用状态的原因：

```
$ yhinfo -R
REASON                USER      TIMESTAMP           NODELIST
Not responding        root      2010-07-09T15:49:46 cn[295,387,600-603,902,942]
Low RealMemory        root      2010-07-12T19:57:01 cn[604,608,622]
disk error            root      2010-07-12T21:11:24 cn882
Node silently failed root      2010-07-13T11:43:22 cn606
```

如果节点的状态或标志是管理员直接设置的，则原因由管理员提供。如果是节点状态或标志是资源管理系统设置的，则系统将提供原因。系统设置的可能原因如下：

- 节点处于 DOWN 状态
  - Not responding [slurm@*timestr*]：节点无响应；*timestr* 为节点被设置为 DOWN 状态的时间
  - Prolog failed：运行作业的 Prolog 程序失败

  - Epilog error：运行作业的 Epilog 程序失败
  - Prolog/epilog failure：运行作业的 Prolog/Epilog 程序失败
  - Low socket*core*thread count：节点监控进程向控制进程注册的 Socket、Core、Thread 数乘积少于系统配置数目
  - Low CPUs：节点监控进程向控制进程注册的 CPU 数少于系统配置数目
  - Low RealMemory：节点监控进程向控制进程注册的物理内存少于系统配置数目
  - Low TmpDisk：节点监控进程向控制进程注册的临时磁盘空间少于系统配置数目
  - Node silently failed and came back [slurm@*timestr*]：节点在两次向控制进程之间曾经重启；*timestr* 为节点被设置为 DOWN 状态的时间

- 节点具有 DRAIN 标志
  - SlurmdSpoolDir is full：节点配置的 SlurmdSpoolDir 目录没有空间，不能写入系统所需要的状态文件或作业脚本文件等。

2.2.5 节点状态监控机制

资源管理系统提供三种节点状态监控机制：ping，register，和 health check。

Ping

控制进程 `slurmctld` 周期性地 ping 所有不处于 DOWN 状态或 POWER_SAVE 状态的节点。ping 的周期为系统配置文件中 SlurmdTimeout 参数值（秒数）的 1/3。

ping 机制简单地向计算节点发送一个消息，并等待其响应。如果节点响应超时，则节点被设置 NO_RESPOND 标志（`yhinfo` 的输出中状态带“*”）。如果节点连续三次响应超时，则节点将被设置为 DOWN 状态，原因为“Not Responding”。

在 `slurmctld` 与节点上的 `slurmd` 其它类型消息的通信过程中，如果节点响应超时，节点也会被设置 NO_RESPOND 标志。

Register

计算节点上的 `slurmd` 启动时，会向控制进程注册节点上的资源与作业状态。控制进程也会周期性地请求节点进行注册。

由于注册机制的开销相对较高，因此在每个 ping 周期内（SlurmdTimeout 参数值的 1/3），系统仅请求不超过节点间树状通信的树宽度（系统配置文件中 TreeWidth 参数确

定）个节点注册。在节点数目众多的情况下，这导致每个节点被请求注册的周期增长。同时，系统保证每个节点在 20 个 ping 周期内最多被请求注册一次，以避免节点数目较少时频繁注册。

Health Check

通过系统配置文件中的 HealthCheckInterval 和 HealthCheckProgram 参数，管理员可以配置周期性地在所有非 DOWN 状态的节点上执行节点健康状态检查程序。HealthCheckProgram 中可根据检查结果设置节点的合适状态。

管理员可以在每个节点上独立设置周期性地进行健康状态检查，但 Health Check 机制可以（接近）同时在节点上启动检查程序。

通过 yhcontrol 命令修改节点状态及原因，管理员还可以定制外部的节点状态监控机制。例如，可以根据节点温度、风扇转速等的监控结果，设置节点的合适状态。

第三章
作业状态查看

3.1 作业状态

3.1.1 状态查看

查看系统队列中作业的状态使用 yhqueue 命令。

```
$ yhqueue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
 295376      work     xhpl     root   R    1:54:04    255 cn[256-453,455-511]
 295375      work     xhpl     root   R    1:57:09    768 cn[0-255,512-1023]
```

默认地，yhqueue 仅显示处于 PD、R、S 状态以及具有 CG 标志的作业。可通过 --format 选项定制 yhqueue 的输出，详情请参见第??节。

可通过 yhcontrol 命令查看系统中具有 COMPLETING 标记的作业，及其相关的节点：

```
$ yhcontrol completing
JobId=1468 Nodes(COMPLETING)=cn[0,43,51,59,68,76,100,116-117,125,133,173-174,
214,230-231,239,247,255,263,279,296,328,344,401-403,459,476,500,508,517,573-574,
582,590,630,655,720,753]
JobId=1469 Nodes(COMPLETING)=cn[0-1,173,182,198,230,255,263,271,279,287,296,312,
328,344,353,401-403,435,443,459,476,500,508]
```

yhcontrol 命令可查看作业的详细信息：

```
$ yhcontrol show jobs 295375
JobId=295375 Name=xhpl
```

```
   UserId=root(0) GroupId=root(0)
   Priority=2 Account=(null) QOS=normal
   JobState=RUNNING Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=0 ExitCode=0:0
   RunTime=01:59:24 TimeLimit=69-10:40:00 TimeMin=N/A
   SubmitTime=2010-05-10T15:26:41 EligibleTime=2010-05-10T15:26:41
   StartTime=2010-05-10T15:28:01 EndTime=2010-07-19T02:08:01
   SuspendTime=None SecsPreSuspend=0
   Partition=work AllocNode:Sid=ln0:24947
   ReqNodeList=cn[0-255,512-1023] ExcNodeList=(null)
   NodeList=cn[0-255,512-1023]
   NumNodes=768 NumCPUs=6144 CPUs/Task=1 ReqS:C:T=1:1:1
   MinCPUsNode=1 MinMemoryNode=0 MinTmpDiskNode=0
   Features=(null) Reservation=(null)
   Shared=0 Contiguous=0 Licenses=(null) Network=(null)
```

3.1.2 历史作业信息

运行结束后，作业信息将在系统中保留一段时间，然后被删除。用户可使用 yhacct 命令，从记账数据库中获取历史作业信息。

```
$ yhacct
     JobID    JobName  Partition    Account  AllocCPUS      State ExitCode
---------- ---------- ---------- ---------- ---------- ---------- --------
        18       xhpl       work       test          0 CANCELLED+      0:0
        30       xhpl       work       test          0 CANCELLED+      0:0
        32       xhpl       work       test          0 CANCELLED+      0:0
        34    test.sh       work       test        256  COMPLETED      0:0
        37    test.sh       work       test        256     FAILED      0:0
        40    test.sh       work       test       2048  COMPLETED      0:0
      40.0       xhpl                  test       2048  COMPLETED      0:0
        41    test.sh       work       test       2048 CANCELLED+      0:0
      41.0       xhpl                  test       2048 CANCELLED+      0:0
        42    test.sh       work       test       2048 CANCELLED+      0:0
        45    test.sh       work       test       2048 CANCELLED+      0:0
        46    test.sh       work       test       2048    RUNNING      0:0
      46.0       xhpl                  test       2048    RUNNING      0:0
        48    test.sh       work       test          0    PENDING      0:0
```

可以通过命令行选项定制 yhacct 命令的输出与作业查询条件，详情请参见第??节。

3.1.3 作业状态

作业在提交到系统中后，将经历一系列的状态，如图 3.1所示。

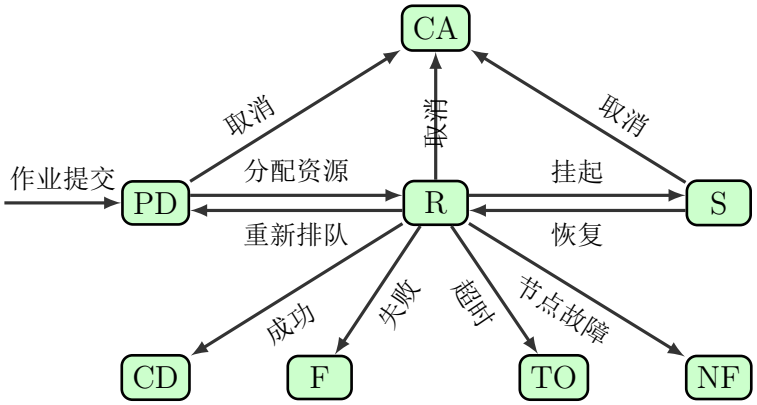


图 3.1: 作业状态转换

其中，CD、F、CA、TO、NF 均为作业运行结束的状态，只是结束原因不同。

- PENDING：排队状态

  作业在队列中排队调度，等待分配资源。在 yhqueue 输出中此状态显示为“PD”。

- RUNNING：运行状态

  作业被分配资源后进入运行状态。运行状态并不意味着作业的计算任务已经在节点上执行，仅表示作业已经获得所请求的资源。在 yhqueue 输出中此状态显示为“R”。

- SUSPENDED：挂起状态

  用户可以将运行状态下的作业挂起。被挂起的作业将释放分配的资源；如果有计算任务在节点上运行，则这些任务将收到挂起信号，从而让出 CPU。然而，计算任务占用的其它资源如内存等不能被释放。在 yhqueue 输出中此状态显示为“S”。

- COMPLETED：成功结束

  作业运行结束，且退出代码为 0。对于批处理作业，指作业脚本的退出代码为 0；对于交互作业，指作业的所有任务的退出代码均为 0；对于分配模式作业，指所执行的命令的退出代码为 0。参见第 4章。在 yhqueue 输出中此状态显示为“CD”。

- FAILED：失败结束

  作业因为参数错误而不能运行，如没有足够的权限访问分区等；或作业运行结束，但退出代码非 0。在 yhqueue 输出中此状态显示为“F”。

- CANCELLED：被取消

  作业在排队、挂起或运行状态，被用户或管理员取消；或因为系统配置变化原因，如作业所在的分区被删除等，被系统取消。取消作业的用户的 uid 将被记录在记账信息中；如果是系统取消的作业，将记录 uid 为 -1。在 yhqueue 输出中此状态显示为“CA”。

- TIMEOUT：超时

  作业的运行超出了其请求的运行时间限制；或者对于交互作业（或分配模式作业），请求分配资源的 yhrun 命令（或 yhalloc 命令）对控制进程 slurmctld 的 ping 响应超时。在 yhqueue 输出中此状态显示为“TO”。

  控制进程定期 ping 交互作业和分配模式作业的 yhrun 和 yhalloc 命令，以确认其进程活跃。如果响应超时，相应的作业将被终止。

- NODE_FAIL：节点故障

  在作业运行过程中，分配给作业的节点变为 DOWN 状态，且作业提交时没有指定节点故障容忍参数。在 yhqueue 输出中此状态显示为“NF”。

3.1.4 作业标志

除了状态之外，作业也有一些标志：

- COMPLETING：退出标志

  COMPLETED、FAILED、CANCELLED、TIMEOUT 和 NODE_FAIL 都是作业已经结束的状态，只是结束的原因不同。如果作业曾经运行即曾经被分配资源，则作业结束后除了进入相应的状态，还将被设置 COMPLETING 标志，表示作业正在退出过程中，等待释放资源。当作业的所有计算任务在所分配的节点上全部结束，资源完全释放后，COMPLETING 标志被清除。具有 COMPLETING 标志的作业在 yhqueue 的输出中状态显示为“CG”。

- CONFIGURING：启动标志

  在为作业分配资源时，如果分配给作业的节点处于节能状态，则作业将被设置 CONFIGURING 标志，且节能状态的节点将会被恢复正常。在所分配的全部节点进入正常状态后，作业的 CONFIGURING 标志被清除。具有 CONFIGURING 标志的作业在 yhqueue 的输出中状态显示为“CF”。

3.1.5 作业状态原因

当作业处于排队和结束状态时，系统将会记录其处于该状态的原因。在 `yhqueue` 的输出可以看到作业处于当前状态的原因。可能的原因有：

- 排队状态
  - Priority：作业优先级不够高
  - 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
  - BeginTime：作业请求的启动时间还未到达
  - Licenses：当前可用的许可证不能满足作业需求
  - 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：作业所在的分区处于 DOWN 状态
  - 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

3.2 作业步信息

使用 yhqueue 命令查看系统中的作业步的信息。

```
$ yhqueue -s
STEPID          NAME PARTITION     USER      TIME NODELIST
45.0            xhpl      work  test605   2:00:54 cn[0-255,512-1023]
46.0            xhpl      work  test605   1:57:49 cn[256-511]
```

yhcontrol 命令可以查看作业步的详细信息。

```
$ yhcontrol show steps 46.0
StepId=46.0 UserId=1000 StartTime=2010-05-20T08:19:27 TimeLimit=6-22:40:00
   Partition=work Nodes=cn[256-511] Tasks=256 Name=xhpl Network=(null)
   ResvPorts=(null) Checkpoint=0 CheckpointDir=/vol6/home/test605/hpl/goto/256-2
```

yhstat 命令可以查看作业步的资源使用信息。

```
$ yhstat -j 46.0
     JobID  MaxVMSize  MaxVMSizeNode  MaxVMSizeTask  AveVMSize     MaxRSS
MaxRSSNode MaxRSSTask     AveRSS MaxPages MaxPagesNode   MaxPagesTask
AvePages     MinCPU MinCPUNode MinCPUTask     AveCPU   NTasks
---------- ---------- -------------- -------------- ---------- ----------
---------- ---------- ---------- -------- ------------ --------------
---------- ---------- ---------- ---------- ---------- --------
      46.0  27354736K          cn256              0  27273306K  27198168K
     cn294         38  27114316K        0         cn256              0
         0 1-00:13:22      cn444        188 1-00:57:08      256
```

yhstat 显示的资源使用信息通过系统记账机制获取。由于记账信息的收集是周期性进行（系统配置文件中的 JobAcctGatherFrequency 参数），因此数据可能存在延时。yhstat 的输出可以通过命令行选项定制，详情及输出字段的秒数，请参见第??节。

作业步仅在运行时存在，运行结束后从系统中删除，并记录到记账数据库中。使用 yhacct 命令可查看历史作业步信息。

3.3 预约信息

使用 yhcontrol 命令查看系统中预约的信息。

详情请参见第 13章与第??节。

第四章
作业与资源分配

为了使用高性能计算机中的计算节点，用户需要向资源管理系统提出资源分配请求。资源分配请求以作业的形式提交，进行排队和调度。在成功分配满足约束的资源后，用户即可在所分配的节点上加载计算任务。

在资源管理系统中，一个作业就是指的一次资源分配请求。作业在系统中可能经历排队、运行、挂起、终止等状态。排队是指作业在等待资源分配。运行是指已经将资源分配给作业，不管有没有加载计算任务。挂起是指暂时将分配的资源释放，如果作业中通过 yhrun 在分配的节点上加载了作业步的计算任务，则这些任务将被挂起，释放处理器。终止是指作业运行结束，资源释放；依照作也终止的原因，又可分为成功、失败、取消、节点故障、超时等几种情况。

4.1 作业运行模式

按照用户的使用模式，作业可以分为三种类型：交互作业、批处理作业和分配模式作业。

4.1.1 交互作业

交互作业是指用户在命令行窗口提交作业资源分配请求，等待资源分配，然后加载要运行的计算任务。计算任务运行过程中，其标准 I/O 被传递到用户的命令行窗口。用户可以在命令行窗口与计算任务进行交互，包括提供输入、发送信号、查看任务运行状态等。在加载的计算任务执行结束后，作业运行结束，用户申请的资源被释放。

交互作业使用 yhrun 命令运行。用户在 yhrun 的参数中指定资源分配的需求约束，以及要加载执行的计算任务及其控制参数。

eg 交互模式作业

```
$ yhrun -n 4 cpi
yhrun: job 52 queued and waiting for resources
yhrun: job 52 has been allocated resources
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
$
```

4.1.2 批处理作业

批处理作业是指用户编写作业脚本，指定资源需求约束，然后作为作业提交。提交批处理作业提交的命令随即执行结束，返回命令行窗口。作业在系统中进行排队调度，在资源需求被满足，分配到计算节点之后，系统将在所分配的第一个节点上加载执行用户的作业脚本，执行时设置合适的环境变量，以反映所分配的资源。用户可以在脚本中加载计算任务，使用为批处理作业所分配的计算节点。批处理脚本执行产生的标准 I/O 被写入文件或从文件读取。

批处理作业使用 yhbatch 命令提交。用户在 yhbatch 的参数中指定资源分配的需求约束。编写的作业脚本中，如果使用 yhrun 命令加载计算任务，则 yhrun 将通过环境变量感知到已经分配资源，从而直接创建作业步而不会再次提交作业。

批处理作业的脚本是一个文本文件。脚本文件的第一行应以“#!”开头，指定脚本文件的解释程序。在脚本中，如果一行以“#SBATCH”开头，则该行中的其余部分被当作命令行选项，被 yhbatch 处理。

eg 批处理作业

脚本文件：

```
$ cat job.sh
#!/bin/sh
#SBATCH -N 16 -t 100 -n 16 -c 4
yhrun -n 16 hostname
```

提交作业：

```
$ yhbatch job.sh
Submitted batch job 53
$ yhqueue
JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
   53      work   job.sh  test605  PD       0:00     16 (Priority)
   47      work  test.sh  test605  PD       0:00    256 (Resources)
   46      work  test.sh  test605   R    7:04:01    256 cn[256-511]
```

运行后，生成输出文件：

```
$ ls
hpl-2.0  iotest  job.sh  NPB3.3-MPI  slurm-53.out  test.sh
```

4.1.3 分配模式作业

分配模式作业类似于交互作业与批处理作业的结合。用户通过指定资源分配的需求约束，向资源管理系统提出作业的资源分配请求。作业在系统中排队调度，而用户等待资源分配请求被满足。然后，*在用户申请资源分配请求*的节点上，执行用户所指定的命令，执行时设置合适的环境变量，以反映所分配的资源。由于在前台运行，用户可在命令执行过程中与其进行交互。指定的命令执行结束后，作业运行结束，用户申请的资源被释放。

如果用户指定的命令为 `yhrun` 等任务加载命令，则可直接在所分配的计算节点上加载计算任务。如果用户指定的命令为 `/bin/sh` 等 shell 解释器，则用户获得一个设置了合适环境变量的 shell 环境。在此 shell 中，用户可以通过 `yhrun` 等命令，多次加载计算任务。

在通过分配模式作业执行 shell 解释器时，应注意在使用结束后以 exit 命令等方式退出，以终止作业并释放资源。

分配模式作业使用 yhalloc 命令提交。用户在 yhalloc 的参数中指定资源分配的需求约束。如用户未指定，则 yhalloc 分配资源后执行系统配置文件中 SallocDefaultCommand 参数指定的程序。

分配模式作业

```
$ yhalloc -N 2 -n 4 -c 2 -t 100 /bin/sh
yhalloc: Granted job allocation 56
sh-3.2$ yhrun -n 4 hostname
cn1
cn1
cn0
cn0
sh-3.2$ ssh cn0 ls
test605@cn0's password:
hpl-2.0
iotest
job.sh
NPB3.3-MPI
slurm-53.out
test.sh
sh-3.2$ yhrun -n 2 date
Thu May 20 15:46:05 CST 2010
Thu May 20 15:46:05 CST 2010
sh-3.2$ exit
exit
yhalloc: Relinquishing job allocation 56
$
```

4.2 作业的资源需求约束

用户在提交作业时（yhrun/yhbatch/yhalloc），可以指定要分配的资源的需求约束。本节介绍主要的需求约束选项；具体的选项请参见第??、??、??节。

4.2.1 资源数量

节点数目

作业所需要的节点数目范围可通过 -N,--nodes 选项指定，格式为“min[-max]”，表示最少分配 min 个节点，最多分配 max 个节点。max 是可省的。若只指定一个数值 min，则严格分配 min 个节点。

只指定 min 而省略 max 的含义与以前的版本不同。以前的含义是至少分配 min 个节点，实际分配的节点数目可能超过 min。

如未指定作业的节点数目，则系统根据作业的其它资源需求约束，分配足够多的节点。

作业所能使用的节点数目范围，受分区配置的作业节点数目范围的限制，以及作业的帐号与 QOS 的资源约束的限制。

处理器数目

作业需要的处理器数由几个参数确定：

- 作业要加载的任务数（-n,--ntasks）

  对于交互作业，资源分配和任务加载通过同一条 yhrun 命令进行，此参数容易理解。对于批处理作业和分配模式作业，yhbatch 和 yhalloc 命令仅用于分配资源，故此参数应设为作业脚本或要执行的命令中，将要加载的最大任务数。对于批处理和分配模式作业，若申请分配的资源少于实际任务加载所需要使用的资源，则有可能因资源不够而不能创建作业步，导致加载任务失败。

