**Aldi中的并行化**

翻译：@老那(asr.naxingyu@gmail.com)

时间：2015年4月

**简介**

使用Kaldi最理想的环境是配备集群任务分发工具，如Sun GridEngine。如果同时使用多个服务器组成的集群，还需要能同时访问的共享文件系统，如NFS。即便没有这些，你也可以在单个一台服务器上方便地安装Kaldi。

在主示例脚本中，如egs/wsj/s5/run.sh，可以看到如下的命令

steps/train\_sat.sh --cmd "$train\_cmd" \

4200 40000 data/train\_si284 data/lang exp/tri3b\_ali\_ai284 exp/tri4a

在run.sh的最上面，可以看到脚本引用了一个叫做cmd.sh的文件

. ./cmd.sh

在cmd.sh里面，可以看到如下的变量赋值语句

export train\_cmd="queue/pl -l arch=\*64"

如果你没有配置GridEngine，或者你的任务队列设置与约翰霍普金斯大学的CLSP实验室（译者注：Daniel Povey所在的实验室，下文中简称CSLP@JHU）不同，那么你需要更改这个变量的值。如果在一台本地服务器上运行，则要设置export train\_cmd=run.pl

在steps/train\_sat.sh中，变量cmd被传给了--cmd选项，也就是说在这个示例中，--cmd选项的值被设为queue/pl -l arch=\*64。在任务脚本中，可以看到如下的命令

$cmd JOB=1:$nj $dir/log/fmllr.$x.JOB.log \

ali-to-post "ark:gunzip -c $dir/ali.JOB.gz|" ark:- \| \

weight-silence-post $silence\_weight $silphonelist $dir/$x.mdl ark:- ark:- \| \

gmm-est-fmllr --fmllr-update-type=$fmllr\_update\_type \

--spk2utt=ark:$sdata/JOB/spk2utt $dir/$x.mdl \

"$feats" ark:- ark:$dir/tmp\_trans.JOB || exit 1;

这条命令的意思是执行$cmd（如queue.pl或者run.pl）指令。它负责产生任务，然后等待它们结束，最后如果中间出现什么差错的话返回一个非零的状态标记。这些指令（除此之外还有其他的如slurm.sh和ssh.sh等）的基本使用方法是

queue.pl <options> <log-file> <command>

最简单的一个例子是

run.pl foo.log echo hellow world

（举这个run.pl的例子是因为它可以在任何系统上运行，不依赖于GridEngine）。也可以尝试创建一个任务队列，如

run.pl JOB=1:10 foo.JOB.log echo hello world number JOB

这样，被执行的指令就会将所有JOB字符替代为给定范围的数字。所以，需要确认你的工作目录中不包含JOB，否则无法正常工作。借助引用符和输出符，你也可以用pipe或者redirection提交任务

run.pl JOB=1:10 foo.JOB.log echo "hello world number JOB" \| head -n 1 \> output.JOB

这种情况下，实际被执行的指令是

echo "hello world number JOB" | head -n 1 > output.JOB

如果想查看实际执行的指令，可以打开foo.1.log文件，其内容如下

# echo "hello world number 1" | head -n 1 output.1

# Started at Sat Jan 3 17:44:20 PST 2015

#

# Accounting: time=0 threads=1

# Ended (code 0) at Sat Jan 3 17:44:20 PST 2015, elapsed time 0 seconds

**并行化工具的通用接口**

这部分介绍并行化工具的使用。这些工具在使用时是可替换的，也就是说，使用一种并行化工具测试成功的脚本，也可以使用其他并行化工具。只需改变$cmd的值，就可以在不同的并行化工具之间切换。

这些工具的基本用法如下

queue.pl <options> <log-file> <command>

下面介绍的内容同样适用于run.pl，ssh.pl和slurm.pl。

< options>可以包含以下内容：

* 任务序号标记（例如JOB=1:10）。名称使用大写仅为方便使用，名称中也可以包含下划线。起始标记必须大于等于1，这是GridEngine的要求。
* 符合GridEngine要求的用于传递给qsub的选项。例如-l arch=\*64\*, or -l mem\_free=6G,ram\_free=6G, or -pe smp 6。除queue.pl以外的其他脚本会忽略这个选项。
* 新定义的选项，例如-mem 10G（见下文）。

< log-file>是一个文件名，对于任务队列，文件名可以包含队列的标记（如exp/foo/log/process\_data.JOB.log）。

< command>可以是任何字串，可以包含shell可解释的符号。但是，如果一个字串先被bash解释了，那就无法再被queue.pl处理。比如，如下的写法是错误的

queue.pl test.log echo foo | awk 's/f/F/';

因为queue.pl无法看到|符号后面的内容，而是会将它的输出交给awk指令。正确的写法是

queue.pl test.log echo foo \| awk 's/f/F/';

对于|、;和>的使用都应当注意。另外，如果< command>的选项中包含空格，queue.pl会假设它应当加\符，并且在传给bash之前替你加上\符号。默认情况下，使用单引用符，如果指令本身包含单引用符，就使用双引用符\\。这种方法通常可以行得通。运行queue.pl的shell环境变量PATH会被传入被执行的脚本中。为保证所需的变量都已定义，./path.sh会再次被引用。然后用bash执行指令。

**新定义的选项（统一接口）**

当我们编写Kaldi示例脚本时，如果需要指定内存需求，就把类似于-l ram\_free=6G,mem\_free=6G的选项传递给queue.pl。在使用steps/train\_sat.sh等脚本时，由于无法获知GridEngine配置的方式（甚至到底有没有GridEngine），这类选项需要从最外层的脚本传进去，这样很奇怪。最近，我们为并行化工具定义了新的接口，使得它们可以理解如下的选项

--config conf/queue\_mod.conf

--mem 10G

--num-threads 6

--max-jobs-run 10

--gpu 1

其中，配置文件定义了如何将新接口转换为GridEngine（或者你使用的其他任务分发工具）支持的格式。目前，只有queue.pl支持这些选项，其他的脚本会忽略这些选项。我们计划逐渐修改steps/中的脚本，使得它们都能支持这种新的接口，并且使queue.pl能够支持任务分发工具的其他选项（如有必要）。

如果有conf/queue.conf配置文件，queue.pl会读取它。否则，会使用代码中定义的默认值。配置文件定义了如何将新的接口转换为GridEngine或者其他任务分发工具支持的选项。下面的例子是默认的配置转换方法

| **新接口** | **转换后 (GridEngine格式)** | **注释** |
| --- | --- | --- |
| –mem 10G | -l mem\_free=10G,ram\_free=10G |  |
| –max-jobs-run 10 | -tc 10 | (We use this for jobs that cause too much I/O). |
| –num-threads 6 | -pe smp 6 | (general case) |
| –num-threads 1 | (no extra options) | (special case) |
| –gpu 1 | -q g.q -l gpu=1 | (general case) |
| –gpu 0 | (no extra options) | (special case for gpu=0) |

可以使用这种格式定义其他选项，也就是形如–foo-bar后面跟着一个参数。默认的配置是与CLSP的集群配合的，在其他地方可能无效，因为GridEngine的配置方式各异。因此，可以创建一个配置文件conf/queue.conf，以配合你的集群。下面的配置文件就是默认配置，可以以此为例创建你自己的配置文件

# Default configuration

command qsub -v PATH -cwd -S /bin/bash -j y -l arch=\*64\*

option mem=\* -l mem\_free=$0,ram\_free=$0

option mem=0 # Do not add anything to qsub\_opts

option num\_threads=\* -pe smp $0

option num\_threads=1 # Do not add anything to qsub\_opts

option max\_jobs\_run=\* -tc $0

default gpu=0

option gpu=0

option gpu=\* -l gpu=$0 -q g.q

command开头的行定义了指令中不可更改的部分，你可以修改这行使它能配合其他任务分发工具使用。option开头的行定义了如何转换输入选项，例如–mem。以"option mem=\*"开头的行处理一些通用的选项（$0被选项的实际输入参数代替）。形如option gpu=0的行允许你指定一些参数的特殊情况，比如在这个例子中的gpu=0。default gpu=0定义了这个选项的默认值，如果你没有给出-gpu的值，queue.pl就会默认你指定了-gpu 0。在这个示例中，我们可以忽略default gpu=0这个配置，因为它没有给出额外的参数信息。在之前的版本中，曾使用option gpu=0 -q all.q，在那种情况下，default gpu=0是有作用的。