  如果没有直接通过 --ntasks 参数指定要加载的任务数，也可以通过指定作业所需的节点数目，以及 --ntasks-per-node 参数指定每个节点上要加载的任务数。默认地，认为用户将在每个所分配的节点上加载 1 个计算任务。

- 每个任务使用的处理器数（-c,--cpus-per-task）

  默认地，每个计算任务使用一个处理器。对于某些多线程的计算任务，例如 OpenMP 程序，使用多个处理器可以得到最佳性能。

- 是否过度使用资源（-O,--overcommit）

  一般地，在一个处理器上仅能加载一个任务；但如果显式请求了过度使用资源，则可以在一个处理器上加载多个任务。默认地，不过度使用资源。

系统将根据用户的指定参数进行计算，（在指定的节点数目范围内）分配足够提供所需处理器数目的节点。

eg 节点与处理器数目需求约束

分配 4 个节点：

```
$ yhbatch -N 4 job.sh
```

分配足够提供 128 个处理器的节点：

```
$ yhbatch -n 128 job.sh
```

分配足够提供 $64 \times 4 = 256$ 个处理器的节点，但至少 32 个节点，最多 48 个节点：

```
$ yhbatch -N 32-48 -n 64 -c 4 job.sh
```

分配 8 个节点：

```
$ yhbatch -N 8 -n 128 -O job.sh
```

分配 $128 \div 2 = 64$ 个节点：

```
yhbatch --ntasks-per-node=2 -n 128 job.sh
```

即使过度使用资源，在一个节点上所能加载的最大任务数也是受限的。目前版本中此限制为 128。

物理内存

--mem 选项指定作业在每个节点上使用的物理内存数量（MB）。--mem-per-cpu 选项指定作业在所分配的每个处理器上使用的内存数量（MB）。

如果没有指定，则作业所需要的物理内存由系统配置文件中 DefMemPerCPU 或 DefMemPerNode 参数定义的默认值确定。

在计算节点上，对用户作业的任务以及批处理脚本实施内存限制。如果用户的计算任务在执行过程中使用的物理内存超过所申请的内存数量，则计算任务将被终止，或者因不能分配到内存而执行失败。

为了使系统在为作业分配资源时考虑内存需求，管理员需要在系统配置文件中设置内存为消耗性资源。否则，仅在计算节点上对作业的任务实施内存限制，而在分配节点时不考虑可用的内存数量。

作业可请求的每节点或每处理器的物理内存数量，不能超过系统配置文件中 MaxMemPerCPU 或 MaxMemPerNode 参数设置的值。

临时磁盘空间

--tmp 选项指定作业在每个节点上需要的临时磁盘空间大小（MB）。

由于资源管理系统不支持将临时磁盘空间作为消耗性资源，因此此选项的含义仅仅是，分配给作业的节点上，临时磁盘空间不能少于此选项值。

运行时间

-t,--time 选项指定用户可以指定作业的运行时间限制（分钟）。如果作业的运行时间超过限制，该作业将被系统终止，作业的状态被标记为超时。

如未指定，作业的运行时间限制将被设置为其所在分区的默认作业运行时间限制。

作业运行时间影响其调度。在 backfill 时，作业的运行时间是决定其能否提前运行的重要因素。因此，建议用户尽可能准确地估计作业运行时间（并稍作保守放大，以免因作业运行时间估计过短而导致作业被终止）。

管理员可在系统配置文件中通过参数 OverTimeLimit 设置作业可以超出所请求运行时间的长度。

许可证

--license 选项指定作业需要的许可证及数量，格式为“*lic1***n1*[,*lic2***n2*...]”。在作业调度时，只有许可证足够的作业才可能被运行。

系统中可用的许可证由管理员在系统配置文件中定义。

4.2.2 节点属性

分区

-p,--partition 选项指定作业要使用的分区。作业将从指定的分区中分配资源，同时使用指定分区的配置进行访问控制检查、资源限制等。如未指定，作业将被提交到系统的默认分区。

一个作业必定位于一个分区中。作业不能跨分区。

预约

--reservation 选项指定作业要使用的预约。作业将仅从预约的节点中分配资源。用户和作业使用的帐号需要有访问预约的权限。

如果作业在预约中运行，则作业必须在预约的时间结束之前终止，否则将被强行终止，其状态被标记为超时。

管理员可在系统配置文件中通过参数 ResvOverRun 设置作业可以超出预约时间的长度。

指定节点

-w,--nodelist 选项指定分配给作业的资源中至少要包含的节点。参数格式为节点列表表达式。

-F,--nodefile 选项指定分配给作业的资源中至少要包含的节点，但节点写在文件中，而不是在命令行直接给出。

-x,--exclude 选项指定分配给作业的资源中不要包含的节点。参数格式为节点列表表达式。

处理器特征

--mincpus 选项指定分配给作业的节点上至少要具有的处理器数目。

--sockets-per-node 选项指定分配给作业的节点上，每个节点至少具有的可用的 socket 数目。即在每个节点上最少分配给作业的 socket 数目。从这个意义上，此参数也可视为资源数量请求。

--cores-per-socket 选项指定分配给作业的节点上，每个 socket 至少具有可用的 core 数目。即分给作业的每个 socket 上至少分配给作业的 core 数目。

`--threads-per-core` 选项指定分配给作业的节点上，每个 core 至少具有的 thread 数目。

`-B,--extra-node-info` 选项相当于同时指定 `--sockets-per-node,--cores-per-socket,--threads-per-core` 三个选项，格式为“$S[:C[:T]]$”，S、C、T 分别表示每节点的 socket 数、每 socket 的 core 数、每 core 的 thread 数。C 和 T 可省，默认为 1。

共享节点

高性能计算机采用消耗性资源分配方法，以提供细粒度的资源分配控制。管理员可将节点上的处理器和内存配置为消耗性资源；资源管理系统为作业选择节点时，将考虑节点上已经被使用的资源。

`--exclusive` 指定作业不能与其它作业共享节点。默认作业可以共享节点。

作业的共享节点情况还受分区配置的影响。详情参见第**??**节。

连续节点

`--contiguous` 选项表示作业需要被分配连续的节点。所谓连续，是指节点的定义在系统配置文件中的出现顺序是连续的。

节点特性约束

`-C,--constraint` 选项表示分配给作业的节点需要具有指定的特性。节点的特性由管理员在系统配置文件中定义。参见第**??**节。

此选项的格式可为如下：

- *f1*[**n1*][&*f2*[**n2*]...]：分配给作业的节点中，有 *n1* 个节点具有 *f1* 特性，有 *n2* 个节点具有 *f2* 特性，等等。即指定的特性之间是“与”关系。
- *f1*[|*f2*...]：分配给作业的每个节点，要么具有 *f1* 特性，要么具有 *f2* 特性，等等。
- [*f1*|*f2*...]：分配给作业的所有节点，要么全部具有 *f1* 特性，要么全部具有 *f2* 特性，等等。

4.3 作业运行参数

除了资源需求约束外，用户还可在提交作业时指定一些参数。这些参数将影响作业的调度、访问控制、资源限制、记账等方面。

- 作业名字（-J,--job-name）

  用户可为作业指定名字。如未指定，则默认地，交互作业的名字为所加载的任务的可执行文件的名字；批处理作业的名字为作业脚本文件的名字；分配模式作业的名字为所执行命令的可执行文件名。

  作业名字在查看作业队列状态时显示，也保存在作业记账数据中。

- 帐号（-A,--account）

  用户可指定作业要使用的帐号。如未指定，则使用用户的默认帐号。

  作为关联的一个组成因子，帐号影响作业的记账、资源限制、QOS 等。详情请参见第 14、15、16章。

- QOS（--qos）

  用户可指定作业要使用的 QOS 级别。如未指定，则使用系统的默认 QOS 级别。

- 负载特性（--wckey）

  用户可指定作业的负载特性。作业的负载特性将保存在记账数据中，主要用于对系统的使用情况进行统计分析。

- 依赖关系（-d,--dependency）

  用户可指定作业将在其它作业启动、结束等之后运行。

- 启动时间（--begin）

  用户可指定作业的最早启动时间不早于指定时间。

- 用户/组（--uid,--gid）

  仅 root 用户可用。以指定用户/组的身份运行作业。这种情况下，作业提交时系统的访问控制检查针对提交作业的用户身份即管理员进行；而实际的作业（计算任务/批处理脚本/指定命令），以指定的用户/组的身份运行。可用于由管理员替用户在无访问权限的分区中提交作业。

- 工作目录（-D,--chdir/--work-dir）

  作业的工作目录。对交互作业，是所加载的计算任务执行时的工作目录；对批处理作业，是作业脚本执行时的工作目录；对分配模式作业，是用户指定命令执行时的工作目录。默认地，作业的工作目录为提交作业的命令（yhalloc/yhbatch/yhrun）的工作目录。

- 节点故障容忍（`-k,--no-kill`）

  默认地，如果分配给作业的节点发生故障（即节点变为 DOWN 状态），则在节点上运行的作业将被终止，并回收所分配的资源。用户在提交作业时可以指定，该作业可以在节点故障的情况下继续运行，从而避免在节点故障时作业被终止。这对于两种情况有用：一是用户的计算任务本身可以容错，在某些任务异常终止的情况下可以继续运行并生成有效结果；二是对于批处理或分配模式作业，用户可以继续在未发生故障的节点上加载任务。

- 邮件通知（`--mail-type,--mail-user`）

  用户可请求在作业开始、结束、或失败时向指定用户发送邮件。`--mail-type` 指定发送邮件的时机，可选值为：

  - BEGIN：在作业开始运行时发送邮件；
  - END：在作业成功结束或被取消时发送邮件；
  - FAIL：在作业失败、超时、因节点失效而失败时发送邮件；
  - ALL：上述三种情况下均发送邮件。

  `--mail-user` 指定接收邮件的用户。默认为提交作业的用户。

  此功能的正常工作需要管理员对在系统配置文件中设置合适的邮件发送程序（系统配置文件中的 MailProg 参数），以使得管理节点上发送的邮件能够被用户接收。

4.4 作业环境变量

在为作业分配资源后加载执行任务/批处理脚本/指定程序时，会设置一些环境变量。用户可通过这些环境变量获取相应的信息。例如，用户可以从环境变量获取所分配的节点，并将其指定为 `mpiexec` 的参数，以在分配的节点上加载 MPI 任务。

- SLURM_NODELIST：分配的节点列表
- SLURM_NNODES：分配的任务数
- SLURM_NPROCS：要加载的任务数
- SLURM_JOBID：作业的 JobID
- SLURM_TASKS_PER_NODE：每节点要加载的任务数
- SLURM_JOB_ID：作业的 JobID

- SLURM_SUBMIT_DIR：提交作业时的工作目录
- SLURM_JOB_NODELIST：作业分配的节点列表
- SLURM_JOB_CPUS_PER_NODE：每个节点上分配给作业的 CPU 数
- SLURM_JOB_NUM_NODES：作业分配的节点数

4.5 作业的 Epilog 和 Prolog

管理员可以配置在作业的生命周期中，资源管理系统在特定时刻执行一些程序，以进行初始化或清理工作，或对作业进行记录等。作业的 Prolog 和 Epilog 均以 root 身份执行。

4.5.1 PrologSlurmctld

当为作业分配资源后，控制进程在一个独立的线程中执行 PrologSlurmctld 程序。该程序执行时无参数，但可通过环境变量获取作业信息：

- SLURM_JOB_ID：作业 JobID
- SLURM_JOB_NAME：作业名字
- SLURM_JOB_PARTITION：作业所在分区
- SLURM_JOB_NODELIST：作业分配的节点列表
- SLURM_JOB_ACCOUNT：作业的计费帐号
- SLURM_JOB_UID：作业的用户 UID。
- SLURM_JOB_USER：作业的用户名字。
- SLURM_JOB_GID：作业的组 GID。
- SLURM_JOB_GROUP：作业的组名字

此外，如果使用了 SPANK 插件，则相应的 SPANK 选项对应的环境变量也可用。

在此程序执行过程中，分配给作业的节点状态被标记为 POWER_UP，以示正在为作业准备节点。

如果 PrologSlurmctld 程序执行失败（退出代码非 0），如果作业不能被重排队，则作业将被直接终止；否则将把作业重新排队，以进行重试。但如果作业连续执行 PrologSlurmctld 程序失败，则作业仍将被终止。

4.5.2 EpilogSlurmctld

当作业终止，释放所分配的节点后，控制进程在一个独立的线程中执行 EpilogSlurmctld 程序。同 PrologSlurmcltd 类似，该程序执行时无参数，从环境变量中获取作业信息。

4.5.3 Prolog

当作业第一次在计算节点上加载进程（计算任务，或者是批处理脚本）时，节点监控进程 `slurmd` 将在加载任务或批处理脚本之前执行 Prolog 程序。该程序执行时无参数，可通过环境变量获取作业信息：

- SLURM_JOB_ID：作业 JobID
- SLURM_JOB_UID：作业的用户 UID
- SLURM_JOB_USER：作业的用户名

此外，如果使用了 SPANK 插件，则相应的 SPANK 选项对应的环境变量也可用。

如果作业的 Prolog 执行失败（退出代码非 0），则作业的进程不会被加载，相应的节点将被设置为 DOWN 状态，原因为“Prolog failed”。

4.5.4 Epilog

当作业的所有进程在计算节点上都终止后，节点监控进程将执行 Epilog 程序。类似于 Prolog，该程序执行时无参数，从环境变量中获取作业信息。

如果作业的 Epilog 执行失败，则相应的节点将被设置为 DOWN 状态，原因为“Epilog error”。

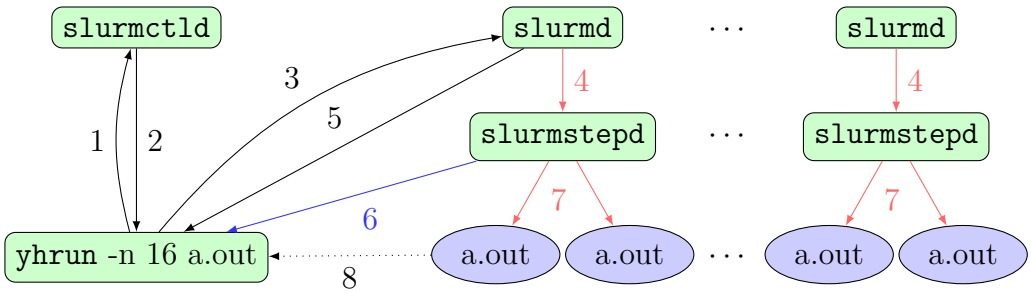
TODO: 强调 prolog 和 epilog 的区别：prolog 只有在加载任务时才执行。

第五章
作业步与任务加载

在分配资源后，用户可以在计算节点上加载任务进行计算。资源管理系统提供了快速加载大规模并行任务的机制。yhrun 命令用于创建作业步并加载用户任务。同时，高性能计算机允许用户登录计算节点，以支持更多应用的运行。

5.1 作业步任务加载

作业步任务是指通过 yhrun 命令加载的用户任务。作业步任务加载流程如图 5.1所示。



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 传递连接　7. 派生任务　8. PMI 连接

图 5.1: 作业步任务加载流程

5.1.1 作业步创建

作业步是一个单独的资源管理系统实体。yhrun 命令在加载任务之前，向控制进程请求创建作业步。作业步的创建涉及到资源分配、任务布局、证书创建等。

资源分配

资源分配是指从分配给作业的节点中，选择可用的节点和处理器分配给作业步。这相当于对资源进行二次分配。在为作业步分配资源时，将考虑已经被作业中其他作业步使用的资源，并尽可能地将作业的任务均衡分配到各节点和处理器上。

资源分配不仅选择可用的节点，还选择 CPU 数目。对于批处理作业和分配模式作业中的作业步，将仅分配足够作业步使用的 CPU，以保留资源供作业中可能存在的其它作业步使用；而对于交互作业，因为只可能有一个作业步，所以该作业步将获得作业所分配的所有 CPU。

因此，同样的加载命令，在交互作业和批处理作业中可能得到不同的资源分配结果。例如，假设每个节点具有 8 个 CPU，则如下交互作业运行命令将会把所分配的每个节点上的 8 个 CPU 都分配给作业步：

```
$ yhrun -N 128 -n 128 --exclusive -t 100 ./xhpl
```

而如下的批处理作业，在脚本中的加载任务时产生的作业步，仅在每个节点上分配 1 个 CPU：

```
$ cat job.sh
#!/bin/sh
yhrun -n 128 -t 100 ./xhpl
$ yhbatch -N 128 -n 128 --exclusive -t 100 ./job.sh
```

如果所运行的程序 xhpl 是多线程的，例如每个任务为 8 线程，则批处理作业中的每个节点上的 8 个线程会被绑定在 1 个 CPU 上执行，导致执行缓慢。解决的办法之一是指定 yhrun 的参数“-c 8”，从而告知系统每个任务需要 8 个 CPU。

可以通过 yhrun 命令行选项设置的作业步资源分配参数包括：

- 节点数目，-N *min*[-*max*]

  至少为作业步分配 *min* 个节点，最多 *max* 个。*max* 可以省略，如果仅指定了 *min*，则表示至少为作业步分配 *min* 个节点。

  yhrun 的 -N 选项仅指定一个参数值 *min* 时，在为作业分配节点和为作业步分配节点时的含义不同。参见第 4.2.1节。

- 处理器数目，-n *ntasks*；-c *cpuspertask*

若为交互作业，将作业分配的所有 CPU 都分配给作业步。对批处理和分配模式作业，如果设置了过度使用处理器，则给作业步的每个节点分配 1 个处理器；否则分配 *ntasks*×*cpuspertask* 个处理器。

- 至少要包含的节点，`-w` *nodelist*

  分给作业步的节点中至少要包含指定的节点。

- 是否共享资源，`--exclusive`

  如果设置，则此作业步不与其它作业步共享资源（处理器与内存）。否则，当多个作业步运行在重叠的节点上时，它们可能过度使用节点上的处理器和内存。

- 是否过度使用处理器，`-O`

  如果设置，则可以在 1 个处理器上加载多于 1 个的任务。

- 内存数量，`--mem-per-cpu` *num*

  为作业步分配的每个处理器，所要分配的内存数量（MB）。

- 运行时间，`-t` *minute*

  超出运行时间的作业步将被终止。

任务布局

任务布局是指在分配给作业步的节点上分布要加载的任务数目与编号。资源管理系统支持多种任务布局方式，详情请参见第 5.1.6节。

证书创建

创建作业步时，`slurmctld` 将会为作业步创建一个证书，并返回给 `yhrun`。`yhrun` 在请求计算节点加载任务时，需要提供此证书，以向计算节点认证，此次任务加载是经 `slurmctld` 授权的。证书中包含了时间、节点、用户等信息，并采用公钥加密，以避免证书的伪造、重用等。

5.1.2 任务状态

在加载任务之后，`yhrun` 对 SIGINT 信号进行特殊处理。对一秒钟之内收到的第一次 SIGINT，`yhrun` 将显示各任务的状态；对一秒钟之内收到的第二次 SIGINT，将导致 `yhrun` 将此信号传递到所有任务，并终止作业步。

此功能一般用于交互模式作业，因为通常在终端通过 Ctrl+C 键可向前台进程发送 SIGINT 信号。此功能可通过 `yhrun` 的 `--disable-status` 选项禁用。

任务的可能状态包括：

- failed to start：加载失败。
- running：运行中。
- exited abnormally：已结束，退出代码非 0。
- exited：已结束，退出代码为 0。
- unknown：未知，还未收到来自节点的加载响应消息。

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
...
```

5.1.3 任务 I/O

I/O 传递

默认情况下，作业步所有任务的标准输出将传递到 yhrun 的标准输出，所有任务的标准错误将传递到 yhrun 的标准错误，而 yhrun 的标准输入将被广播到所有任务的标准输入。任务的标准 I/O 传递方式可以通过 yhrun 的 --input，--output，--error 选项分别控制。这些选项的参数值可为：

- `all`：默认行为。即标准输出/标准错误从所有任务传递到 `yhrun`，标准输入从 `yhrun` 传递到所有任务。
- `none`：不传递 I/O。
- *taskid*：仅把指定任务的标准输出/标准错误传递到 `yhrun`，或仅将 `yhrun` 的标准输入传递到指定的任务。
- *filename*：将任务的标准输出/标准错误写入文件，或从文件读取任务的标准输入。文件名中可以包含特定的格式串并被转换为任务 ID，节点名字等，可用于将不同任务的输出写入不同的文件中。支持的格式转换附包括：
  - `%s`：作业步 ID（不含作业 ID）。
  - `%t`：任务 ID，从 0 开始。
  - `%n`：节点 ID，即当前节点在作业步分配的节点中的编号，从 0 开始。
  - `%N`：节点主机名字。
  - `%j`：作业 ID。
  - `%J`：含作业 ID 的作业步 ID。

任务的 I/O 传递

将任务的标准输出写入文件 result：