由配置文件定义参数到qsub选项参数的转换规则有时需要在perl代码中调整。比如，在现在的脚本中，非分布式任务会忽略–max-jobs-run这个选项。

**计算网格相关配置示例**

我们举一个在真实环境中如何使用配置文件的例子。有一段时间，在使用K20训练神经网络时，程序会崩溃，但是用K10就没事。缘由是我们当时在网格中安装的一个过时的CUDA开发库中存在一个bug。因此，我们创建了一个名为conf/no\_k20.conf的配置文件，其内容与上面的示例类似，但加入了如下几行

default allow\_k20=true

option allow\_k20=true

option allow\_k20=false -l 'hostname=!g01\*&!g02\*&!b06\*'

然后将脚本中的$cmd设置为queue.pl -config conf/no\_k20.conf –allow-k20 false。另外一个等效的方法是直接编辑配置文件中的command行

command qsub -v PATH -cwd -S /bin/bash -j y -l arch=\*64\* -l 'hostname=!g01\*&!g02\*&!b06\*'

这样就不需要添加–allow-k20 false了。

**如何使用不同的脚本进行并行训练**

在这一部分中，我们解释不同的并行化工具是如何使用的。

**使用queue.pl并行化**

queue.pl是Kaldi首选的并行训练工具。设计的初衷是配合GridEngine使用。不过由于我们定义了新的接口，现在它应当可以与其他并行任务分发软件配合了，如Tork和slurm。如果你开发了与这些软件配合的配置文件，请与Kaldi维护团队联系。为了更好的支持其他软件，queue.pl也有必要修改。因为有些命令行是无法通过调用参数配置的，例如，加入-o foo/bar/q/jobname.log以使输出由qsub本身的控制转到另外一个独立的日志文件。另外，对于队列任务，可以向命令行中添加如-t 1:40的选项。在调用qsub运行某个脚本时，它利用了一个环境变量，叫做$SGE\_TASK\_ID，这是一个SGE用来标识不同任务的变量。我们的计划是扩展现有的配置文件机制，使它能够兼容为配合不同任务分发软件所做的任何合理修改。

因为在上文中我们已经用了很多篇幅解释queue.pl的工作原理，这里就不提供更多细节了。但是，请看下面的*如何为Kaldi安装GridEngine*。

**使用run.pl并行化**

如果没有GridEngine，run.pl是一个不错的选择。这个脚本非常简单，它会在本机上运行你所提交的任务，如果你指定了任务队列，如JOB=1:10，它将在本机上提交一系列并行的任务。它不会监测可用的CPU数目和以及内存的容量。所以，如果一个脚本本来应当用queue.pl在一个大的网格中提交，而你用run.pl在本地提交，最后可能耗尽内存或负载资源导致任务失败。如果要使用run.pl，建议你仔细研究将要提交的任务脚本，对于一些解码任务要尤其小心，因为它们通常被提交到后台运行（用&符）。此外，对于一些并行任务数较多的也要格外注意，如–nj 50。通常，你可以减小-nj的值，这样做一般不会影响结果。但在有些脚本中，同样的-nj被多个脚本使用，它们必须保持一致，否则后续的步骤就会崩溃。

除了任务队列标记外，run.pl会忽略其他所有的参数。

**使用ssh.pl并行化**

ssh.pl是穷人版的queue.pl，如果你有一个小的服务器集群，又不想费力安装和配置GridEngine，就可以使用它。与run.pl类似，它并不监测CPU或者内存的使用情况。它的工作机制与run.pl几乎相同，唯一的区别是，它可以把任务分发到不同的机器上去。为了使用它，你需要创建一个叫做.queue/machines的文件（.queue是训练环境的一个子目录）。这个文件的每一行包含一个机器名。使用这个脚本的前提是，这些机器必须支持免密码的ssh登录，也就是说你需要建立一个ssh公钥。

**使用slurm.pl并行化**

slurm.pl是用来兼容slurm任务分发软件的。这个软件的功能和GridEngine差不多。不过这个脚本最近没有测试。现在更好的选择是用queue.pl搭配一个兼容slurm的配置文件，这样就不需要slurm.pl了。

**如何为Kaldi安装GridEngine**

Sun GridEngine（SGE）是一个开源的计算集群\网格管理工具。是Kaldi的维护团队使用最多的版本。目前由甲骨文负责维护，所以他们现在管它叫Oracle GridEngine（译者注：从2013年10月22日起，GridEngine的维护和客户服务已经由Univa软件公司接管）。在CSLP@JHU使用的版本是6.2u5。SGE是一个稳定的工具，所以使用哪个版本其实没那么重要。除SGE外，还有很多其他类似的软件，有一些是后期创建的开源分支。在这个文档中，我们仅指由Oracle维护的原版SGE。