```
$ yhrun -n 16 -o result a.out
```

将每个任务的标准输出写入不同文件中：

```
$ yhrun -n 16 -o result-%t a.out
```

任务从文件 input 读取标准输入：

```
$ yhrun -n 16 -i input a.out
```

任务标号

为了区分作业步的标准输出或标准错误由哪个任务生成，可以设置 `yhrun` 的 `-l,--label` 选项，使 `yhrun` 在显示任务的标准输出和标准错误前加上任务号。

eg 标准输出加任务号

```
$ yhrun -N 4 hostname
cn0
cn1
cn2
cn3
$ yhrun -l -N 4 hostname
0: cn0
1: cn1
2: cn2
3: cn3
```

伪终端

用户可以通过 yhrun 的 --pty 选项，在伪终端中执行 0 号任务。

eg 在伪终端中执行 0 号任务

没有伪终端时 ls 顺序列出目录中的文件；有伪终端时按窗口宽度适当排列：

```
$ 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
```

没有伪终端时 top 命令不能执行：

```
$ 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
...
```

指定 --pty 选项后，将仅从 0 号任务传递标准 I/O，且不能指定 --input，--output，--error 选项。

附接 I/O

除了将标准输出/标准错误传递到加载任务的 yhrun 或者文件之外，用户可以在作业步运行过程中，通过 yhattach 命令附接查看任务的标准输出和标准错误。

eg 附接查看作业步任务输出

```
$ yhqueue -s
STEPID         NAME PARTITION     USER      TIME NODELIST
405.0       ft.C.16      work  test605      0:10 cn[479-480]
$ yhattach 405.0

NAS Parallel Benchmarks 3.3 -- FT Benchmark

No input file inputft.data. Using compiled defaults
Size                :  512x 512x 512
Iterations          :             20
Number of processes :             16
Processor array     :          1x  16
Layout type         :             1D
T =    1     Checksum =    5.195078707457D+02    5.149019699238D+02
T =    2     Checksum =    5.155422171134D+02    5.127578201997D+02
T =    3     Checksum =    5.144678022222D+02    5.122251847514D+02
...
```

此功能可用于查看批处理作业中的作业步任务的标准输出。

5.1.4 信号传递

在 yhrun 加载任务之前，对信号的处理是标准的；SIGINT 将直接把 yhrun 终止。而在加载任务之后，yhrun 将对下列信号做特殊处理。

- SIGINT：一秒钟之内的第一次 SIGINT 将显示所加载任务的状态，一秒钟内的第二次 SIGINT 将传递到所有任务并强制终止作业步。参见第 5.1.2节
- SIGQUIT、SIGTERM、SIGHUP：强制终止作业步。
- SIGCONT：忽略。
- SIGUSR1、SIGUSR2、SIGALRM：传递到所有任务。

此外，用户可以通过 yhcancel 命令给作业步的所有任务发送信号，详情请参见第6.2节。

5.1.5 任务执行环境

环境变量

yhrun 运行时的环境变量将会被传递到计算节点上任务的执行环境中。因此，若想在程序运行时设置某环境变量，只需要加载程序时对 yhrun 的环境进行设置即可。

设置程序执行时的环境变量

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

此外，资源管理系统会在任务执行前设置一些环境变量，以反映相关的参数：

- SLURM_TASK_PID：任务的进程 ID
- SLURM_NPROCS：作业步加载的任务数
- SLURM_CPUS_PER_TASK：每任务需要的 CPU 数
- SLURM_NTASKS_PER_NODE：每节点上的任务数
- SLURM_NTASKS_PER_SOCKET：每个处理器 Socket 上的任务数
- SLURM_NTASKS_PER_CORE：每个处理器 Core 上的任务数
- SLURM_CPUS_ON_NODE：节点上的 CPU 数
- SLURM_DISTRIBUTION：任务布局方式，参见第 5.1.6节
- SLURM_DIST_PLANESIZE：基于面的布局方式中面大小
- SLURM_DIST_LLLP：逻辑处理器布局
- SLURM_CPU_BIND_VERBOSE：CPU 绑定是否输出详细信息
- SLURM_CPU_BIND_TYPE：CPU 绑定类型
- SLURM_CPU_BIND_LIST：CPU 绑定列表
- SLURM_CPU_BIND：CPU 绑定信息

- SLURM_MEM_BIND_VERBOSE：内存绑定是否输出详细信息
- SLURM_MEM_BIND_TYPE：内存绑定类型
- SLURM_MEM_BIND_LIST：内存绑定列表
- SLURM_MEM_BIND：内存绑定信息
- SLURM_OVERCOMMIT：是否过度使用资源
- SLURMD_DEBUG：`slurmd` 和 `slurmstepd` 调试信息发送回 `yhrun` 的详细级别
- SLURM_LABELIO：任务的标准输出/标注错误是否附加任务号
- SLURM_JOB_ID：作业的 ID
- SLURM_NODEID：节点 ID，即当前节点在作业分配的节点列表中的位置编号，从 0 开始
- SLURM_PROCID：进程 ID，即任务号，从 0 开始
- SLURM_LOCALID：本地 ID，即进程在本节点上的任务号，从 0 开始
- SLURM_STEPID：作业步 ID
- SLURM_NNODES：作业步使用的节点数
- SLURM_NODELIST：作业步使用的节点列表
- SLURM_TASKS_PER_NODE：同 SLURM_NTASKS_PER_NODE
- SLURM_SRUN_COMM_PORT：`yhrun` 的通信端口，用于 PMI 连接
- SLURM_LAUNCH_NODE_IPADDR：`yhrun` 所运行的节点的地址
- SLURM_GTIDS：本节点上所有任务的任务号
- SLURM_PTY_PORT：伪终端端口
- SLURM_PTY_WIN_COL：伪终端窗口列数
- SLURM_PTY_WIN_ROW：伪终端窗口行数
- SLURM_CHECKPOINT_IMAGE_DIR：检查点映象文件保存目录
- SLURM_RESTART_COUNT：作业步重启次数
- SLURMD_NODENAME：本节点的节点名

资源限制

资源管理系统可以将 yhrun 进程的资源限制传递到计算节点上的任务进程。

在系统配置文件中，可以指定默认地传递哪些资源限制：通过 PropagateResourceLimits 配置项指定要传递的资源限制，或通过 PropagateResourceLimitsExcept 指定不传递的资源限制。可用的资源限制关键字如下：

- NONE：不设置任何资源限制
- ALL：所有资源限制
- CPU：CPU 时间
- FSIZE：文件大小
- DATA：数据段大小
- STACK：栈段大小
- CORE：Core Dump 文件大小
- RSS：物理内存大小（Linux 下未实现强制限制）
- NPROC：用户进程数
- NOFILE：打开文件数
- MEMLOCK：锁定内存大小
- AS：内存空间大小

yhrun 的 --propagate 选项可用来覆盖系统的默认配置。

传递资源限制

为作业任务设置 memlock 资源限制：

```
$ ulimit -l 1024000
$ yhrun --propagate MEMLOCK -n 16 a.out
```

某些 MPI 实现需要锁定内存作为通信使用的缓存，若 MEMLOCK 的限制太小，程序运行时会报错退出。如果程序需要在栈上使用大量空间，如函数嵌套调用或分配临时变量，则 STACK 限制不能太小，否则程序可能报错退出或 Segmentation Fault。管理员通过修改登录节点的 /etc/security/limits.conf 文件为用户设置登录后的默认资源限制。

进程优先级

资源管理系统可以将 yhrun 进程的优先级传递到计算节点上的任务进程。系统配置文件中的 PropagatePrioProcess 控制是否进行此传递。另外，yhrun 的 --nice 选项可用来设置任务进程的优先级。

eg 以低优先级运行程序

```
$ yhrun --nice 10 -n 16 a.out
```

工作目录

默认地，资源管理系统将任务进程的工作目录设置为 yhrun 进程的工作目录，但是用户可以通过 yhrun 的 --chdir 选项来设置远程任务的工作目录。

eg 设置任务的工作目录

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

5.1.6 任务布局

任务布局指任务在作业步所分配的节点上的分布，包括任务数分布与任务号分布。资源管理系统支持多种任务布局方式，yhrun 的 --distribution 选项可以指定作业步的任务布局方式。

在本节下面的例子中，假设系统中四个节点 cn[0-3] 每个节点有两个双核处理器。为了展示不同布局方式的结果，使用的简单的示例程序输出程序的任务号以及所在节点的主机名字。程序可以是使用 YH-MPI 编译的 MPI 程序，或 Shell 脚本程序。

示例程序代码

MPI 程序代码：

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[])
{
  int rank, size, len;
  char node[MPI_MAX_PROCESSOR_NAME];
  MPI_Init(0, 0);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Get_processor_name(node, &len);
  printf("This is process %d of %d running on %s\n", rank, size, node);
  MPI_Finalize();
  return 0;
}
```

Bash 脚本代码：

```
#!/bin/sh
echo "This is process $SLURM_PROCID of $SLURM_NPROCS running on `hostname`"
```

- 循环布局：--distribution cyclic

  系统在每个节点上分配任务数的方法是，在所分配的处理器数目范围内，尽可能将任务在节点间按可用的处理器数均衡分配。例如，作业分配了四个节点 cn[0-3]，在 cn0 和 cn3 上分配了四个处理器，在 cn1 和 cn2 上分配了两个处理器。则循环布局方式下，不同任务数时的分布结果如下：

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

系统首先按可用处理器在各节点间轮转分配任务，当所有处理器都已分配或，则按节点轮转分配任务。各任务的编号也依上述方法进行。

循环任务布局

```
$ yhrun -N 4 -n 16 -m cyclic hw | sort -n -k 4
This is process 0 of 16 running on cn0
This is process 1 of 16 running on cn1
This is process 2 of 16 running on cn2
This is process 3 of 16 running on cn3
This is process 4 of 16 running on cn0
This is process 5 of 16 running on cn1
This is process 6 of 16 running on cn2
This is process 7 of 16 running on cn3
This is process 8 of 16 running on cn0
This is process 9 of 16 running on cn1
This is process 10 of 16 running on cn2
This is process 11 of 16 running on cn3
This is process 12 of 16 running on cn0
This is process 13 of 16 running on cn1
This is process 14 of 16 running on cn2
This is process 15 of 16 running on cn3
```

- 块布局：`--distribution block`

  块布局和循环布局方式的不同之处仅在于任务号的分布上；这两种布局方式下，各节点上运行的任务数是相同的。在分配任务号时，先将一个节点内的任务全部编号，再依次给下一个节点上的任务编号。延上例，16 个和 10 个任务时块布局结果分别如下：

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

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

块任务布局

```
$ yhrun -N 4 -n 16 -m block hw | sort -n -k 4
This is process 0 of 16 running on cn0
This is process 1 of 16 running on cn0
This is process 2 of 16 running on cn0
This is process 3 of 16 running on cn0
This is process 4 of 16 running on cn1
This is process 5 of 16 running on cn1
This is process 6 of 16 running on cn1
This is process 7 of 16 running on cn1
This is process 8 of 16 running on cn2
This is process 9 of 16 running on cn2
This is process 10 of 16 running on cn2
This is process 11 of 16 running on cn2
This is process 12 of 16 running on cn3
This is process 13 of 16 running on cn3
This is process 14 of 16 running on cn3
This is process 15 of 16 running on cn3
```

- 基于面的布局：`--distribution plane=`*plane_size*

  基于面的布局是一种块循环布局，块大小为指定的 *plane_size*。延上例，16 个任务，块大小为 3 时基于面的布局结果如下：

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

基于面的任务布局

```
$ yhrun -N 4 -n 16 -m plane=3 hw | sort -n -k 4
This is process 0 of 16 running on cn0
This is process 1 of 16 running on cn0
This is process 2 of 16 running on cn0
This is process 3 of 16 running on cn1
This is process 4 of 16 running on cn1
This is process 5 of 16 running on cn1
This is process 6 of 16 running on cn2
This is process 7 of 16 running on cn2
This is process 8 of 16 running on cn2
This is process 9 of 16 running on cn3
This is process 10 of 16 running on cn3
This is process 11 of 16 running on cn3
This is process 12 of 16 running on cn0
This is process 13 of 16 running on cn0
This is process 14 of 16 running on cn0
This is process 15 of 16 running on cn1
```

基于面的布局在节点间分配任务数时不进行负载均衡，也不考虑硬件的处理器数，仅按用户指定的块大小进行循环分配。

- 逻辑处理器布局：`--distribution [cyclic|block]:[cyclic|block]`

冒号前面为任务在节点间的布局方式；冒号后面为任务在节点内逻辑处理器间的布局方式，用于自动生成 CPU 绑定掩码。

- 任意布局：`--distribution arbitrary|hostfile`

用户直接指定各任务分别运行在哪个节点上。具体而言，用户编写一个节点列表文件，每个节点一行，然后指定 `yhrun` 的 `--nodelist` 选项，参数为此文件。文件行数即包含的节点数必须与作业步要加载的任务数相同。作业步的 0 号任务运行在文件中的第一个节点上，1 号任务运行在第二个节点上，依此类推。

eg 任意任务布局

节点列表文件 hostlist 内容：

```
cn0
cn1
cn0
cn1
cn0
cn1
cn2
cn3
cn2
cn3
cn2
cn3
cn0
cn1
cn2
cn3
```

任务布局结果：

```
$ yhrun -N 4 -n 16 -w ./hostlist -m arbitrary hw | sort -n -k 4
This is process 0 of 16 running on cn0
This is process 1 of 16 running on cn1
This is process 2 of 16 running on cn0
This is process 3 of 16 running on cn1
This is process 4 of 16 running on cn0
This is process 5 of 16 running on cn1
This is process 6 of 16 running on cn2
This is process 7 of 16 running on cn3
This is process 8 of 16 running on cn2
This is process 9 of 16 running on cn3
This is process 10 of 16 running on cn2
This is process 11 of 16 running on cn3
This is process 12 of 16 running on cn0
This is process 13 of 16 running on cn1
This is process 14 of 16 running on cn2
This is process 15 of 16 running on cn3
```

5.1.7 多程序作业步

通过指定 --multi-prog 选项，可使用 yhrun 加载多程序作业步，即不同的任务执行不同的程序和/或参数。这时，yhrun 命令的参数中，最后跟的不是要执行的程序，而是一个配置文件，其中注明每个任务需要加载的程序。

配置文件按行组织，每行用空白分隔为若干个字段。第一个字段为任务号部分，可以包含逗号分隔的任务号列表，可以用“*min*-*max*”表示任务号范围，或在最后一行用“*”表示其余所有的任务号。第二个字段为所指定的任务要执行的程序。其它字段为执行程序时的参数。在程序和参数中，可以使用“%t”和“%o”，分别表示替换为任务的任务号和任务在配置文件该行所指定的任务号中的偏移。

eg 多程序作业步

配置文件 mp.conf

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

加载作业步

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

所有任务执行的程序分别为：

- 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

5.1.8 作业步终止

在加载的任务执行结束退出后，计算节点上的 `slurmstepd` 将会向 `slurmctld` 和 `yhrun` 发送通知消息。全部任务都退出后，作业步将会终止。

一般地，对于 MPI 程序，如果有一个任务异常退出，则其余任务将因为不能与其通信而导致程序不能继续往下执行；但很多情况下这些任务并不退出，而是阻塞等待在通信部分。这将不能及时释放作业分配的节点，直到作业的运行超时，造成计算资源的浪费。为此，资源管理系统提供了一种处理机制。用户可以选择在第一个任务退出（无论是否异常退出）一段时间之后，即强行终止作业步。等待的时间（秒数）通过 `yhrun` 的 `--wait` 选项指定；等待时间为 0 表示不强制终止作业步。系统配置文件的 `WaitTime` 参数设置等待时间的默认值。

类似地，用户可以通过 `yhrun` 的 `--kill-on-bad-exit` 选项，选择在有任务异常退出，即退出代码非 0 时，强制终止作业步。

用户可以通过 `yhcancel` 命令取消作业步，终止其执行。

5.1.9 任务的 Prolog 与 Epilog

类似于作业，作业步任务在执行时也可以有 Prolog 和 Epilog。

- SrunProlog

  通过 `yhrun` 的 `--prolog` 选项设置，系统配置文件中的 SrunProlog 参数为其默认值。由 `yhrun` 在加载作业步前执行。此程序可能在登录节点（交互/分配模式作业）或计算节点（批处理作业）上执行。

- SrunEpilog

  通过 `yhrun` 的 `--epilog` 选项设置，系统配置文件中的 SrunEpilog 参数为其默认值。由 `yhrun` 在作业步结束后执行。此程序可能在登录节点（交互/分配模式作业）或计算节点（批处理作业）上执行。

- 系统 TaskProlog

  由系统配置文件中的 TaskProlog 参数设置。由 `slurmstepd` 在每个用户任务开始执行之前执行。此程序的输出被按行解析，符合如下格式的输出将被处理，其余输出被忽略：

  - `print` *string*：将 *string*（直到行尾）写到任务的标准输出。
  - `export` *var*=*val*：设置任务的环境变量 *var* 值为 *val*
  - `unset` *var*：取消任务的环境变量 *var*

- 系统 TaskEpilog

  由系统配置文件中的 TaskEpilog 参数设置。由 `slurmstepd` 在每个用户任务执行结束之后执行。

- 用户 TaskProlog

  由 `yhrun` 的 `--task-prolog` 选项设置。由 `slurmstepd` 在每个用户任务开始执行之前，系统 TaskProlog 执行之后执行。此程序的输出作类似系统 TaskProlog 的处理。

- 用户 TaskEpilog

  由 `yhrun` 的 `--task-epilog` 选项设置。由 `slurmstepd` 在每个用户任务执行结束之后，系统 TaskEpilog 执行之前执行。

5.1.10 批处理作业步

对于批处理作业，分配资源后系统将在为其分配的第一个节点上加载执行作业脚本。作业脚本的加载执行过程与用户任务的加载执行过程有很多相似之处，对于计算节点而言，类似与有一个批处理作业步；但在控制进程中并没有相应的作业步实体。

批处理作业步可以被发送信号、通过 `yhbatch` 的选项控制其 I/O 传递、进行 CPU 绑定、执行 TaskProlog/TaskEpilog 等。

5.2 登录计算节点

用户在分配资源后，可以使用 SSH 登录所分配的计算节点，然后加载任务或查看节点状态等。但这样加载的任务不受资源管理系统直接管理控制，不会自动进行 CPU 绑定。

通过系统配置文件中的作业 Epilog 机制，资源管理系统保证当用户在节点上的所有作业都结束，即用户不再有作业分配该节点后，其在相应节点上的所有进程将被终止。

5.3 MPI 程序加载

5.3.1 MPICH2 及其派生版本

MPICH2，及其派生版本包括 MVAPICH2、YH-MPI 等，使用 PMI 接口进行进程管理。资源管理系统中集成了 PMI 实现，故这些 MPI 环境编译的程序可以通过 `yhrun` 直接作为作业步任务加载。

安装这些 MPI 时，在配置阶段可能需要对 `configure` 指定 `--without-pm` 和 `--with-pmi=slurm` 选项。

5.3.2 Open MPI

Open MPI 编译的程序使用 `mpiexec` 命令加载。用户首先通过分配模式或批处理作业分配资源，然后运行 `mpiexec` 命令或在脚本中用 `mpiexec` 加载 MPI 程序。

如果在编译安装 Open MPI 时加入了 SLURM 支持（`--with-slurm`），则 `mpiexec` 可自动感知所分配的资源、要加载的任务数等信息，并通过 `yhrun` 加载 `orted`。这样，每次加载程序时，都将产生一个作业步，只是作业步运行的程序是 `orted`。如果编译 Open MPI 时没有加入 SLURM 支持，则需要用户自己从环境变量中获取分配的节点，指定要加载的任务数；这种情况下 `orted` 通过 SSH 加载，不会产生作业步。参见第 5.3.3节。

5.3.3 其他 MPI

加载 MPI 程序的通用方法是，通过分配模式或批处理作业进行资源分配，然后利用 MPI 运行环境提供的 `mpiexec` 命令加载程序执行。`mpiexec` 一般通过 SSH 在计算节点上加载程序或进程管理程序（MPD、`orted` 等）。

在分配模式和批处理作业中，用户可以通过环境变量来获取所分配的节点、要加载的任务数等信息，具体环境变量请参见第 4章。

第六章
作业控制

资源管理系统提供了一系列的功能对作业进行控制。一般地，用户仅能控制自己的作业，而 root 可以控制任意用户的作业。

6.1 取消作业

用户可以使用 `yhcancel` 命令取消自己的作业或作业步。

对于排队作业，取消作业将简单地把作业标记为 CANCELLED 状态而结束作业。

对于运行中或挂起的作业，取消作业将终止作业的所有作业步，包括批处理作业脚本，将作业标记为 CANCELLED 状态，并回收分配给作业的节点。一般地，批处理作业将会马上终止；交互作业的 `yhrun` 进程将会感知到任务的退出而终止；分配模式作业的 `yhalloc` 进程不会自动退出，除非作业所执行的用户命令因作业或任务的结束而终止。但是在作业被取消时，控制进程都会发送通知消息给分配资源的 `yhrun` 或 `yhalloc` 进程。用户可以选择通过 `yhalloc` 的 `--kill-command` 选项设置在收到通知时向所执行的命令发送信号将其终止。详情请参见第??节。