在这一部分，我们将说明如何在一个集群中安装并配置GridEngine。如果你的集群使用亚马逊云EC2，并且要一个能够方便扩展新节点的工具，我建议你看看MIT的StarCluster项目。我们在Sourceforge上也创建过一个类似的项目，叫做kluster，因为当时StarCluster不是很稳定。但相信它现在已经有很大进步了。

**安装GridEngine**

首先，你需要安装GridEngine的基本工具。在GridEngine中，队列管理软件运行在主控节点（master）上，另外一些工具运行在所有队列节点上。master本身也可以加入队列。这里有个概念叫做影子主控（shadow master），相当于master的备份，以防master节点掉线。但在这里我们不做探讨（作者注：我们感觉就是在另外一个节点上安装gridengine-master，然后把master设置为原来的master。但不确定这么做是否可行）。

根据我们的经验，从源代码编译安装GridEngine是非常痛苦的。如果你的Linux发行版的源中包含GridEngine包，那就简单多了。你得谨慎一点，因为并不是所有的发行版中都包含GridEngine包。我们是在Debian上安装的，所以下面介绍如何在Debian上安装和配置。

在主控节点上安装GridEngine的指令是

sudo apt-get install gridengine-master gridengine-client

在选择界面中选择yes进行自动配置。安装程序将会让你输入cell name，可以用默认的default。然后它会问你master的名字，需要填上你所选择作为主控节点的计算机的主机名。通常，需要填写主控计算机的域名全称，但我觉得任何一个能通过主机名查找的方式找到这台计算机名字都可以。需要注意的是，有时候GridEngine会比较挑剔，要求主机名查找和域名（DNS）查找的结果完全匹配。很多GridEngine的运行问题都是由这个问题导致的。另外需要注意，使用apt-get remove删除这些工具并重新安装并不会要求你重新配置，因为Debian会记住你上次安装时的选择（译者注：管理员可以使用aptitude purge完全删除软件绑定的配置文件）。

然后把你自己加入SGE的管理员组

sudo qconf -am <your-user-id>

这里-am表示添加（add）管理员（manager），以此类推，-dm用来删除（delete）管理员，-sm用来显示（show）管理员列表。使用qconf -help查看所有可用的选项。

在普通节点上安装GridEngine的指令是

sudo apt-get install gridengine-client gridengine-exec

同样，cell name不变，master就是你之前安装了主控节点的主机名。你可以在主控节点上安装，这样主控节点也可以运行队列中的任务。

安装完成后，运行qstat和qhost -q，你会看到SGE是否已经开始工作了。下面就是一切正常的情况下你会看到的内容

dpovey\_gmail\_com@instance-1:~$ qstat

dpovey\_gmail\_com@instance-1:~$ qhost -q

HOSTNAME ARCH NCPU LOAD MEMTOT MEMUSE SWAPTO SWAPUS

-------------------------------------------------------------------------------

global - - - - - - -

instance-1.c.analytical-rig-638.internal lx26-amd64 1 0.07 3.6G 133.9M 0.0 0.0

到这一步时，还没有完全配置好，这一步只是用来检查工作节点和主控节点是否已经连通。如果在这一步出现错误，那很可能跟DNS查找、反DNS查找、或者是你的/etc/hostname和/etc/hosts文件的内容有关。当这些信息不一致的时候，GridEngine会罢工。如果你想改变主控节点的名字，需要编辑文件

/var/lib/gridengine/default/common/act\_qmaster

（作者注：这是在Debian Wheezy发行版中的位置）（译者注：修改之后最好运行/etc/init.d/gridengine-exec restart或者重新启动，以确保配置成功被加载）

**配置GridEngine**

首先需要定义一个队列（queue）。因为GridEngine默认不会定义任何队列。我们先来创建一个叫做all.q的队列。请确保shell环境变量EDITOR的值是你常用的编辑器，如vim或emacs。以下的配置方法适用于主控节点和工作节点。

qconf -aq

这条指令会打开一个编辑器。在编辑器中编辑以下这行

qname template

将它改为

qname all.q

然后，将shell的值改为/bin/bash。这个设置比默认值更好，不过对于Kaldi没有影响。退出编辑器后，这些设置会自动保存，但是如果存在语法错误，qconf会拒绝你的修改。后面我们会用qconf -md all.q进一步修改这个队列的属性。