对于作业步，取消其执行将给作业步的所有任务发送 SIGKILL 信号，并每隔一段时间重新发送，直到任务退出。两次信号发送间隔的时间（秒数）由系统配置文件中 `KillWait` 参数设置。

有时候会出现用户的程序不能被终止的情形，这时作业将长期处于 CG 状态，不能及时释放资源。资源管理系统允许系统管理员配置在计算节点上出现这种情形时，自动执行某外部程序进行处理。外部程序执行时，环境变量 SLURM_JOB_ID 和 SLURM_STEP_ID 将分别被设置为不能终止的程序的作业和作业步 ID。外部程序和执行外部程序前等待的时间（秒数）分别由系统配置文件中的 `UnkillableStepProgram` 和 `UnkillableStepTimeout` 参数设置。

eg 取消作业

```
$ yhqueue
JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  549      work   job.sh  test605   R       0:42     16 cn[866-881]
  548      work   job.sh  test605   R       0:43     16 cn[845-860]
  547      work   job.sh  test605   R       0:48     16 cn[845-860]
$ yhcancel 548
$ yhqueue
JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  549      work   job.sh  test605   R       0:48     16 cn[866-881]
  547      work   job.sh  test605   R       0:54     16 cn[845-860]
```

yhqueue 输出中状态为 CG 的作业是具有 COMPLETING 标志的作业，这些作业已经运行结束，只是在等待确认其任务及相关进程的退出。不需要再用 yhcancel 取消这些作业，yhcancel 也不能去除其 COMPLETING 标志。

除了指定作业 ID 外，用户可以通过 yhcancel 提供的一系列过滤选项来选择要取消的作业，包括选择作业的用户、计费帐号、分区、状态等等。详情请参见第??节。

eg yhcancel 的过滤选项

取消用户 test 的所有作业：

```
$ yhcancel -u test
```

取消分区 debug 中排队状态的所有作业：

```
$ yhcancel -p debug -t pd
```

6.2 信号发送

用户可以使用 yhcancel 命令给作业或作业步发送信号。如果指定的是作业步，则给作业步的所有任务发送指定的信号；如果指定的是作业，则信号发送到作业所有正在运行

的作业步的所有任务。对于批处理作业，还可以选择将信号仅发送到批处理脚本进程。

要发送的信号通过 yhcancel 的 --signal 选项指定，参数可以为信号数值，或者下列名字之一（大小写无关）：

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

同样，用户可以通过过滤选项选择要发送信号的作业或作业步。

eg 向作业发送信号

向作业 123 的所有作业步的所有任务发送 SIGUSR1 信号：

```
$ yhcancel -s usr1 123
```

向作业步 456.1 的所有任务发送 SIGUSR1 信号：

```
$ yhcancel -s usr1 456.1
```

向批处理作业 789 的脚本进程发送 SIGUSR1 信号：

```
$ yhcancel -b -s usr1 789
```

6.3 重新排队

用户可以将自己的正在运行或挂起的批处理作业重新排队。作业重排队通过 yhcontrol 命令进行。重排队将把作业置为 PENDING 状态，释放作业所分配的资源。重新排队的作业在可以分配资源运行之前，系统将强制其等待一段时间（10 秒）。

重新排队不会为作业重新分配作业 ID，但作业的提交时间将会被设置为其被重排队的时间，从而在记账数据中与原作业区分。原作业在记账数据中的状态被记录为 CANCELLED。

用户在提交作业时可以通过 yhbatch 的 --requeue 或 --no-requeue 设置作业是否可以被重排队。系统配置文件的 JobRequeue 选项设置作业是否可以被重排队的默认值。

eg 作业重排队

将运行中的作业重新排队：

```
$ yhqueue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  22852     work1   job.sh    hjcao   R       0:03     11 cn[507-517]
$ yhcontrol requeue 22852
$ yhqueue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  22852     work1   job.sh     root  PD       0:00      2 (BeginTime)
```

一段时间后作业分配资源重新运行：

```
$ yhqueue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  22852     work1   job.sh     root   R       2:42     11 cn[463-473]
```

重排队的批处理作业在分配资源后将会重新运行，即系统将在为其分配的首个节点上重新加载批处理作业的脚本。如果需要作业继续运行，请使用检查点/恢复功能，参见第 7.2节。

6.4 修改作业

在作业提交后，用户可以使用 yhcontrol 命令对其某些参数进行修改。根据作业所处的状态不同，所能够的修改参数也不同。

只能在 PENDING 状态下修改的参数

- Account：作业的计费帐号
- Contiguous：作业是否需要连续节点
- Dependency：作业的依赖关系
- EligibleTime/StartTime：作业的开始时间
- ExcNodeList：不能分配给作业的节点

- Features：分配给作业的节点的特性
- MinCPUsNode：每个节点最少的 CPU 数
- MinMemoryCPU：每个 CPU 最少的内存数
- MinMemoryNode：每个节点最少的内存数
- MinTmpDiskNode：每个节点最少的临时磁盘空间数
- Name：作业名字
- PartitionName：作业所在的分区
- ReqCores：作业最少需要的处理器 Core 数
- ReqNodeList：必须分配给作业的节点
- ReqNodes/NumNodes：作业请求的节点数
  格式为 $min[\text{-}max]$
- ReqProcs/NumTasks：作业要加载的任务数
- ReqSockets：作业最少需要的处理器 Socket 数
- ReqThreads：作业最少需要的处理器 Thread 数
- Shared：作业是否可以和其它作业共享节点
- TasksPerNode：每节点的任务数
- WCKey：作业的负载特性关键词

只能在运行状态修改的参数

- EndTime：作业的结束时间
  用于设置作业的时间限制。
- ReservationName：作业使用的预约

能够在排队、运行、挂起状态修改的参数

- TimeLimit：作业运行时间限制。

  只有 root 用户能够在作业运行或挂起时增大其运行时间限制，普通用户只能减小自己作业的运行时间限制。

- Nice：作业优先级偏移

  普通用户不能提高作业的优先级，故只能设置正的 Nice 值。参见第 12章。

- Priority：作业优先级

  只有 root 用户能够增高作业的优先级，普通用户只能降低自己作业的优先级。

  修改作业的优先级时，作业的优先级偏移被清零，且不再由系统根据作业属性为作业动态计算优先级，而是将其优先级固定为设置的值。如果设置的作业优先级为 0xFFFFFFFF，则取消设置的优先级，恢复由系统计算。

- Licenses：作业需要的许可证

能够在所有状态修改的参数

- Comment：作业的注释信息

- Requeue：是否允许作业重排队

在修改作业的参数时，系统要进行类似于提交作业时的访问权限、资源限制等检查。

改变排队作业的分区

```
$ yhqueue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
   1252      work     xhpl     root  PD      00:00    768 (Resources)
$ yhcontrol update jobid=1252 partition=debug
$ yhqueue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
   1252     debug     xhpl     root   R      00:06    768 cn[0-767]
```

eg 改变已运行作业的运行时间

```
$ yhcontrol show job 3284 | grep TimeLimit
AllocNode:Sid=ln0:31478 TimeLimit=10000 ExitCode=0:0
$ yhcontrol update jobid=3284 timelimit=3000
$ yhcontrol show job 3284 | grep TimeLimit
AllocNode:Sid=ln0:31478 TimeLimit=3000 ExitCode=0:0
```

eg 改变排队作业的名字

```
$ yhqueue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
 295376      work     xhpl     root   R    1:54:04    255 cn[256-453,455-511]
 295375      work     xhpl     root  PD      00:00    768 (Resources)
$ yhcontrol update jobid=295375 name=myjob
$ yhqueue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
 295376      work     xhpl     root   R    1:54:04    255 cn[256-453,455-511]
 295375      work    myjob     root  PD      00:00    768 (Resources)
```

第七章
作业检查点

资源管理系统支持对作业进行检查点/恢复，以在节点发生故障时进行续算。高性能计算机一般配置使用 BLCR 检查点支撑环境。

7.1 作业步检查点

7.1.1 yhrun 加载任务

对于用 yhrun 加载的作业步任务，使用 yhcontrol 命令对作业步请求检查点：

yhcontrol checkpoint [create|vacate] *jobid*[.*stepid*]

“checkpoint create”仅请求作业/作业步进行检查点，“checkpoint vacate”命令请求作业/作业步进行检查点后终止。详细命令格式及选项请参见第??节。

如果给出了作业步 ID，则对指定的作业步进行检查点；否则对指定的作业进行检查点。如果作业为批处理作业，则对整个批处理作业及其脚本进行检查点操作，参见第7.2节；否则对作业中所有的作业步进行检查点。

加载作业步的 yhrun 命令的 --checkpoint-dir=*dir* 选项指定作业步检查点的映象文件的保存目录，其默认值为作业步任务的工作目录。每次请求检查点时，可以通过 yhcontrol 命令的 ImageDir=*dir* 选项覆盖作业步的检查点映象文件保存目录。由于任务进程的检查点/恢复在计算节点上进行，因此用户需要注意，所指定的检查点映象文件保存路径为计算节点上的路径。

系统将在检查点映象文件保存目录下创建名字为作业步 ID 的目录，并将任务的映象文件保存到该目录中。对每个任务，生成的映象文件名字为 task.*taskid*.ckpt。

请求检查点时可以通过 MaxWait=*seconds* 指定等待检查点完成的最长时间，默认为 15 秒。如果在指定时间内没有完成，则认为检查点操作失败。

eg 作业步的检查点

运行程序：

```
$ yhrun -N 2 -n 4 -l prog
0: process 0, sum: 0
3: process 3, sum: 3
1: process 1, sum: 1
2: process 2, sum: 2
...
```

对作业步进行检查点：

```
$ yhqueue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  21194     work1 job_test    hjcao   R       0:19      2 cn[1-2]
$ yhcontrol checkpoint create 21194.0
```

生成的检查点映象文件：

```
$ ls -l 21194.0/
-r-------- 1 hjcao hjcao 168675 Jul 23 20:55 task.0.ckpt
-r-------- 1 hjcao hjcao 168675 Jul 23 20:55 task.1.ckpt
-r-------- 1 hjcao hjcao 168675 Jul 23 20:55 task.2.ckpt
-r-------- 1 hjcao hjcao 168675 Jul 23 20:55 task.3.ckpt
```

在恢复作业步执行时，用户需要重新加载作业步任务，通过 yhrun 的 --restart-dir=*dir* 选项指定映象文件所在的目录。这样，将生成一个新的作业和/或作业步。

eg 作业步恢复执行

```
$ yhrun -N 2 -n 4 -l --restart-dir=./21194.0 prog
0: process 0, sum: 0
1: process 1, sum: 5
2: process 2, sum: 10
3: process 3, sum: 15
...
```

恢复作业步任务时，--restart-dir 选项指定的目录要指定到系统生成的以作业步 ID 为名字的目录一级。

7.1.2 yhrun_cr 加载任务

用户还可以使用 yhrun_cr 加载作业步。yhrun_cr 是一个包装程序，它将调用 yhrun 进行实际的任务加载；其可用的选项与 yhrun 相同。上述 yhrun 加载任务的检查点/恢复方法完全适用于 yhrun_cr 加载的任务。除此之外，还可以通过 BLCR 的命令工具 cr_checkpoint 请求对 yhrun_cr 进程进行检查点，yhrun_cr 将自动处理所加载任务的检查点。

yhrun_cr 加载作业步任务的检查点

加载作业步：

```
$ yhrun_cr -N 2 -n 4 prog
process 0, sum: 0
process 1, sum: 1
process 3, sum: 3
process 2, sum: 2
...
```

请求检查点：

```
$ ps aux | grep yhrun_cr
hjcao    31073  0.0  0.0  14672   760 pts/0    Sl+  09:46   0:00 yhrun_cr -N 2 -n 4 prog
$ cr_checkpoint 31073
```

生成的映象文件：

```
$ ls -l
drwxrwxr-x 2 hjcao hjcao   4096 Jul 24 09:48 21736.0
-r-------- 1 hjcao hjcao 697893 Jul 24 09:48 context.31073
$ ls -l 21736.0/
-r-------- 1 hjcao hjcao 168675 Jul 24 09:48 task.0.ckpt
-r-------- 1 hjcao hjcao 168675 Jul 24 09:48 task.1.ckpt
-r-------- 1 hjcao hjcao 168675 Jul 24 09:48 task.2.ckpt
-r-------- 1 hjcao hjcao 164579 Jul 24 09:48 task.3.ckpt
```

类似地，通过 BLCR 的 cr_restart 命令恢复 yhrun_cr 进程将自动恢复（重新加载并续算）作业步任务。

eg yhrun_cr 加载作业步的恢复

恢复执行：

```
$ cr_restart context.31073
process 3, sum: 96
process 2, sum: 64
process 1, sum: 32
process 0, sum: 0
...
```

恢复的进程：

```
$ ps aux
...
hjcao    31336  0.0  0.0  12160   584 pts/1    Sl+  09:49   0:00 cr_restart context.31073
hjcao    31340  0.0  0.0 105320  4364 pts/1    Sl   09:49   0:00 /usr/bin/srun -N 2 -n 4 pr
...
```

恢复的作业：

```
JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
21742     work1     prog    hjcao   R       1:18      2 cn[0-1]
```

7.1.3 自动检查点

在加载作业步任务时，用户可以请求系统对作业步进行周期性的自动检查点；yhrun 的 --checkpoint=*minutes* 选项指定两次检查点之间的间隔时间（分钟数）。

eg 作业步的自动检查点

加载作业步：

```
$ yhrun --checkpoint=30 -n 4 prog
...
```

生成的映象文件：

```
$ ls -l 21836.0
-r-------- 1 hjcao hjcao 168675 Jul 24 10:27 task.0.ckpt
-r-------- 1 hjcao hjcao 168675 Jul 24 10:27 task.1.ckpt
-r-------- 1 hjcao hjcao 168675 Jul 24 10:27 task.2.ckpt
-r-------- 1 hjcao hjcao 164579 Jul 24 10:27 task.3.ckpt
```

7.2 批处理作业检查点

批处理作业中可能包含多个作业步，或者作业脚本中包含某些前/后处理命令。这时，仅对作业步进行检查点/恢复不能完全实现对用户透明的作业容错。例如，一个批处理作业中可能包含了两次作业步任务加载；在作业运行过程中，节点发生故障。尽管用户对两次作业步任务加载都请求了自动检查点操作，但若要恢复该作业的运行，就需要用户分析是在那个作业步运行过程中发生的节点故障，修改批处理作业脚本（删除已经运行完成的作业步；对发生故障时正在运行的作业步的加载命令 yhrun 添加 --restart-dir 选项），并重新提交作业。

资源管理系统支持对整个批处理作业进行检查点和恢复。为此，在批处理脚本中用户需要使用 yhrun_cr 替换 yhrun 进行作业步任务的加载。如果批处理脚本包含非 yhrun_cr 的任务加载程序如 mpiexec，则该程序应负责处理所加载的任务的检查点与恢复。

7.2.1 检查点

用户使用 yhcontrol 命令请求对批处理进行检查点操作：

yhcontrol checkpoint [create|vacate] *jobid*

checkpoint create 仅请求作业进行检查点，checkpoint vacate 命令请求作业进行检查点后终止。详细命令格式及选项请参见第??节。

提交批处理作业的 yhbatch 命令的 --checkpoint-dir 选项指定作业检查点的映象

文件的保存目录，其默认值为作业的工作目录。每次请求检查点时，可以通过 yhcontrol 命令的 ImageDir=*dir* 选项覆盖作业的检查点映象文件保存目录。用户需要注意，所指定的检查点映象文件保存路径为计算节点上的路径。

系统将在检查点映象文件保存目录下创建名字为作业 ID 的目录，并将批处理脚本及其子进程的映象文件保存到该目录中，文件名为 script.ckpt。对于脚本中正在运行的通过 yhrun_cr 加载的作业步，其检查点映象文件保存目录为此新创建的目录，即系统将在名字为作业 ID 的目录下创建名字为作业步 ID 的目录，保存所加载作业步任务的映象文件。

类似与作业步检查点，用户可以通过 MaxWait=*seconds* 指定等待检查点完成的最长时间。

批处理作业检查点

批处理脚本 job.sh 内容：

```
#!/bin/sh
yhrun_cr -n 4 prog
```

提交作业：

```
$ sbatch -N 2 -n 4 job.sh
Submitted batch job 21952
```

请求进行检查点：

```
$ yhcontrol checkpoint create 21952
```

生成的检查点映象文件：

```
$ ls -l 21952/
total 1004
drwxrwxr-x 2 hjcao hjcao    4096 Jul 24 16:19 21952.0
-r-------- 1 hjcao hjcao 1017365 Jul 24 16:19 script.ckpt
$ ls -l 21952/21952.0/
total 712
-r-------- 1 hjcao hjcao 176867 Jul 24 16:19 task.0.ckpt
-r-------- 1 hjcao hjcao 172771 Jul 24 16:19 task.1.ckpt
-r-------- 1 hjcao hjcao 172771 Jul 24 16:19 task.2.ckpt
-r-------- 1 hjcao hjcao 176867 Jul 24 16:19 task.3.ckpt
```

除了作业脚本进程以及脚本中加载的任务进程的映象文件外，系统还需要为作业保存一些元信息。这些信息保存在系统配置文件中 JobCheckpointDir 参数指定的目录下，名字为 *jobid*.ckpt。该目录下的文件不会被系统自动清除，因此需要管理员定期进行清理，删除不需要的作业信息，以免目录下积累太多的文件。

7.2.2 恢复执行

用户通过 yhcontrol 命令请求恢复批处理作业：

yhcontrol checkpoint restart *jobid*

其中 *jobid* 为要恢复的批处理作业的作业 ID。可以通过 ImageDir=*dir* 选项指定检查点映象文件的保存目录；若未指定则取作业最近一次检查点时的映象文件保存目录。可以通过 StickToNodes 选项指定在作业原来（被检查点时）运行的节点上恢复作业，否则所恢复作业的资源需求约束与原作业相同，并由系统根据调度进行分配。

恢复执行的作业 ID 与原作业 ID 相同，但是其提交时间为被恢复时间，以在记账数据中进行区分。所恢复的作业中如果有作业步，则其作业步 ID 将会被分配新的编号。

eg 批处理作业的恢复

恢复作业：

```
$ yhcontrol checkpoint restart 21952
```

恢复的作业：

```
$ yhqueue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  21952     work1   job.sh    hjcao   R       1:19      2 cn[534-535]
$ yhqueue -s
STEPID          NAME PARTITION     USER      TIME NODELIST
21952.1         prog     work1    hjcao      1:23 cn[534-535]
```

7.2.3 自动检查点与恢复

用户提交作业时可以通过 yhbatch 的 --checkpoint=*minutes* 选项指定对批处理作业进行周期性自动检查点，参数为两次检查点之间的时间间隔（分钟数）。自动检查点的批处理作业在由于节点故障运行失败时，系统会将其自动重新排队，等待资源分配后进行恢复续算。在作业失败和恢复运行之间系统将强制其等待一段时间（10 秒）。

eg 批处理作业自动周期性检查点

提交作业并运行：

```
$ yhbatch -N 2 -n 4 --checkpoint=30 job.sh
Submitted batch job 21945
$ yhqueue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  21945     work1   job.sh    hjcao   R       0:01      2 cn[540-541]
```

周期性生成检查点文件：

```
$ ls -l 21945/
drwxrwxr-x 2 hjcao hjcao    4096 Jul 24 16:10 21945.0
-r-------- 1 hjcao hjcao 1021461 Jul 24 16:10 script.ckpt
$ ls -l 21945/21945.0/
-r-------- 1 hjcao hjcao 172771 Jul 24 16:10 task.0.ckpt
-r-------- 1 hjcao hjcao 176867 Jul 24 16:10 task.1.ckpt
-r-------- 1 hjcao hjcao 172771 Jul 24 16:10 task.2.ckpt
-r-------- 1 hjcao hjcao 172771 Jul 24 16:10 task.3.ckpt
```

节点故障时作业重新排队，并在分配资源后恢复运行：

```
# yhcontrol update nodename=cn540 state=down
# yhqueue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  21945     work1   job.sh    hjcao  PD       0:00      2 (BeginTime)
...
$ yhqueue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  21945     work1   job.sh    hjcao   R       0:03      2 cn[541-542]
```

继续周期性生成检查点文件（恢复的作业中，作业步获得新编号）：

```
$ ls -l 21945/
drwxrwxr-x 2 hjcao hjcao    4096 Jul 24 16:11 21945.0
drwxrwxr-x 2 hjcao hjcao    4096 Jul 24 16:14 21945.1
-r-------- 1 hjcao hjcao 1021461 Jul 24 16:14 script.ckpt
$ ls -l 21945/21945.1/
-r-------- 1 hjcao hjcao 172771 Jul 24 16:14 task.0.ckpt
-r-------- 1 hjcao hjcao 176867 Jul 24 16:14 task.1.ckpt
-r-------- 1 hjcao hjcao 172771 Jul 24 16:14 task.2.ckpt
-r-------- 1 hjcao hjcao 172771 Jul 24 16:14 task.3.ckpt
```

7.3 MPI 程序的检查点

MPI 程序的检查点需要 MPI 运行环境的支持。不同版本的 YH-MPI、MPICH2、MVAPICH2、Open MPI 等有不同程度的检查点功能支持。根据 MPI 运行环境与资源管理系统结合进行进程管理的方法不同，资源管理系统对这些 MPI 程序的检查点支持也有所不同。

很多第三方应用软件使用 MPICH1，或从 MPICH1 派生的 MPI 版本，或其它不支持检查点的 MPI 版本进行编译构建。这些程序不支持检查点。具体程序的支持情况请咨询软件提供商。

7.3.1 YH-MPI

如果 MPI 编译生成的并行程序使用 `yhrun` 直接加载，如 YH-MPI，上述资源管理系统对检查点的支持完全适用。用户通过 `yhrun` 或 `yhrun_cr` 直接加载应用程序，并可以使用作业步与批处理作业的检查点/恢复功能。

7.3.2 MVAPICH2

对于 MVAPICH2 编译生成的程序，需要使用 `mpiexec_cr` 加载。用户可以使用其自带的检查点/恢复功能，即通过 BLCR 工具对 `mpiexec_cr` 进程进行检查点/恢复，`mpiexec_cr` 将与其进程管理服务 `MPD` 完成所加载任务检查点与恢复。但由于这样的程序加载不会生成作业步，因此资源管理系统无从管理其检查点映象文件。其生成的检查点映象文件名字与保存位置由 MVAPICH2 自行管理，详情请参见 MVAPICH2 的文档。

用户可以在批处理作业脚本中通过 `mpiexec_cr` 加载 MVAPICH2 编译生成的并行程序，并获得资源管理系统的批处理作业检查点/恢复支持。这种情况下，资源管理系统负责批处理作业脚本及其子进程的检查点/恢复，以及作业元信息的保存，程序的检查点/恢复仍由 MVAPICH2 负责。

7.3.3 Open MPI

对于 Open MPI 编译生成的程序，需要使用 `mpiexec` 加载。用户可以使用其自带的检查点/恢复功能，即通过 `ompi_checkpoint` 请求检查点，通过 `ompi_restart` 进行恢复。虽然 Open MPI 可以使用高性能计算机的资源管理系统的 `yhrun` 加载程序，并在资源管理系统看到作业步，但第 7.1节所述的检查点/恢复机制并不适用。

7.3.4 MPICH2

TODO。

第八章
触发器

资源管理系统支持触发器功能，可以在发生特定事件时执行指定的动作程序进行处理。

8.1 触发器事件

系统支持的触发器事件如下：

- 指定作业运行结束
- 指定作业到达运行时间限制
- 指定节点或分配给指定作业的节点状态变为 DOWN
- 指定节点或分配给指定作业的节点状态变为 DRAINED
- 指定节点或分配给指定作业的节点状态变为 FAILING
- 指定节点状态保持为 IDLE
- 指定节点从 DOWN 状态恢复
- 重读配置文件

根据设置时的请求参数，触发器可分为节点事件触发器、作业事件触发器和配置事件触发器，与之关联的实体分别为节点、作业和系统配置。

8.2 触发时机

触发器的动作程序并不是在发生事件时立即执行。当节点状态发生变化时，将设置一个标志。资源管理系统的控制进程周期性地扫描系统中发生的事件，并触发相应的程序执行。当前设置的扫描周期为 15 秒。因此，在同一扫描周期内多次发生的同一事件，将会被合并处理。

设置触发器时，可以指定执行动作程序与事件发生的时间偏移。例如，可以指定在作业到达运行时间限制前 15 分钟执行某程序。触发器被触发时，相应的事件标志将被清除。执行设置的程序后，触发器将会被从系统中清除。因此，如果希望对某一事件重复触发，则应在执行的动作程序中重新设置触发器。

触发器是一次性的。如果需要重复触发，则应在所执行的动作程序中重新设置触发器。但如果控制进程不是以 root 身份运行，即系统配置文件中 SlurmUser 参数值不是 root，则只有管理员才能设置触发器。

触发器的动作程序在控制进程运行的节点，即管理节点上运行。对于节点事件触发器，执行动作程序的参数为发生指定事件的节点列表；对于作业事件触发器，参数为作业的 JobID；对于配置时间触发器，执行动作程序的参数为字符串“reconfig”。

8.3 触发器管理

用户通过 yhtrigger 命令设置、查看和删除系统中的触发器。yhtrigger 的完整命令格式与选项请参见第??节。系统为每个触发器分配一个数值 ID，用于唯一标识。

触发器主要供管理员使用。尽管允许用户设置触发器，但触发器的处理脚本由 slurmctld 在管理节点上执行，而管理节点不挂载全局共享文件系统，所以对用户而言其用处受限。

8.3.1 设置触发器

设置触发器的命令为：

yhtrigger --set [options]

可通过选项指定：

- 相关实体：--node[=*nodes*] 指定节点；--jobid=*id* 指定作业；--reconfig 表示系统配置

- 事件类型：`--down` 表示节点 DOWN；`--fini` 表示作业结束等

- 动作程序：`--program=`*path*

- 触发时间偏移：`--offset=`*seconds*

当有节点状态变为 DOWN 时，向系统管理员发送通知邮件

命令：

```
yhtrigger --set --node --down --program=/usr/sbin/slurm_admin_notify
```

脚本 `slurm_admin_notify` 内容：

```
#!/bin/bash
# Submit trigger for next event
yhtrigger --set --node --down --program=/usr/sbin/slurm_admin_notify
# Notify administrator by e-mail
/bin/mail slurm_admin@site.com -s NodesDown:$*
```

有节点空闲超过 600 秒时执行程序 `slurm_suspend_node`

命令：

```
yhtrigger --set --node --idle --offset=600 --program=/usr/sbin/slurm_suspend_node
```

作业 1234 到达运行时间限制 10 分钟前执行程序 `/home/joe/clean_up`

命令：

```
yhtrigger --set --jobid=1234 --time --offset=-600 --program=/home/joe/clean_up
```

eg 分配给作业 1234 的节点进入 DOWN 状态时执行程序 /home/joe/node_died

命令：

```
yhtrigger --set --jobid=1234 --down --program=/home/joe/node_died
```

8.3.2 查看触发器

查看触发器的命令为：

yhtrigger --get [options]

可通过选项指定相关实体、事件类型、触发器 ID、触发器所属用户等。

eg 查看与作业 1234 相关联的触发器

命令：

```
yhtrigger --get --jobid=1234
TRIG_ID RES_TYPE RES_ID TYPE OFFSET USER PROGRAM
    123      job   1234 time   -600  joe /home/bob/clean_up
    125      job   1234 down      0  joe /home/bob/node_died
```

8.3.3 取消触发器

取消触发器的命令为：

yhtrigger --clear [options]

可通过选项指定相关联的作业 JobID、触发器 ID、触发器所属用户等。

eg 取消触发器 123

命令：

```
yhtrigger --clear --id=123
```

第三部分

系统管理

第九章
系统设置

资源管理系统通过一组配置文件进行配置，包括系统配置文件、节点配置文件、分区配置文件、调度配置文件、记账存储配置文件、拓扑配置文件、插件配置文件等。配置文件位于 /etc/slurm 目录下。

资源管理系统的运行还需要一些系统服务与支撑环境的正确设置。

9.1 资源管理系统配置

资源管理系统的配置文件均为文本文件，按行进行组织。每行中以“#”开始直到行尾为注释。详细的配置参数请参考第??章。

9.1.1 系统配置

系统配置文件中可以包含系统参数设置、节点定义、分区定义等。为便于维护，高性能计算机中将系统配置文件通过文件包含机制分成了几个文件，即在 slurm.conf 中通过“Include”关键词引用了其余配置文件：

- slurm.conf：系统配置参数。参数设置行的格式为“*param*=*value*”。
- sched.conf：调度相关参数。格式同上。
- node.conf：节点定义。节点定义行的格式为“NodeName=*nodelist Attribute*=*value ...*”。
- partition.conf：分区定义。分区定义行的格式为“PartitionName=*partition Attribute*=*value ...*”。

资源管理系统的所有组件，包括控制进程 slurmctld、节点监控进程 slurmd、命令行工具等等，在运行时都需要读取系统配置文件，以获得系统的配置信息。

9.1.2 记账存储配置

记账存储配置文件名字为 slurmdbd.conf，其中包含了记账数据库的信息等参数，格式同系统配置文件。记账存储文件仅被运行在管理节点上的记账存储进程读取。

9.1.3 拓扑配置

拓扑配置文件名字为 topology.conf，其中包含了系统互联的拓扑。此文件由控制进程的拓扑插件读取，用于优化作业调度时的资源分配。一般而言，系统互联拓扑在系统构建时确定，管理员无需修改。

9.1.4 修改配置文件

在修改系统配置文件后，可以通过“yhcontrol reconfigure”命令指示系统守护进程重新读取配置文件。但某些参数的修改，如增加或减少系统中的节点，不能通过此方式生效，而必须重新启动系统守护进程。

所有守护进程及命令工具仅读取其执行时所在节点的配置文件。因此管理员需要保证所有节点上配置文件的一致性。

9.2 高可用支持

资源管理系统支持双管理节点的备份。由于系统的正常运行需要一些支撑服务，因此，管理节点发生故障时可能需要管理员进行干预。

9.2.1 服务部署

当系统配备两个管理节点时，mn0 为主管理节点，mn1 为备份管理节点。系统中的服务部署如下：

- mn0 运行资源管理系统主控 slurmctld 进程，mn1 运行资源管理系统备份 slurmctld 进程。
- mn0 运行资源管理系统主记账存储进程 slurmdbd，mn1 运行备份记账存储进程 slurmdbd。
- mn1 运行记账数据库 mysql。
- 系统中所有节点都运行 NTP 服务。mn0 的 NTP 服务从本机时钟获取参考时间，mn1 的 NTP 服务从 mn0 获取参考时间，其他所有节点从 mn0 和 mn1 获取参考时间。

- 如果使用基于 LDAP 的用户管理，则 mn0 运行主 LDAP 服务，mn1 运行从 LDAP 服务。

系统平时主要由 mn0 提供服务，mn1 处于 standby 或轻载状态。当 mn0 故障时，mn1 可以接管系统提供服务。

TODO：单点故障是不可避免的，因为系统运行需要一致的状态文件和记账数据。

9.2.2 故障与恢复流程

9.3 支撑环境

9.3.1 用户管理

需要所有的节点有一致的用户。

记账数据库中存储的是用户名。

9.3.2 时间同步

高性能计算机的正常运行需要所有节点上的时间保持一致。

高性能计算机系统一般使用 NTP 保持各节点之间的时间同步。在所有节点上运行 NTP 服务，并将管理节点设置为时间服务器。其中，mn0 的优先级最高，mn1 次之，依此类推。

管理员修改系统时间应该 mn0 上进行，NTP 服务将自动使其它节点与之保持同步。

管理员应定期检查系统时间，并确保系统时间正确。否则可能导致系统记账数据的偏差，并有可能不能加载计算任务。

9.3.3 节点命名与地址解析

9.3.4 节点主机名

系统中所有节点的主机名（hostname）都设置为内部名字。为防止在用户单位有多台高性能计算机的情况下，用户和管理员登录时造成混淆，在管理员确定系统名称后，修改登录节点、管理节点和 I/O 节点的登录 Shell 配置，使之显示系统名字。

系统中节点的命名如下：

- 登录节点：ln0，ln1，...
- 管理节点：mn0，mn1，...
- 计算节点：cn0，cn1，...
- I/O 节点
  - 元数据服务节点：mds0，mds1，...
  - 对象存储节点：ost0，ost1，...

9.3.5 节点地址

高性能计算机中节点上可能使用三种类型的网络：以太网、Infiniband 网络、GLEX 网络。

对系统中互联互通的同一类型网络，各节点的网络接口地址配置在同一 B 类子网中；不互通的网络接口地址配置在不同的 B 类子网中。登录节点和管理节点一般通过以太网连接到用户单位的内部网络，其相应接口的地址由管理员确定。

不同类型网络使用的子网如下；如果节点为无盘启动，需要使用 DHCP 分配临时地址，则使用 4.8.0.0/16 子网范围内的地址。

- 以太网：25.8.0.0/16，25.9.0.0/16，...
- Infiniband 网络：89.72.0.0/16，89.73.0.0/16，...
- GLEX 网络：121.104.0.0/16，121.105.0.0/16，...

管理员仅需要设置连接到用户单位网络的以太网接口地址。请不要改动系统中节点的主机名和内部网络接口地址，否则可能导致系统不可用。

9.4 数据备份

管理员需要定期备份管理性数据，以避免因软硬件故障导致的数据丢失。

- 系统配置文件

  每次修改配置文件后，应进行备份。

- 记账数据

  应备份记账数据，以查看历史作业信息。

- 管理数据即帐号、用户、QOS 等。

第十章
系统控制

管理员可以通过 yhcontrol 命令，对系统中的分区、节点、作业、配置等进行控制。

10.1 分区控制

管理员可以通过 yhcontrol 命令创建、修改、删除系统中的分区。

10.1.1 创建分区

命令格式为：

yhcontrol create PartitionName=*name* [*attr*=*val* ...]

要创建的分区名字不能与系统已有分区名字重叠。分区的所有属性，包括分区状态、分区中包含的节点，都是可选的。详细的分区属性请参考第 2.1.3节，命令格式请参考第??节。

eg 创建分区

```
# yhinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
work*        up   infinite    176  alloc cn[0-175]
# yhcontrol create partitionname=test nodes=cn[0-15]
# yhinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
work*        up   infinite    176  alloc cn[0-175]
test         up   infinite     16  alloc cn[0-15]
```

10.1.2 修改分区

命令格式为：

yhcontrol update PartitionName=*name* [*attr*=*val* ...]

要修改的分区必须已经存在。分区的所有属性，包括分区状态、分区中包含的节点，都是可选的。详细的分区属性请参考第 2.1.3节，命令格式请参考第??节。

eg 修改分区

```
# yhcontrol update partitionname=test nodes=cn[0-31] state=down
# yhinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
work*        up   infinite    176  alloc cn[0-175]
test       down   infinite     32  alloc cn[0-31]
```

10.1.3 删除分区

命令格式为：

yhcontrol delete PartitionName=*name*

删除分区将导致分区中的排队、运行和挂起的作业被终止，其状态标记为 FAIL，原因为“PartitionDown”。

eg 删除分区

```
# yhcontrol delete partitionname=test
# yhinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
work*        up   infinite    176  alloc cn[0-175]
```

控制进程将会把分区的状态和属性等信息保存到磁盘文件中。当重启控制进程或重读配置文件时，可以从磁盘文件恢复分区及其属性信息。但是，如果磁盘文件丢失，或指定不恢复状态，则系统将仅根据配置文件的内容重新构建分区。因此，若需作出持久改变，请修改系统配置文件。配置文件中的分区信息仅对控制进程有效。

10.2 节点控制

管理员可以通过 yhcontrol 命令修改系统中节点的某些属性。命令格式为：

yhcontrol update NodeName=*name* [*attr*=*val* ...]

可以修改的节点属性包括：

- 节点特性：Features=*string*
- 节点调度权重：Weight=*number*
- 节点状态：State=*state*
- 节点状态原因：Reason=*string*

具体的命令格式请参见第??节。

若需要在系统中增加或减少节点，则必须修改配置文件，并重新启动系统。

10.2.1 修改节点状态

管理员有时候需要手工修改节点的状态。例如，将重启的节点的状态置为 IDLE，或为节点设置 DRAIN 标志等。通过 yhcontrol 命令可以设置的节点状态值包括：

- NoResp：设置 NO_RESPOND 标志。
- DRAIN：设置 DRAIN 标志。
- FAIL：设置 FAIL 标志。
- RESUME：继续节点的正常状态监测。具体为：清除节点的 DRAIN 和 FAIL 标志；若节点为 DOWN 状态，将其设置为 IDLE 状态，但为其设置 NO_RESPOND 标志。

- POWER_DOWN：设置节点的 POWER_SAVE 标志。

- POWER_UP：设置节点的 POWER_UP 标志。

- DOWN：设置节点为 DOWN 状态。

- IDLE：设置节点为 IDLE 状态。节点的 DRAIN 和 FAIL 标志将被清除。

- ALLOCATED：设置节点为 ALLOCATED 状态。

并不是所有的节点状态转换都是有效的：

- 新状态值 NoResp、DRAIN、FAIL、POWER_DOWN、POWER_UP、DOWN 总是有效。

- 新状态值 RESUME 仅在节点原状态为 DOWN，或节点状态不为 UNKNOWN 且节点具有 DRAIN 或 FAIL 标志时有效。

- 新状态值 IDLE 仅在节点原状态为 DOWN 或 IDLE 时有效。

- 新状态值 ALLOCATED 仅在节点原状态为 ALLOCATED 时有效。

为节点设置 DRAIN 标志或 FAIL 标志时，必须指定原因。将节点设置为 DOWN 状态将导致在节点上运行的作业被终止（除非作业设置了容错选项），状态为 NODE_FAIL，原因为“NodeDown”。

对节点属性的修改将被控制进程保存到磁盘文件中，并在重启和重读配置文件时从中读取。对于节点状态，仅 DOWN 状态、DRAIN/FAIL/POWER_SAVE/POWER_UP 标志被恢复。

eg 修改节点状态

为节点设置 DRAIN 标志，以暂停将节点分配到作业：

```
# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
work*        up   infinite    176  alloc cn[0-175]
# yhcontrol update nodename=cn123 state=drain reason="system maintain"
# yhinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
work*        up   infinite      1   drng cn123
work*        up   infinite    175  alloc cn[0-122,124-175]
# yhcontrol update nodename=cn123 state=resume
# yhinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
work*        up   infinite    176  alloc cn[0-175]
```

! yhinfo 输出中处于 comp 状态的节点是具有 COMPLETING 标记的节点。这表示相应节点上有作业的进程正处于退出过程中，系统将定期请求相应进程终止。这是系统确保作业结束时不在节点上残留进程的一种机制，不能通过修改节点状态清除其 COMPLETING 标志。有 comp 的节点就有对应的 CG 的作业，可通过“yhcontrol completing”命令查看 CG 的作业及其关联的节点。

10.3 作业控制

管理员可以进行一些普通用户所无权进行的作业控制操作。

10.3.1 挂起与恢复

将正在运行的作业挂起，可以暂时释放出节点上的 CPU 资源。挂起和恢复作业通过 yhcontrol 命令进行。

挂起作业时，分配给作业的节点将被暂时释放，从而这些节点（上的资源）可以分配给其它作业。当恢复作业时，这些节点被重新分配给作业。

被挂起时，作业的所有作业步的任务将被发送 SIGTSTP 信号，1 秒钟之后再发送

SIGSTOP 信号。SIGTSTP 信号为任务进程提供了处理挂起事件的机会，某些 MPI 库的实现可能需要此机制。恢复作业时，其所有任务被发送 SIGCONT 信号。

作业的挂起与恢复

```
# yhinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
work*        up   infinite     16  alloc cn[0-15]
work*        up   infinite   1008   idle cn[16-1023]
# yhqueue
JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  634      work   job.sh  test605   R       2:40     16 cn[0-15]
# yhcontrol suspend 634
# yhinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
work*        up   infinite   1024   idle cn[0-1023]
# yhqueue
JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  634      work   job.sh  test605   S       2:50     16 cn[0-15]
# yhcontrol resume 634
# yhinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
work*        up   infinite     16  alloc cn[0-15]
work*        up   infinite   1008   idle cn[16-1023]
# yhqueue
JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
  634      work   job.sh  test605   R       2:56     16 cn[0-15]
```

普通用户不能挂起和恢复作业。因为挂起作业只能释放出节点上的 CPU 资源，不能释放内存等资源。但挂起作业会使得可以将其释放的资源分配给其它作业使用。因此当有作业分配到相同的节点上时，会出现几个作业竞争使用资源的情况。