在GridEngine中保存着一些全局配置，这些配置不专属于某个队列。使用qconf -sconf查看这些配置项，或者用qconf -mconf进行编辑修改。比如下面这行

administrator\_mail root

如果你能在服务器上发送邮件（作者注：测试方法是，在服务器上执行mail foobar@gmail.com指令。），那么你就可以把root改成一个有效的邮件地址，这样如果GridEngine出现问题你就会通过邮件收到通知。需要注意的是，由于反垃圾邮件策略，在EC2上从云端发送邮件是非常麻烦的，要想用Google云服务发送邮件更是几乎不可能。所以最好不要修改这个配置项。另外一个可以修改的配置项是

flush\_time=00:00:10

其默认值是00:00:15。这样修改可以使任务提交的速度加快。

在GridEngine中，存在一个叫做“资源”的概念，你可以在任务中请求或指定资源。可以通过qconf -sc查看资源或用qconf -mc修改。修改默认的内存需求的方法是，编辑mem\_free行的值，由0改为1G。

#name shortcut type relop requestable consumable default urgency

#------------------------------------------------------------------------------------------

<snip>

mem\_free mf MEMORY <= YES NO 1G 0

建议加上下面这两行，在任何地方插入都可以

#name shortcut type relop requestable consumable default urgency

#------------------------------------------------------------------------------------------

<snip>

gpu g INT <= YES YES 0 10000

ram\_free ram\_free MEMORY <= YES JOB 1G 0

只有当你想在队列中添加GPU服务时才需要"gpu"项。我们发现ram\_free项在管理节点的内存方面用处很大，而系统默认的mem\_free似乎并不能起到作用。后面往集群中添加节点时，将使用qconf -me <some-hostname>然后编辑complex\_values如下

complex\_values ram\_free=112G,gpu=2

（作者注：此例是对于一个有112G物理内存和2个GPU卡的机器）。如果提交一个内存需求10G的任务，可以在qsub的选项中指定-l mem\_free=10G,ram\_free=10G，mem\_free会确保在任务开始时工作节点有足够的内存。ram\_free确保我们不会往同一个节点上同时提交多个需要大量内存的任务。除了添加ram\_free资源需求外，我们还尝试了用qconf -mc编辑mem\_free项的consumable值为YES，以确保GridEngine会持续监测内存需求。但是这种方法好像不太管用。注意，ram\_free和gpu都是我们选择的名字，对于GridEngine而言它们没有特别的意义，而其他默认的项目，如mem\_free是有确切含义的。ram\_free项的consumable值JOB的意义是ram\_free资源是对每个任务指定的，而不是针对每个线程。这样对Kaldi而言更有用处。

接下来，你需要往GridEngine中添加并行环境smp。大部分GridEngine的管理员都会这样做，但是并没有被默认支持。这是一个非常简单的并行环境，GridEngine并不会做任何特别的事，它只是为你保留一定数目的CPU槽，这样当你执行qsub -pe smp 10 <your-script>时，就会有10个CPU槽可用，这对于多线程或多进程的任务是有帮助的。执行qconf -ap smp，并编辑slots项为999

pe\_name smp

slots 9999

...

然后使用qconf -mq all.q，编辑pe\_list项以添加smp。

pe\_list make smp

这样就在all.q中添加了smp并行环境。

**配置GridEngine（高级）**

在这一部份，我们会介绍一些可能有帮助、但对于基本使用并不必需的小技巧。在CLSP集群中，我们编辑了qconf -mq all.q中的prolog域

prolog /var/lib/gridengine/default/common/prolog.sh

这一项的默认值是NONE。我们将这个脚本拷贝到每个节点上相同的目录中。这一步的作用是，如果任务脚本当前无法获取，管理节点会等待一段时间，因为如果脚本刚刚被编辑，需要给NFS一些时间完成同步。这个脚本的内容如下：