10.3.2 消息发送

管理员可以向作业发送消息：

yhcontrol notify *jobid msg*

对于交互作业，此消息将被发送到 yhrun 进程，并由 yhrun 在其标准错误上输出。

对于分配模式作业，此消息将被发送到 yhalloc 进程，并由 yhalloc 在其标准错误上输出。

对于批处理作业，此消息将被传递到作业所分配的第一个节点上，进而发送给管理作业脚本运行的 slurmstepd 进程，并由其在作业的标准错误文件中输出。

向作业发送消息

```
# yhcontrol notify 164071 hi, this is a test message
# cat slurm-164071.out
slurmd[cn176]: hi, this is a test message
```

10.4 配置控制

“yhcontrol reconfigure”命令指示资源管理系统控制进程重读配置文件。同时，控制进程会将此请求传递到计算节点上的节点监控进程，请求其重读配置文件。

“yhcontrol show config”命令可以查看系统的配置信息，这些配置信息从控制进程获取，而不是从执行 yhcontrol 命令的节点读取。

查看系统配置

```
# yhcontrol show config
AccountingStorageBackupHost = (null)
AccountingStorageEnforce = associations,limits,qos,wckeys
AccountingStorageHost   = mn0
...
```

10.5 系统控制

10.5.1 系统启动

资源管理系统的守护进程运行在管理节点和计算节点上。一般高性能计算机上会配置在系统启动时自动启动资源管理系统的守护进程。如果有需要，管理员可以手工启动或重启相关进程。

一般地，资源管理系统的守护进程通过系统启动脚本启动或关闭。在计算节点上，/etc/init.d/slurm 用来启动或关闭 slurmd 进程；在管理节点上，/etc/init.d/slurm 用来启动或关闭 slurmctld 进程；/etc/init.d/slurmdbd 用来启动或关闭 slurmdbd 进程。

重新启动系统控制进程

```
[root@mn0 ~]# /etc/init.d/slurm restart
stopping slurmctld:                                        [  OK  ]
slurmctld (pid 14753) is running...
slurmctld (pid 14753) is running...
slurmctld is stopped
starting slurmctld:                                        [  OK  ]
```

如需直接启动相关进程，也可以直接执行相应进程的可执行程序，这一般用于系统调试。“-D”选项可以使进程不要进入后台；“-v”选项（可加多个）可以增加进程的调试信息详细级别。

eg 前台带调试信息启动 slurmd 进程

```
[root@cn127 ~]# slurmd -D -vvvvvvv
slurmd: debug3: Trying to load plugin /usr/lib64/slurm/topology_none.so
slurmd: topology NONE plugin loaded
slurmd: debug3: Success.
...
slurmd: slurmd version 2.2.0-pre7 started
slurmd: debug3: finished daemonize
slurmd: killing old slurmd[6372]
slurmd: debug3: cred_unpack: job 21927 ctime:100724152919 revoked:100724153656 expires:10
slurmd: debug3: cred_unpack: job 21931 ctime:100724153904 revoked:100724154159 expires:10
slurmd: debug3: cred_unpack: job 21932 ctime:100724154209 revoked:100724154226 expires:10
slurmd: debug3: Trying to load plugin /usr/lib64/slurm/switch_none.so
slurmd: switch NONE plugin loaded
slurmd: debug3: Success.
slurmd: debug3: successfully opened slurm listen port *:6818
slurmd: slurmd started on Sat 24 Jul 2010 15:42:41 +0800
slurmd: Procs=12 Sockets=2 Cores=6 Threads=1 Memory=48303 TmpDisk=1487 Uptime=62314
```

对于前台启动的资源管理系统进程，按“Ctrl+C”可以使之退出。

10.5.2 关闭系统

“yhcontrol shutdown”命令指示控制进程关闭退出；在退出前控制进程将此请求传送到备份控制进程和计算节点上的节点监控进程。

“yhcontrol abort”命令指示控制进程立即退出并生成 core 文件。这可用于检查控制进程的内部数据结构状态。

10.5.3 接管系统

默认地，备份控制进程定期 ping 主控制进程，以检查其可用状态。如果在一段时间（系统配置文件的 SlurmctldTimeout 参数值）内主控制进程持续无响应，则备份控制进程将认为主控制进程失效，并接管系统。这是为了避免暂时性的网络故障等情况下的系统抖动。

如果主控制进程失效，则在备份控制进程接管之前，资源管理系统的各种命令将无法

使用。如果管理员确认主控制进程失效，则可以通过“yhcontrol takeover”命令通知备份控制进程立即接管。

10.5.4 Ping 控制进程

“yhcontrol ping”命令用于查看系统控制进程的运行状态。yhcontrol 将分别向主控制进程和备份控制进程发送 ping 消息，并等待其响应。

Ping 控制进程

```
# yhcontrol ping
Slurmctld(primary/backup) at mn0/mn1 are UP/DOWN
****************************************
** RESTORE SLURMCTLD DAEMON TO SERVICE **
****************************************
```

10.5.5 调整调试日志详细级别

管理员可以动态调整控制进程调试日志信息的详细级别。控制进程启动时的默认系统调试日志详细级别由系统配置文件中的 SlurmctldDebug 参数设置，默认的调度日志详细级别由系统配置文件中的 SlurmSchedLogLevel 参数设置。如下命令可设置系统调试日志信息的详细级别：

yhcontrol setdebug *level*

或

scontrol update SlurmctldDebug=*number*

其中 *level* 可以是 0–9 之间的数值，也可以是其相对应的如下名字值；*number* 只能是数值；级别数值越高，调试日志信息越详细：

- quiet
- error
- verbose
- debug2
- debug4
- fatal
- info
- debug
- debug3
- debug5

如下命令可设置调度日志信息的详细级别：

yhcontrol schedloglevel *level*

其中 *level* 可以是 0 或 1 的数值，或者 enable 或 disable。

较高调试日志详细级别将导致控制进程输出大量的调试信息。这可能造成很大的日志文件，以及不易查找定位。因此管理员可以关闭默认的调试日志，并在需要定位故障时打开。定位完毕后应记得关闭（设置级别为 0）。

第十一章
许可证管理

资源管理系统支持简单的许可证管理模式。管理员在系统配置文件中定义所拥有的许可证及数量（参见第??章）。用户在提交作业时，指定作业所需要的许可证及数量（参见第 4章）。在进行作业调度时，为作业分配资源之前将先检查是否有足够的许可证。若许可证资源不够，则作业不能运行，将保持在排队状态，原因为“Licenses”。

许可证相当于一种全局可用的、有数量限制的资源。控制进程对系统中可用的许可证进行简单的计数。当将许可证分配给作业时，减少相应的可用许可证数目；作业运行结束时，增加相应的可用许可证数目。由于不与实际的许可证管理服务进行交互，因此这仅适用于系统中所有对许可证的使用为从资源管理系统分配资源的作业的情况。若有程序不经分配资源即直接使用许可证，则可能实际可用许可证少于资源管理系统记录的可用许可证数目，从而出现作业被分配资源后由于许可证不可用而不能正常运行的情况。

系统支持对许可证的预约，详情请参见第 13章。

第十二章
作业调度

作业调度是指系统根据队列中作业的属性和系统的资源可用情况，选择合适的作业并为其分配合适的资源。它主要包括两个方面：选择哪个作业和为作业分配哪些资源。

12.1 概述

12.1.1 调度时机

资源管理系统周期性地，并在发生特定事件时，根据系统配置、作业的优先级、可用资源等情况，选择合适的作业，为其分配资源运行。作业调度的周期为 60 秒；其它调度时机包括：

- 重读配置文件
- 有作业完全结束，其所有节点被释放（COMPLETING 标志被清除）
- 提交批处理作业
- 修改作业参数
- 挂起作业
- 修改节点状态
- 修改分区配置
- 删除分区
- 创建、修改、删除预约

此外，用户提交作业时，若作业在分区中的优先级最高，则检查其是否可以立即运行。

如果系统配置参数 CompleteWait 非 0，则当有作业具有 COMPLETING 标志（CG 状态）且其结束的时间未超过该选项设置的时间秒数，系统将不进行调度，以避免资源被过度碎片化。

12.1.2 调度流程

控制进程把系统中所有的作业维护在一个列表中。在作业调度时，系统首先选择可以运行的作业。可以运行的作业需要满足启动时间、依赖关系、预约可用、未被阻止等条件。然后对可运行作业进行排序：先按分区优先级由高到低排序，同一分区内再按作业的优先级由高到低排序。

系统然后按作业排序顺序进行扫描，依次为作业分配资源，直到有作业因资源不够而不能运行。这时，该作业所在分区中的其它作业（优先级低于当前作业）也将不再被调度运行；且不再将该分区中的节点分配到其它作业，以保留给高优先级分区。系统继续扫描其余分区的作业。

在逻辑上，每个分区中的作业相当于在一个单独的队列中排队，而分区的优先级就是队列的优先级。

12.2 优先级排队

高性能计算机使用基于综合优先级的作业排队机制。作业的优先级为 32 位无符号整数，由作业属性与管理员配置的权重综合计算得出：

$$
\begin{aligned}
\text{作业优先级} = & \ PriorityWeightAge \times \text{作业年龄因子} \\
& + PriorityWeightFairshare \times \text{公平份额因子} \\
& + PriorityWeightJobSize \times \text{作业大小因子} \\
& + PriorityWeightPartition \times \text{分区因子} \\
& + PriorityWeightQOS \times QOS \text{ 因子} \\
& + \text{作业的优先级偏移}
\end{aligned}
$$

上式中，各优先级权重为管理员可在系统配置文件中设置的权重值，为 32 位无符号整数；各因子根据作业属性计算得到，值介于 0.0 和 1.0 之间；优先级偏移在提交作业或修改作业时指定，值为介于 -10000 和 10000 之间的整数。如果用户直接设置了作业的优先级，则系统使用用户指定的值，而不再为其进行计算。

- 作业的年龄指从可以运行算起，作业在队列中等待的时间。作业因依赖关系、启动时间限制等不能运行时在队列中等待的时间不计算在内。年龄因子为作业的年龄与系统配置的最大作业年龄（系统配置文件中 PriorityMaxAge 选项）的比值，并被舍入到 0.0 和 1.0 之间。
- 公平份额因子根据作业的关联（参见第 14章）的公平份额，及其实际资源使用情况计算得来。详情请参见第 12.4节。
- 作业大小因子是作业请求的最少节点数目与系统中的总节点数之间的比值；但是如果系统配置文件中设置了选项 PriorityFavorSmall，则为 1 减去该比值。
- 分区因子为作业所在分区的归一化优先级，即分区优先级与系统中优先级最高的分区的优先级之间的比值。分区的优先级由管理员在系统配置文件中设置。
- QOS 因子为作业所使用的 QOS 的归一化优先级，即 QOS 的优先级与系统中优先级最高的 QOS 的优先级之间的比值。QOS 的优先级由管理员在定义 QOS 时设置。

通过定义不同的优先级权重，管理员可以定义出不同的调度策略。

定制调度策略

先来先服务（FCFS），即排队时间越长，优先级越高：

```
PriorityWeightAge=1000
```

小作业优先：

```
PriorityWeightJobSize=1000
PriorityFavorSmall=1
```

按 QOS 确定优先级：

```
PriorityWeightQOS=1000
```

12.3 回填

回填（backfill）是指在不影响高优先级作业的最早启动时间的条件下，将低优先级的作业提前运行。

回填的重要条件是作业的运行时间限制。回填使得用户要以二维方式理解资源的可用性，而不仅仅是看当前可用的资源数量。因此，用户提交作业时提供的作业运行时间估

计对作业能否被回填具有重要影响。用户应在保证安全的情况下尽可能准确地估计作业运行时间，而不应随意给定一个很长的时间值。

eg 作业回填

设系统共有 128 个节点。下面的队列中，假设作业 1463 的预计运行时间为 2 小时。则按当前负载，预计作业 1464 将在 1 小时 58 分后可以开始运行。因此，如果作业 1465 和 1466 的运行时间不超过 1 小时 58 分，则它们可以被提前运行。若其运行时间超过 1 小时 58 分，则不能将其提前运行，否则作业 1464 的启动时间可能被延迟。

```
$ yhqueue
JOBID PARTITION   NAME USER ST TIME NODES NODELIST(REASON)
 1463      work sbatch root  R 2:00    64 cn[0-63]
 1464      work  myjob root PD 0:00   128 (Resource)
 1465      work   tjob test PD 0:00    32 (Priority)
 1466      work  a.out test PD 0:00    32 (Priority)
```

系统周期性地按优先级顺序（排序与作业调度时相同）扫描作业队列，寻找回填的机会。回填扫描周期通过系统配置文件的 SchedulerParameters 参数定义，格式为“`interval=`*seconds*”，默认为 10 秒。

对于高优先级的作业，如果当前资源数量不能满足其需求，则系统将计算其最早可能的启动时间，并暂时地（仅在回填过程中有效）为其预留资源，以保证其最早启动时间不受低优先级的被回填的作业的影响。但是如果作业的最早启动时间在一天之后，则不为其预留资源。这是考虑到用户估计的作业运行时间精度有限，为了增加回填的机会，以提高系统利用率。回填时最多检查的作业数通过系统配置文件的 SchedulerParameters 参数定义，格式为“`max_job_bf=`*cnt*”，默认为 50 个。

当有作业具有 COMPLETING 状态，且其结束时间小于系统配置文件中的 CompleteWait 参数值，则不进行回填。即使 CompleteWait 为 0，系统也将等待 5 秒中之后再进行 backfill。这是为了避免系统资源被过度碎片化。

12.4 公平份额

管理员可以设置各个计费帐号与用户对系统中的资源使用所占的份额。资源管理系统在进行调度时，将通过调节相关作业的优先级，使得各计费帐号与用户对资源实际使用的比例接近设置的份额。系统根据作业的关联来计算其计算优先级时的公平份额因子：

$$\text{公平份额因子} = (\text{关联设置的份额} - \text{关联实际资源使用份额} + 1)/2.0$$

在计算中，设置的关联份额和实际使用的份额均为归一化的。参见第 12.2节可知，随着关联的实际资源使用份额增加，公平份额因子会逐渐降低，相关作业的优先级也就越低。而随着关联实际资源使用份额的降低，公平份额因子逐渐增大，相关作业的优先级会被增加，最多为 *PriorityWeightFairshare*。

关联的份额由管理员在定义帐号和用户时设置，参见第 14章。

各个关联的实际资源使用份额以系统配置文件中参数 PriorityCalcPeriod 的值为周期进行计算。在计算时，考虑关联在本周期以及在历史周期中的资源使用情况，给当前周期的使用情况以较大的权重，而历史使用情况随着时间的流逝权重逐渐降低。具体的计算公式如下：

$$\begin{aligned} R_i &= NumProcs \times PriorityCalcPeriod \\ U_H &= (D^0 \times U_0) + (D^1 \times U_1) + (D^2 \times U_2) + \cdots \\ R_H &= (D^0 \times R_0) + (D^1 \times R_1) + (D^2 \times R_2) + \cdots \\ &= NumProcs \times PriorityDecayHalfLife \times 2 \\ U_N &= \frac{U_H}{R_H} \end{aligned}$$

上式中，R_i 为第 i 个计算周期内系统可用的资源量（处理器秒）；U_i 为第 i 个周期内关联使用的资源量（处理器秒）；R_H 为系统历史综合可用资源量；U_H 为关联的历史综合资源使用量；U_N 为关联的归一化资源使用份额；D 为衰减因子；$NumProcs$ 为系统中的处理器数目。

衰减因子由管理员通过系统配置文件中的参数 PriorityDecayHalfLife 确定：它的含义是，若一个关联在当前周期内没有新的资源使用，则其资源使用量衰减为上一周期的一半；即 PriorityDecayHalfLife 为资源使用的半衰期。

系统配置文件中参数 PriorityUsageResetPeriod 设置重置实际资源使用份额的周期。重置时，各关联和 QOS 的资源使用份额被清零，并在下个计算周期重新进行计算。该参数的可用值为：

- `NONE`：从不重新计算
- `NOW`：每次控制进程启动后重新计算一次
- `DAILY`：每天重新计算
- `WEEKLY`：每周重新计算

- MONTHLY：每月重新计算
- QUARTERLY：每季度重新计算
- YEARLY：每年重新计算

公平份额只是一种作业优先级调节机制，并不强制也不能保证各帐号和用户的最终实际资源使用比例与设置的份额值一致。

12.5 作业抢占

系统支持作业抢占。当作业优先级足够高，且满足其它限制条件时，系统将尝试为作业选择节点进行分配。如果管理员设置了作业抢占，且当前可用节点不能满足作业的资源需求，则系统在选择节点时将考虑抢占合适的作业，以释放出资源，供当前作业使用。

12.5.1 抢占方式

作业抢占方式决定一个作业可以抢占哪些作业，由系统配置文件中的 PreemptType 参数设置，可选值为：

- preempt/none：不进行作业抢占。
- preempt/qos：基于 QOS 的作业抢占

  管理员在定义 QOS 时，指定其可抢占的 QOS，参见第 15章。抢占是 QOS 间的一种传递关系。

- preempt/partition_prio：基于分区优先级的作业抢占

  高优先级分区中的作业可抢占低优先级分区中的作业。

系统根据抢占方式为作业寻找可被抢占的作业。在抢占时，系统为可被抢占作业计算一个“被抢占优先级”，并根据其顺序对被抢占作业进行排序。对于基于 QOS 的抢占，作业的被抢占优先级首先按 QOS 的优先级由低到高排序，然后按作业分配的节点数由大到小排序；对于基于分区优先级的抢占，作业的被抢占优先级首先按其分区的优先级由低到高排序，然后按作业分配的节点数由大到小排序。

只有抢占作业能够释放出足够多的资源时，作业才真正被抢占。

12.5.2 抢占模式

系统支持几种不同的抢占模式，即作业在被抢占时的处理方式，由系统配置文件中的 PreemptMode 参数设置，可选值为：

- `OFF`：不支持抢占，抢占方式只能设置为 `preempt/none`
- `CANCEL`：取消被抢占的作业
- `CHECKPOINT`：对被取消作业进行检查点，然后取消之
- `REQUEUE`：将被抢占作业重排队

12.6 节点选择

12.6.1 节点权重

系统配置文件中定义节点时，每个节点有一个调度权重 Weight（参见第**??**节）。在同等条件下，此权重值越小，节点越优先被分配给作业。

12.6.2 消耗性资源分配

对节点上的 CPU 和内存资源，资源管理系统支持消耗性分配方法。例如，若 1 个节点上有 8 个处理器，某个作业分配了 3 个，则其它作业还可以分配剩余的 5 个。类似地，若 1 个节点上有 32G 内存，一个作业分配了 16G，则其它作业还可分配剩余的 16G。

系统配置文件中的 SelectTypeParams 参数设置哪些资源作为消耗性资源。一般地，该参数值设置为 CR_CPU_Memory。由于高性能计算机一般不使用超线程，这表示按处理器核和内存进行消耗性资源分配。

12.6.3 拓扑感知

高性能计算机系统可能使用多种互联拓扑。对于作业调度和资源选择而言，这些互联可以简化为树拓扑。系统拓扑结构由开发人员在拓扑配置文件中定义。出于系统作业调度效率考虑，拓扑定义一般在符合系统实际的情况下进行简化。

系统在为作业选择节点时，将考虑系统互联拓扑，使得分配给作业的节点尽可能通过较少的交换机跳数联接。

第十三章
资源预约

13.1 概述

资源管理系统支持资源预约功能。管理员可以为特定的用户或计费帐号创建预约，以保证其运行作业时资源可用。

预约包含三个要素：

- 起止时间：在该时间段内，预约有效。
- 预约的资源：即预约哪些计算节点。
- 访问控制：允许哪些用户和/或计费账号的作业在预约中运行。

常见的预约使用模式包括：为用户预留资源；创建带有 MAINT 标志的预约进行系统维护；创建周期性预约为特定用户在特定时间预留资源；创建无时间限制的预约以对特定节点进行故障诊断等等。

管理员通过 `yhcontrol` 命令对预约进行操作，具体命令参数请参见第??节。

13.2 预约控制

13.2.1 创建预约

创建预约的命令为：

`yhcontrol create reservation` *param*=*val* ...

可以指定的参数包括：

- 预约的名字：`ReservationName=`*name*

  若未指定，则系统自动为预约生成一个名字。

- 要预留的许可证：Licenses=*license*

  系统在创建预约时并不检查是否有足够的许可证，而是在作业调度时保证，被预留的许可证不会分给不相关（不再预约中的）作业使用。

- 要预留的节点数目：NodeCnt=*number*

  系统将自动为预约选择满足约束的节点。

- 要预留的节点：Nodes=*nodelist*

  必须指定 NodeCnt 或 Nodes 之一。

- 预约的开始时间：StartTime=*timespec*

  必须指定。

- 预约的结束时间：EndTime=*timespec*

- 预约的持续时间：Duration=*time*

  必须指定 EndTime 或 Duration 之一。

- 预约所在的分区：PartitionName=*partition*

  和作业类似，预约不能跨分区。

- 预留节点的特性：Features=*features*

  只有满足特性约束的节点才会被分配给预约。参见第 4.2.2节。

- 可使用预约的用户：Users=*userlist*

- 可使用预约的帐号：Accounts=*accountlist*

  必须指定 Users 或 Accounts，两者可同时指定。

- 预约的标志：Flags=*flags*

  可用的标志有：

  - MAINT：系统维护。在记账时此预约将被特殊处理。系统维护预约可以与其它预约重叠。
  - OVERLAP：此预约可以与其它预约重叠。默认情况下，预约中的节点不会与其它预约中的节点重叠。

- IGNORE_JOBS：在创建预约时忽略当前运行的作业。默认情况下，在为预约分配节点时，要考虑当前运行的作业，即运行的作业在预约开始时应该已经运行结束（使用作业的运行时间限制计算）。此选项可用于创建包含全部节点的系统维护预约。

- DAILY：每天在相同时间重复预约。

- WEEKLY：每周在相同时间重复预约。

系统在预约创建时即为其分配节点。创建成功后，将返回预约的名字。

创建预约，在 2 月 6 日下午 16:00 进行系统维护，持续 2 个小时

```
$ yhcontrol create reservation starttime=2009-02-06T16:00:00 duration=120
user=root flags=maint,ignore_jobs nodes=ALL
Reservation created: root_3
$ yhcontrol show reservation
ReservationName=root_3 StartTime=2009-02-06T16:00:00
EndTime=2009-02-06T18:00:00 Duration=120
Nodes=ALL NodeCnt=20
Features=(null) PartitionName=(null)
Flags=MAINT,SPEC_NODES,IGNORE_JOBS Licenses=(null)
Users=root Accounts=(null)
```

在上例中，“Nodes=ALL”表示预约系统中的所有节点；“MAINT”标志表明预约的目的是进行系统维护；“IGNORE_JOBS”标志表示忽略当前运行的作业。在这种情况下，到时候管理员可能需要手工取消运行中的作业以进行系统维护；但是在预约创建成功后，只有预计在预约开始前可以运行结束的作业才会被调度运行。“SPEC_NODES”标志表示创建预约时指定了节点。

eg 创建预约，对节点 cn123 进行故障诊断

```
$ yhcontrol create reservation user=root starttime=now duration=infinite
flags=maint nodes=cn123
Reservation created: root_5
$ yhcontrol show res
ReservationName=root_5 StartTime=2009-02-04T16:22:57
EndTime=2009-02-04T16:21:57 Duration=4294967295
Nodes=cn123 NodeCnt=1
Features=(null) PartitionName=(null)
Flags=MAINT,SPEC_NODES Licenses=(null)
Users=root Accounts=(null)
```

在上例中创建了持续时间为无限长的预约，以避免作业在需要进行故障诊断的节点上运行。同时，MAINT 标志可以表明预约目的为系统维护。

eg 创建周期性预约，每天中午 12:00-13:00 为用户 alan 和 brenda 提供 10 个节点

```
$ yhcontrol create reservation user=alan,brenda starttime=noon duration=60
flags=daily nodecnt=10
Reservation created: alan_6
$ yhcontrol show res
ReservationName=alan_6 StartTime=2009-02-05T12:00:00
EndTime=2009-02-05T13:00:00 Duration=60
Nodes=cn[000-003,007,010-013,017] NodeCnt=10
Features=(null) PartitionName=pdebug
Flags=DAILY Licenses=(null)
Users=alan,brenda Accounts=(null)
```

13.2.2 查看预约

查看预约的命令为：

yhcontrol show reservationname[=*name*]

或

`yhcontrol show reservations` [*name*]

请参见第 13.2.1节中的示例。

13.2.3 修改预约

修改预约的命令为：

`yhcontrol update reservationname=`*name param=val ...*

可修改的参数包括标志、分区、许可证、起止时间、可访问用户/帐号等。可以使用“+=”表示为预约增加标志、增加可访问的用户/帐号，使用“-=”表示为预约删除标志、删除可访问的用户/帐号等。

修改预约的持续时间和可访问用户

```
$ yhcontrol update reservationname=root_3 duration=150 users=admin
Reservation updated.
$ yhcontrol show reservationname=root_3
ReservationName=root_3 StartTime=2009-02-06T16:00:00
EndTime=2009-02-06T18:30:00 Duration=150
Nodes=ALL NodeCnt=20 Features=(null)
PartitionName=(null) Flags=MAINT,SPEC_NODES Licenses=(null)
Users=admin Accounts=(null)
```

13.2.4 删除预约

在结束时间到达后，预约被系统自动删除；预约的信息可以从记账数据里查询。管理员也可以用如下命令手工删除预约；但是当预约中有作业在运行时，不能删除预约：

`yhcontrol delete reservationname=`*name*

13.3 使用预约

要使用预约，用户在提交作业时必须指定预约的名字。这通过 `yhalloc`、`yhbatch` 或 `yhrun` 命令的“`--reservation=`*name*”选项进行。

eg 在预约中运行批处理作业

```
$ yhbatch --reservation=alan_6 -N 4 my.script
yhbatch: Submitted batch job 65540
```

作业必须被完全包含在指定的预约中，即分配给作业的节点必须全部在预约中，且作业的运行期间必须位于预约的时间内。当到达预约的结束时间时，若作业还未运行结束，则作业将被取消。如果想要允许作业在预约的时间结束后继续运行，可以通过系统配置参数 ResvOverRun 设置作业可以继续运行的时间。

13.4 预约的记账

在预约中执行的作业按正常情况记账，即其对资源的使用记录在作业的所有者用户和计费账号上。

如果预约中的资源没有被使用，则将被平分记录在所有可以使用预约中资源的用户和计费帐号上。例如，如果有且仅有两个用户可以使用某个预约，则每个用户将被记录使用了一半的预约资源。

系统维护预约中的资源不被计入系统的使用时间，而是计入系统的计划宕机时间（Planed Down Time）。

第十四章
关联

14.1 概述

14.1.1 关联的概念

所谓关联（association），是指由每个作业可以唯一确定一个四元组：<*cluster*，*account*，*user*，*partition*>。关联的信息保存在系统记账数据库中。

cluster 是作业所在的高性能计算机系统的名字，即系统配置文件中 ClusterName 参数的值。多个高性能计算机系统可以使用一个记账数据库，*cluster* 用于区分不同系统上的关联。

account 即作业所使用的帐号。帐号用于对系统中的用户进行组织，以进行利用统计、收费等。

user 即提交作业的用户。在确定关联时，使用用户的名字而不是用户 UID 进行识别。同一个用户的不同作业可以使用不同的帐号。

partition 即作业所在的分区。如果关联的 *partition* 域为空，表示任意分区。

14.1.2 关联的属性

关联的作用在于管理员可以为关联设置一系列的属性，用于影响用户和帐号的作业运行。

资源限制

每个关联都有一组资源限制。在用户提交、调度和运行作业时，这些资源限制将被实施。

- GrpCPUMins：帐号及其子帐号最多占用的 CPU 时间之和

- GrpCpus：帐号及其子帐号最多同时占用的 CPU 数之和
- GrpJobs：帐号及其子帐号最多同时运行的作业数之和
- GrpNodes：帐号及其子帐号最多同时占用的节点数之和
- GrpSubmitJobs：帐号及其子帐号最多同时提交的作业数之和
- GrpWall：帐号及其子帐号最多占用的墙钟时间之和
- MaxCPUMinsPerJob：每作业最多占用的 CPU 时间
- MaxCpusPerJob：每作业最多占用的 CPU 数
- MaxJobs：最多同时运行的作业数
- MaxNodesPerJob：每作业最多占用的节点数
- MaxSubmitJobs：最多同时提交的作业数
- MaxWallDurationPerJob：每作业最多运行的墙钟时间

关联的资源限制和分区的资源限制可以同时有效。系统配置文件中的参数 EnforcePartLimits 设置是否实施分区资源限制；参数 AccountingStorageEnforce 设置是否实施关联资源限制。

TODO: 注明，对于某些资源限制，如果 QoS 中也定义了，则 Qos 的优先，且这种情况下将不检查相应的关联的资源限制。

公平份额

管理员可以为帐号和用户定义份额，份额实际存储在帐号和用户所对应的关联数据结构中。份额影响作业的调度，详情请参见第 12.4节。

管理员定义帐号或用户的份额时，给出的都是原始份额值。在实际计算关联的份额时，要进行归一化，即将整个系统的全部份额计算为 1，各关联的份额按比例缩小。子帐号和用户共享父帐号的份额。例如，如图 14.1所示的份额结构中，如果有 10 个用户可以使用 A3 帐号，则每个用户的份额为 1%。

QOS

可以定义关联可以使用的 QOS。提交作业时指定的 QOS 必须为管理员定义其关联可以使用的 QOS 级别。QOS 影响作业的资源限制和抢占级别等，详情请参见第 15章。

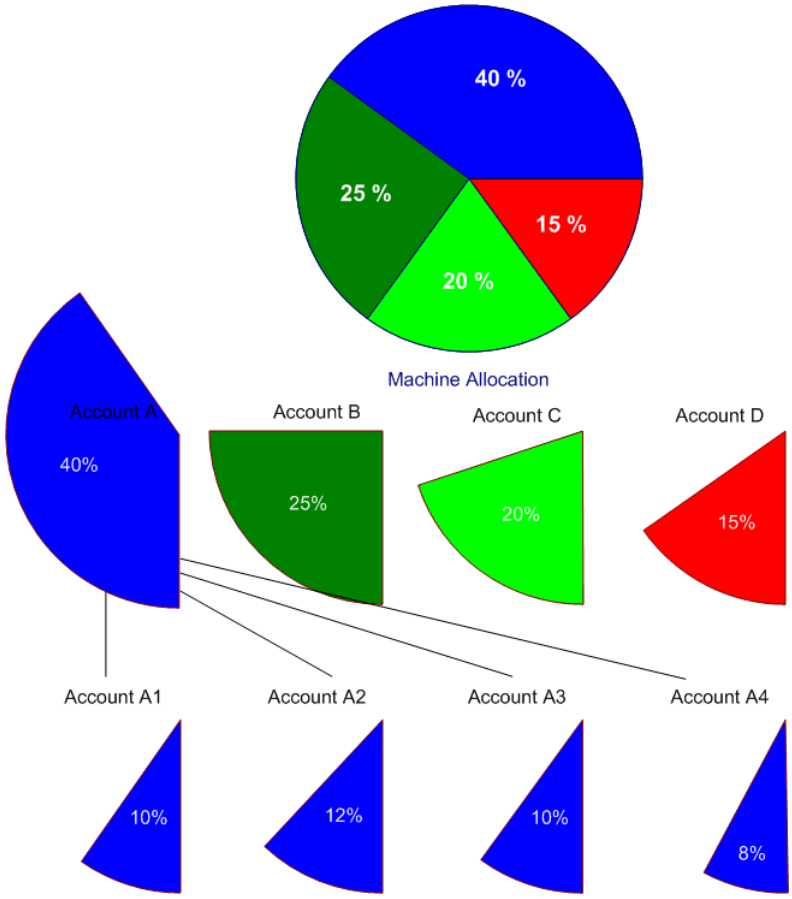


图 14.1: 份额

负载特性

负载特性是一个字符串，用于在另一个维度上区分不同作业的负载类型，例如用“research”表示科研作业，用“business”表示业务作业等。负载特性仅具有标识意义，可用于系统利用情况统计分析。系统配置文件的参数 AccountingStorageEnforce 如果设置了 wckey 则强制作业必须有负载特性才能提交；参数 TrackWCKey 设置是否记录作业的负载特性。

管理员可以为用户定义可使用的和默认的负载特性。如果提交作业时没有指定负载特性，则系统将使用关联的默认负载特性。

14.1.3 关联的控制

管理员通过 `yhacctmgr` 命令操作系统中的关联信息。但是管理员并不直接定义关联，而是通过定义集群、帐号和用户，来隐含地生成或修改相应的关联。当用户或帐号被删除时，相应的关联被删除。

“`yhacctmgr list associations`”命令可以查看系统中的关联。或者，在查看帐号

或用户时，可以请求列出相关关联的信息。

14.2 集群操作

要定义关联信息，管理员首先应向记账数据库中添加所管理的集群（即高性能计算机，并不一定是集群结构）的信息。这通过如下命令进行：

`yhacctmgr add cluster Name=`*name* [*option*=*value*]

可以设置的选项包括：

- 集群类别，Classification。集群类别是管理员对系统做的一种分类，包括 `capacity`，表示运行大量作业；`capability`，表示运行大规模作业；`capapacity`，表示运行大量大规模作业。如果设置的类别值中包含字符“`*`”或串“`class`”，如“`*capacity`”，则表示该集群为机密。系统的分类仅具有标识意义，并无明确界限。
- 公平份额，Fairshare。
- 资源限制，包括 GrpCPUs，GrpJobs，GrpNodes，GrpSubmitJobs，MaxCPUMinsPerJob，MaxCpusPerJob，MaxJobs，MaxNodesPerJob，MaxSubmitJobs，MaxWallDurationPerJob。
- QOS 级别，QosLevel。

这里给出的公平份额、资源限制和 QOS 级别实质上是设置系统的根帐号的属性。

添加集群

```
# yhacctmgr add cluster name=testcluster classification=*capacity
grpjobs=16 grpnodes=64
 Adding Cluster(s)
  Name           = testcluster
  Classification= *Capacity
 Default Limits
  GrpJobs        = 16
  GrpNodes       = 64
Would you like to commit changes? (You have 30 seconds to decide)
(N/y): y
```

一般地，高性能计算机系统名字在系统构建阶段已经由开发人员定义好，并添加到记账数据库中。如果管理员需要更改系统名字，则需要向记账数据库中添加新的系统名字及信息。

“yhacctmgr list cluster”命令可以列出记账数据库中的集群信息，输出格式可以定制；详情请参见第??节。

eg 查看集群信息

```
# yhacctmgr list cluster
   Cluster     ControlHost  ControlPort RPC FairShare GrpJobs GrpNodes
GrpSubmit MaxJobs MaxNodes MaxSubmit     MaxWall                  QOS
---------- --------------- ------------ --- --------- ------- --------
--------- ------- -------- --------- ----------- --------------------
    yhstar       89.72.6.0         6817   8         1
                                                    emergency,normal,st+
testclust+                             0   0         1      16       64
                                                    emergency,normal,st+
```

“yhacctmgr modify cluster”命令可以修改集群信息，可修改的选项与添加集群时相同。

eg 修改集群信息

```
# yhacctmgr modify cluster where name=testcluster set maxjobs=8
maxnodes=32 classification=capability
 Setting
 Default Limits =
  MaxJobs       = 8
  MaxNodes      = 32
 Cluster Classification = Capability
 Modified cluster defaults for associations...
  C = testcluster A = root
  C = testcluster A = root                 U = root
 Modified cluster classifications...
  testcluster
Would you like to commit changes? (You have 30 seconds to decide)
(N/y): y
# yhacctmgr list cluster
   Cluster     ControlHost  ControlPort RPC FairShare GrpJobs GrpNodes
GrpSubmit MaxJobs MaxNodes MaxSubmit     MaxWall                   QOS
---------- --------------- ------------ --- --------- ------- --------
--------- ------- -------- --------- ----------- --------------------
yhstar       89.72.6.0         6817   8         1
                                                    emergency,normal,st+
testclust+                              0   0         1      16       64
                 8       32                         emergency,normal,st+
```

“yhacctmgr delete cluster”命令从记账数据库中删除集群信息。删除集群将把相应的关联信息删除。

eg 删除集群

```
# yhacctmgr delete cluster testcluster
 Deleting clusters...
  testcluster
Would you like to commit changes? (You have 30 seconds to decide)
(N/y): y
```

14.3 帐号操作

帐号是对系统中的用户进行组织的一种方法。系统中的帐号组织成层次结构，通常和用户单位的组织架构一致；每个集群中有一个隐含的名字为 root 的根帐号。如图 14.2所示为某个集群中帐号的示意。

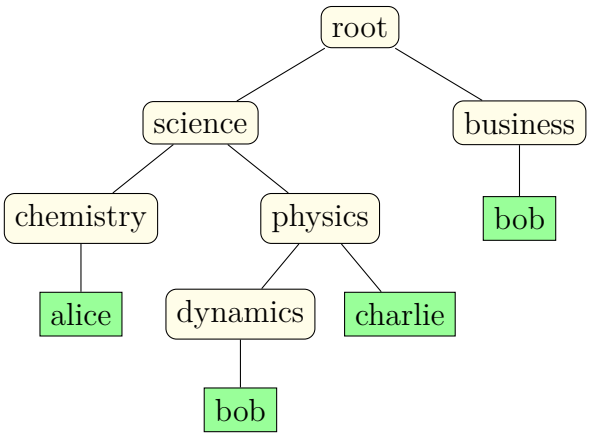


图 14.2: 帐号的层次结构

添加帐号的命令为：

`yhacctmgr add account` [`Names=`|`Accounts=`]*name*[*option*=*val*]

可以设置的选项有：

- 集群，Cluster

  向哪些集群中添加帐号对应的关联（<*cluster*，*account*，，>）。

- 组织单位，Organization

  即此帐号对应与用户单位中的哪个组织架构。

- 描述，Description

  对帐号的描述信息。

- 父帐号，Parent

  默认的父帐号为“root”。

- 公平份额，Fairshare

- 资源限制，包括 GrpCPUMins，GrpCpus，GrpJobs，GrpNodes，GrpSubmitJobs，GrpWall，MaxCPUMinsPerJob，MaxCpusPerJob，MaxJobs，MaxNodesPerJob，MaxSubmitJobs，MaxWallDurationPerJob

- QOS 级别，QosLevel

添加帐号

```
# yhacctmgr add account name=ioc organization=ioc descr="institute of
computer" parent=test cluster=yhstar fairshare=5 qos=standby
 Adding Account(s)
  ioc
 Settings
  Description     = institute of computer
  Organization    = ioc
 Associations
  A = ioc         C = yhstar
 Settings
  Fairshare     = 5
  QOS           = standby
Would you like to commit changes? (You have 30 seconds to decide)
(N/y): y
```

“yhacctmgr list account”命令可列出系统中的帐号及其关联信息；输出信息可定制，详情请参见第??节。

查看帐号

仅查看帐号信息：

```
# yhacctmgr list account
ccount                Descr                  Org
------ -------------------- --------------------
  test                 test                 test
   ioc institute of comput+                  ioc
```

同时查看关联信息：

```
# yhacctmgr list account withassoc
   Account                Descr                  Org    Cluster   Par Name
      User FairShare GrpJobs GrpNodes  GrpCPUs GrpSubmit     GrpWall
 GrpCPUMins MaxJobs MaxNodes  MaxCPUs MaxSubmit     MaxWall  MaxCPUMins
                 QOS
---------- -------------------- -------------------- ---------- ----------
---------- --------- ------- -------- -------- --------- -----------
----------- ------- -------- -------- --------- ----------- -----------
--------------------
      test                 test                 test     yhstar       root
                   1
emergency,normal,st+
      test                 test                 test     yhstar
   test605         1
emergency,normal,st+
       ioc institute of comput+                  ioc     yhstar       test
                   5
             standby
```

帐号信息可通过“yhacctmgr modify account”命令进行修改。

eg 修改帐号

```
# yhacctmgr modify account where name=ioc set grpnodes=20
 Modified account associations...
  C = yhstar     A = ioc of test
Would you like to commit changes? (You have 30 seconds to decide)
(N/y): y
# yhacctmgr list account withassoc
   Account                Descr                  Org    Cluster   Par Name
      User FairShare GrpJobs GrpNodes  GrpCPUs GrpSubmit     GrpWall
 GrpCPUMins MaxJobs MaxNodes  MaxCPUs MaxSubmit     MaxWall  MaxCPUMins
                QOS
---------- -------------------- -------------------- ---------- ----------
---------- --------- ------- -------- -------- --------- -----------
----------- ------- -------- -------- --------- ----------- -----------
--------------------
      test                 test                 test     yhstar       root
                   1
emergency,normal,st+
      test                 test                 test     yhstar
   test605         1
emergency,normal,st+
       ioc institute of comput+                  ioc     yhstar       test
                   5               20
             standby
```

“yhacctmgr delete account”命令从记账数据库中删除帐号。

eg 删除帐号

```
# yhacctmgr delete account ioc
 Deleting accounts...
  ioc
Would you like to commit changes? (You have 30 seconds to decide)
(N/y): y
```

14.4 用户操作

如果系统配置文件中 AccountingStorageEnforce 选项设置了 associations，则只有添加到记账数据库中的用户才能提交作业，因为不然将无法为其作业找到关联。

通过如下命令将用户添加到记账数据库中：

`yhacctmgr add user` [`Names=`|`Users=`]*name* [*option*=*val*]

可以设置的选项有：

- 集群，Clusters

  向哪些集群中添加关联。

- 帐号，Accounts

  添加哪些帐号的关联，即用户可以使用哪些帐号。

- 默认帐号，DefaultAccount

  用户的默认帐号。如未指定，则取用户可使用的第一个帐号为默认帐号。

- 负载特性，WCKeys

  用户可使用的负载特性。

- 默认负载特性，DefaultWCKey

  用户的默认负载特性。如未指定，则取用户可用的第一个负载特性为默认负载特性。

- 分区，Partitions

  添加哪些分区的关联，即用户可以使用哪些分区。默认为任何分区。

- 管理级别，AdminLevel

  可以设置的值为“None”，“Operator”（操作员），“SuperUser 或 Admin”（超级用户）。

  超级用户拥有操作记账数据库中任意数据的权限，而操作员可以查看所有用户和帐号的关联信息与作业记账信息。

- 公平份额，Fairshare

- 资源限制，包括 GrpCPUMins，GrpCpus，GrpJobs，GrpNodes，GrpSubmitJobs，GrpWall，MaxCPUMinsPerJob，MaxCpusPerJob，MaxJobs，MaxNodesPerJob，MaxSubmitJobs，MaxWallDurationPerJob

- QOS 级别，QosLevel

eg 添加用户

```
# yhacctmgr add user testuser accounts=test wckeys=research adminlevel=none
qoslevel=standby
 Adding User(s)
  testuser
 Settings =
  Default Account = test
  Default WCKey   = research
  Admin Level     = None
 Associations =
  U = testuser  A = test       C = yhstar
 WCKeys =
  U = testuser  W = research   C = yhstar
 Non Default Settings
  QOS           = standby
Would you like to commit changes? (You have 30 seconds to decide)
(N/y): y
```

管理员可以将某些用户指定为某些帐号及其子帐号的“协调员”（coordinator）。协调员可以调整帐号内子帐号和用户的资源限制、公平份额、QOS 等。

eg 添加协调员

```
# yhacctmgr add coord names=test605 account=test
 Adding Coordinator User(s)
  test605
 To Account(s) and all sub-accounts
  test
Would you like to commit changes? (You have 30 seconds to decide)
(N/y): y
# yhacctmgr list account withcoord
   Account                Descr                   Org         Coordinators
---------- -------------------- -------------------- --------------------
      test                 test                 test              test605
```

通过“yhacctmgr list users”命令查看系统中的用户及其关联信息。

eg 查看用户

仅查看用户信息：

```
# yhacctmgr list users
      User   Def Acct  Def WCKey     Admin
---------- ---------- ---------- ---------
   test605       test                None
  testuser       test   research      None
```

同时查看关联信息

```
# yhacctmgr list users withassoc
      User   Def Acct  Def WCKey     Admin    Cluster    Account  Partition
FairShare MaxJobs MaxNodes  MaxCPUs MaxSubmit     MaxWall  MaxCPUMins
                QOS
---------- ---------- ---------- --------- ---------- ---------- ----------
--------- ------- -------- -------- --------- ----------- -----------
--------------------
   test605       test                None     yhstar       test
        1
emergency,normal,st+
  testuser       test   research      None     yhstar       test
        1
             standby
```

通过“yhacctmgr modify users”命令可以修改用户数据。

eg 修改用户数据

```
# yhacctmgr modify users where name=testuser set maxjobs=8 maxcpus=16
 Modified account associations...
  C = yhstar     A = test                  U = testuser
Would you like to commit changes? (You have 30 seconds to decide)
(N/y): y
# yhacctmgr list users withassoc testuser
      User   Def Acct  Def WCKey     Admin    Cluster    Account  Partition
FairShare MaxJobs MaxNodes  MaxCPUs MaxSubmit     MaxWall  MaxCPUMins
                  QOS
---------- ---------- ---------- --------- ---------- ---------- ----------
--------- ------- -------- -------- --------- ----------- -----------
--------------------
  testuser       test   research      None     yhstar       test
         1       8                16
              standby
```

“yhacctmgr delete users”删除记账数据库中的用户及其关联。

eg 删除用户

```
# yhacctmgr delete user testuser
 Deleting users...
  testuser
Would you like to commit changes? (You have 30 seconds to decide)
(N/y): y
```

第十五章
作业 QOS

15.1 QOS 概念

QOS，即服务质量，由系统管理员定义，保存在记账数据库中。QOS 用名字进行标识，并可以包含一个可读的描述信息串。QOS 的主要属性是一系列资源限制、QOS 之间的抢占关系、优先级和使用因子。用户提交作业时，指定作业要使用的 QOS（或使用系统默认的 QOS）；用户可使用的 QOS 由管理员设置。系统配置文件中参数 AccountingStorageEnfor 参数确定是否实施 QOS 的资源限制等。

15.1.1 资源限制

QOS 的资源限制如下；这些资源限制在作业提交、作业调度以及作业运行中实施。

- GrpCPUMins：使用此 QOS 的作业可以使用的处理器时间之和
- GrpCpus：使用此 QOS 的作业可以同时使用的处理器数
- GrpJobs：可以同时运行的使用此 QOS 的作业数
- GrpNodes：使用此 QOS 的作业可以同时使用的节点数
- GrpSubmitJobs：可以同时排队、挂起、提交的使用此 QOS 的作业数
- GrpWall：使用此 QOS 的作业的墙钟运行时间之和
- MaxCPUMinsPerJob：使用此 QOS 的作业最多可以使用的处理器时间（处理器 × 分钟）
- MaxCpusPerJob：使用此 QOS 的作业最多可以使用的处理器数
- MaxJobsPerUser：每个用户可以同时运行的使用此 QOS 的作业数

- MaxNodesPerJob：每个使用此 QOS 的作业最多可以使用的节点数
- MaxSubmitJobsPerUser：每个用户可以同时排队、挂起、运行的使用此 QOS 的作业数
- MaxWallDurationPerJob：使用此 QOS 的作业最多可以运行的墙钟时间

作业提交时的检查

- 提交作业数限制 GrpSubmitJobs 和 MaxSubmitJobsPerUser。超出限制后作业不能提交，直到有其它相关作业结束。
- 节点数限制 GrpNodes 和 MaxNodesPerJob。若作业请求的最小节点数超出限制，则作业不能提交；若作业请求的最多节点数超出限制，则其最大节点数被缩小。
- 处理器数限制 GrpCPUs 和 MaxCpusPerJob。若作业请求的最小 CPU 数超出限制，则作业不能提交；若作业请求的最大 CPU 数超出限制，则其最大 CPU 数被缩小。
- 运行时间限制 MaxWallDurationPerJob。若作业请求的运行时间超出限制，则作业不能提交。

作业调度时的检查

作业调度时进行的检查包括：

- 处理器时间限制 GrpCPUMins。超出限制的 QOS 的作业不能运行，原因为 WAIT_ASSOC_JOB_LIMIT。
- 处理器数限制 GrpCPUs。超出限制的 QOS 的作业不能运行，原因为 WAIT_ASSOC_RESOUCE_LIMIT。若作业所需的最小 CPU 数目超出限制，则作业被终止，状态设置为 FAILED，原因为 FAIL_BANK_ACCOUNT。
- 运行作业数限制 GrpJobs。超出限制的 QOS 的作业不能运行，原因为 WAIT_ASSOC_JOB_LIMIT。
- 节点数限制 GrpNodes。超出限制的 QOS 的作业不能运行，原因为 WAIT_ASSOC_RESOUCE_LIMIT。若作业所需的最小 CPU 数目超出限制，则作业被终止，状态设置为 FAILED，原因为 FAIL_BANK_ACCOUNT。
- 运行时间限制 GrpWall。超出限制的 QOS 的作业不能运行，原因为 WAIT_ASSOC_JOB_LIMIT。

- 处理器时间限制 MaxCpuMinsPerJob。根据作业所需的最少 CPU 数目和作业运行时间限制计算。超出限制的作业将被终止，状态设置为 FAILED，原因为 FAIL_BANK_ACCOUNT。
- 处理器数限制 MaxCpusPerJob。若作业需要的最小 CPU 数超出限制，则作业被终止，状态设置为 FAILED，原因为 FAIL_BANK_ACCOUNT。
- 运行作业数限制 MaxJobsPerUser。超出限制的作业不能运行。
- 节点数限制 MaxNodesPerJob。若作业需要的最小节点数超出限制，则作业被终止，状态设置为 FAILED，原因为 FAIL_BANK_ACCOUNT。
- 运行时间限制 MaxWallDurationPerJob。若作业请求的运行时间超出限制，则作业被终止，状态设置为 FAILED，原因为 FAIL_BANK_ACCOUNT。

作业运行中的检查

- 处理器时间限制 GrpCPUMins 和 MaxCpuMinsPerJob。超出限制的作业或超出限制的 QOS 的作业将被终止，作业状态置为 TIMEOUT。
- 运行时间限制 GrpWall。超出限制的作业将被终止，作业状态置为 TIMEOUT。

当作业运行时，其分配的节点数和处理器数被计算到 QOS 的已分配节点数与处理器数中。而随着作业的运行，QOS 的已使用墙钟运行时间和已使用处理器时间被定期更新。与为作业计算优先级时计算公平份额的方法类似，系统对 QOS 已经使用的墙钟运行时间和处理器时间进行定期的衰退，使得最近的使用情况占的权重最大，而历史上的使用信息随着时间消逝对当前的影响逐渐降低。请参见第 12.4节。

15.1.2 抢占与优先级

管理员可指定 QOS 能够抢占的 QOS 列表；抢占是 QOS 之间的一种序关系。如果系统配置文件中设置了使用基于 QOS 的作业抢占，则在为作业选择节点时，根据定义的 QOS 抢占关系和作业的 QOS 查找可被抢占作业。管理员还可为 QOS 指定一个优先级。此优先级除了用于计算作业的优先级外，在作业抢占时优先级低的 QOS 的作业将优先被抢占。参见第 12.2、12.5节。

15.1.3 QOS 使用因子

管理员可以为 QOS 设置一个使用因子（UsageFactor）。其出发点是，使用了高质量服务的作业，在记账收费时，应当考虑进行较高收费。因此，在计算作业对系统资源的使

用时，将其乘上 QOS 的使用因子。即，如果一个作业使用了 10000 处理器 × 分钟，但其 QOS 的使用因子为 1.2，则在收费时将其计算为使用了 10000 × 1.2 处理器 × 分钟。

15.2 QOS 操作

管理员通过 sacctmgr 命令创建、修改、查看和删除 QOS。

eg 创建 QOS

```
# yhacctmgr add qos testqos where grpjobs=16
 Adding QOS(s)
  testqos
 Settings
  Description    = QOS Name
  GrpJobs        = 16
Would you like to commit changes? (You have 30 seconds to decide)
(N/y): y
```

eg 修改 QOS

```
# yhacctmgr modify qos standby set maxnodes=100
 Modified qos...
  testqos
Would you like to commit changes? (You have 30 seconds to decide)
(N/y): y
```

eg 查看系统中的 QOS

```
# yhacctmgr list qos
      Name   Priority    Preempt GrpJobs GrpNodes GrpSubmit MaxJobs
MaxNodes MaxSubmit     MaxWall
---------- ---------- ---------- ------- -------- --------- -------
-------- --------- -----------
    normal          0
   standby          0
 emergency          0
   testqos          0                 16
     100
```

eg 删除 QOS

```
# yhacctmgr delete qos testqos
 Deleting QOS(s)...
  testqos
Would you like to commit changes? (You have 30 seconds to decide)
(N/y): y
```

删除 QOS 时，正在使用此 QOS 的作业不受影响。

帐号和用户可以使用的 QOS 由管理员定义，详情请参见第 14.3与 14.4节。

15.3 QOS 的使用

用户在提交作业时，可以指定作业要使用的 QOS。如未指定，则使用作业关联所允许使用的第一个 QOS。如果管理员未给关联定义可用的 QOS，则使用系统的默认 QOS，“normal”。

eg 指定作业的 QOS

```
yhrun --qos=standby -n 16 a.out
```

指定的 QOS 必须为用户或帐号有权使用的 QOS。

第十六章
作业记账

资源管理系统在记账数据库中记录运行的作业、提交的预约、节点故障等信息，并根据这些信息计算系统的使用情况。用户和管理员可以通过命令工具从记账数据库中查询历史作业信息，分析系统使用情况。

16.1 历史作业查询

yhacct 命令用于查询历史作业信息。默认地，普通用户只能查询自己的作业。可以选择要查询的时间段、作业状态、帐号、节点范围、作业 ID 等过滤条件。详情请参见第??节。

eg 历史作业查询

默认显示用户一天内的作业。

```
$ yhacct
     JobID    JobName  Partition    Account  AllocCPUS      State ExitCode
---------- ---------- ---------- ---------- ---------- ---------- --------
        18       xhpl       work       test          0 CANCELLED+      0:0
        30       xhpl       work       test          0 CANCELLED+      0:0
        34    test.sh       work       test        256  COMPLETED      0:0
        37    test.sh       work       test        256     FAILED      0:0
        38    test.sh       work       test        256     FAILED      0:0
        54       bash       work       test          1  COMPLETED      0:0
       277    test.sh       work       test          0 CANCELLED+      0:0
       317   hostname       work       test          0 CANCELLED+      0:0
...
```

第十七章
全局并行文件系统

17.1 全局文件系统服务常用管理操作

在管理结点 mn0 上，使用 `cfs_stat` 查看某个文件系统的服务状态。例如：

eg 查看文件系统服务状态

```
[root@mn0%panda ~] # cfs_stat
mds0: YHFS-MDT0000:  45   running
ost0:  YHFS-OST0000:  46   running    YHFS-OST0001:  46   running
      YHFS-OST0002:  46   running
ost1:  YHFS-OST0003:  46   running    YHFS-OST0004:  46   running
      YHFS-OST0005:  46   running
```

`cfs_stat -o ost[xx-yy]`，单独查看某个 ost 状态；

OST 的连接数通常为 client 数加 2，MDT 的连接数为 client 数加 1，网络拥塞情况下，连接数会发生动态变化，正常时应趋于稳定。

在管理结点 mn0 上使用 `cfs_mount` 在计算结点上 mount/umount 文件系统。

`cfs_mount -cn cn0`: 在计算结点 cn0 上 mount 文件系统

`cfs_mount -umount cn0`：在计算结点 cn0 上 umount 文件系统

`cfs_mount -cn cn[0-5]`：在计算结点 cn0-cn5 上 mount 文件系统

`cfs_mount -cn cn[0-100,200-210]`：在计算结点 cn0-cn100 以及 cn200-cn210 上 mount 文件系统

17.2 配置限额

在 ln0 上开启配额限制：`lfs quotaon /vol6`

在 ln0 上关闭配额限制：`lfs quotaoff /vol6`

使用下列命令在 ln0 上执行，进行空间限额（quota）的显示和设定。

`lfs quota -u *$username* /vol6`

显示用户 $username 的限额情况，$username 为用户名

`lfs setquota -u *$username* [-g *$groupname*] -b <block-softlimit> -B <block-hardlimit> -i <inode-softlimit> -I <inode-hardlimit><filesystem>`

设定用户 $username 的空间限额，-u $username 指定用户名，-g $groupname 指定用户组名，-b 参数指定空间限额的软限制，-B 参数指定空间限额的硬限制，-i 参数指定文件数限额的软限制，-I 参数指定文件数限额的硬限制，filesystem 指定文件系统目录，在本系统内为/vol6。

例如，使用如下命令查看用户 test605 的空间限额情况。`lfs quota -u test605 /vol6`

17.3 查看文件系统状态

在服务结点上使用 `df` 命令或 `lfs df`，可以查看系统的存储空间使用情况；

eg 查看文件系统状态

```
[test605@ln0%yhstar ~]$ lfs df -h
UUID                    1K-blocks        Used   Available Use% Mounted on
YHFS-MDT0000_UUID         628.3G      491.3M       585.9G   0% /vol6[MDT:0]
YHFS-OST0000_UUID           4.9T        4.2G         4.7T   0% /vol6[OST:0]
YHFS-OST0001_UUID           4.9T       14.0G         4.6T   0% /vol6[OST:1]
YHFS-OST0002_UUID           4.9T        3.2G         4.7T   0% /vol6[OST:2]
YHFS-OST0003_UUID           4.9T        6.0G         4.7T   0% /vol6[OST:3]
YHFS-OST0004_UUID           4.9T        1.9G         4.7T   0% /vol6[OST:4]
YHFS-OST0005_UUID           4.9T        6.0G         4.7T   0% /vol6[OST:5]

filesystem summary:              29.4T       35.2G       27.9T   0% /vol6
```

mn0 上可以使用 cfs_stat 命令查看全局文件系统的元数据服务和 OST 服务状态，命令格式如下：

eg 查看元数据服务和 OST 服务状态

```
[root@mn0%yhstar ~]# cfs_stat
mds0: YHFS-MDT0000:  47  running(healthy)
ost0: YHFS-OST0000: 48  running(healthy)  YHFS-OST0001: 48  running(healthy)
 YHFS-OST0002: 48  running(healthy)
 ost1: YHFS-OST0003: 48  running(healthy)  YHFS-OST0004: 48  running(healthy)
 YHFS-OST0005: 48  running(healthy)
```

Running 表示工作正常，stopped 表示服务停止，recovery 表示正在自动故障恢复。冒号后的数字表示已连接的客户端。

17.4 存储系统升降级

存储系统可以根据具体情况执行系统的升降级操作，保证系统的可用性。

降级操作可以按照如下步骤执行。假定 ost0 的存储设备 sdb 出现故障，需要将文件系统进行降级，则可以在 mn0 上执行如下命令

```
cfs_mask -o YHFS-OST0000 -t off
```

执行此命令后，mds 会将该 ost 设为非活动状态。此时，在计算结点上，可以看到文件系统容量发生变化，而原有分配在该 ost 上的数据将不可见，不可用。

如果需要将系统进行恢复，则可以在 mn0 上执行如下命令：

```
cfs_mask -o YHFS-OST0000 -t on
```

执行此命令后，mds 将 ost 设置为活动状态，文件系统容量恢复。

如果因为 ost0 的存储设备 sdb 出现异常，可以将其设置为只读状态，在 mn0 上执行命令：

```
cfs_mask -o YHFS-OST0000 -t de
```

重新设置为可读可写：

```
cfs_mask -o YHFS-OST0000 -t en
```

17.5 文件条带的设置和查询

省缺条件件下，全局文件系统自动调度文件在 OST 上的分布，同时支持用户手工设定文件在 OST 上的分布，如文件分布到几个 OST 上、条带大小等。使用 lfs 命令配置文件条带分布或获取文件 stripe 信息：

```
lfs [setstripe ] [getstripe ] filename ……
```

可以通过 lfs 的 getstripe 命令获取文件或者目录的 stripe 信息。例如，在目录/vol5 下，我们要获取某个文件 counter.sh 的 stripe 信息，可以执行如下命令：

```
lfs getstripe counter.sh
```

可以得到如下显示：

eg 查看文件条带

```
[root@ln0%yhstar vol6]# lfs getstripe counter.sh
counter.sh
lmm_stripe_count:   1
lmm_stripe_size:    1048576
lmm_stripe_offset:  0
    obdidx            objid          objid            group
    0            3497          0xda9                0
```

则本显示结果说明该文件分布在 0 号 OST 上，该 OST 所对应的结点可以结合 cfs_stat 显示结果确定。该文件相应的 objid 为 3497。

可以通过 lfs 的 setstripe 命令设置目录 stripe 信息。lfs 不能对已存在的文件设置条带，lfs 可以对目录设置 stripe 信息，然后在该目录下新生成的文件将继承目录的条带属性。假定在 vol6 下新建目录 test，使用如下命令设置该目录的 stripe 信息。

```
lfs setstripe /vol6/test <stripe size><stripe start><stripe count> [stripe pattern]
```

其中，stripe size 表示该目录中的文件在每个 OST 上的 stripe 大小，如果设为 0 表示采用系统默认值；stripe start 表示第一个存放该目录中文件的 OST 索引，－ 1 表示采用系统默认值；stripe count 表示该目录下的文件分布的 OST 数；stripe pattern 表示 stripe 分布的模式，1 表示只使用文件 stripe 模式，将文件分为若干 stripe，在默认情况下，系统使用单 stripe 模式。非特殊应用推荐使用系统的默认值。

查找具有相应属性的文件或者目录可采用 lfs find。通过 lfs find 可以查找分布在某个

ost 少的文件。如使用如下命令：

`lfs find --obd YHFS-OST000e_UUID --verbose /vol6/test` 可以查找/vol6/test 目录下，分布在 YHFS-OST000e_UUID 上的文件。