#!/bin/bash

function test\_ok {

if [ ! -z "$JOB\_SCRIPT" ] && [ "$JOB\_SCRIPT" != QLOGIN ] && [ "$JOB\_SCRIPT" != QRLOGIN ]; then

if [ ! -f "$JOB\_SCRIPT" ]; then

echo "$0: warning: no such file $JOB\_SCRIPT, will wait" 1>&2

return 1;

fi

fi

if [ ! -z "$SGE\_STDERR\_PATH" ]; then

if [ ! -d "`dirname $SGE\_STDERR\_PATH`" ]; then

echo "$0: warning: no such directory $JOB\_SCRIPT, will wait." 1>&2

return 1;

fi

fi

return 0;

}

if ! test\_ok; then

sleep 2;

if ! test\_ok; then

sleep 4;

if ! test\_ok; then

sleep 8;

fi

fi

fi

exit 0;

等待至多14秒，这对我们的NFS来说足够了，因为我们在NFS中设置了刷新缓冲目录的最长等待时间为acdirmax=8。

同时在队列配置中，我们设置rerun为TRUE

rerun TRUE

如果任务失败，它们会在qstat的输出中显示为Eqw状态，E表示出错了，通过运行qmod -cj <numeric-job-id>清除出错标记，你可以要求队列重新分配这些任务。或者如果你不想重新运行这些队列，可以用qmod -dj <numeric-job-id>0删除它们。将队列设置为允许重运行，可以避免从头开始。

除此以外，我们还对CLSP队列做了如下修改

rlogin\_daemon /usr/sbin/sshd -i

rlogin\_command /usr/bin/ssh

qlogin\_daemon /usr/sbin/sshd -i

qlogin\_command /usr/share/gridengine/qlogin-wrapper

rsh\_daemon /usr/sbin/sshd -i

rsh\_command /usr/bin/ssh

这些项目原来的默认值是

qlogin\_command builtin

qlogin\_daemon builtin

rlogin\_command builtin

rlogin\_daemon builtin

rsh\_command builtin

rsh\_daemon builtin

这么做的原因已经无从考证了，但是如果你的qlogin和qrsh出现故障时，可以试试这些配置。

**配置GridEngine（添加节点）**

在这一部分中，我们介绍如何在你的队列中添加节点。如上所述，你可以在新的节点上安装GridEngine

sudo apt-get install gridengine-client gridengine-exec

集群名默认，主控节点设为你配置为master的那个节点的主机名。

安装好并不意味着你的新节点已经添加到队列资源中。GridEngine将主机分为管理主机（administrative hosts）、执行主机（execution hosts）和提交主机（submit hosts）。你的新机器需要同时具有这三个角色。你可以通过qconf -sh，qconf -sel和qconf -ss查看这三类主机的列表。通过如下命令将新节点设置为管理主机和提交主机

qconf -ah <新节点的主机名>

qconf -as <新节点的主机名>

用如下命令将新节点配置为执行主机

qconf -ae <新节点的主机名>

这一操作会打开一个编辑器，你可以将前面定义的ram\_free和GPU项添加进去

complex\_values ram\_free=112G,gpu=1

到这里，你可能注意到了在命令行中存在不对称现象，一边是qconf -sh和qconf -ss，而另一边是qconf -sel。后者的"l"表示以列表为命令执行对象。这里的区别在于，管理节点和提交节点的列表中只包含主机，而执行节点的列表包含一系列相关信息，因此，对于执行节点用不同的数据结构加以区别。如果你想查看某个执行节点的详细信息，可以使用qconf -se <节点的主机名>。如果添加或者修改执行节点信息，则分别使用qconf -ae <节点的主机名>和qconf -me <节点的主机名>。这是GridEngine的通用格式。对于包含详细信息的资源，如队列，可以使用"l"结尾的指令查看资源的列表，如qconf -sql，或者用"a"和"m"指令进行资源的操作。

仅仅告知GridEngine执行节点的存在还不够，你得把它加到队列里，然后告诉队列这个节点可以分配多少个任务槽。首先要明确新节点的cpu核心数目：

grep proc /proc/cpuinfo | wc -l

假设是48，那么你可以用一个略小于这个数字的值，如40，作为任务槽的数目。使用qconf -mq all.q编辑队列资源，将新的节点添加到主机列表中，设置任务槽数目，举例如下

qname all.q

hostlist gridnode1.research.acme.com,gridnode2.research.acme.com

<省略若干行>

slots 30,[gridnode1.research.acme.com=48],[gridnode1.research.acme.com=48]

<省略若干行>

在slots项中，开头的30是一个默认值，如果不特别指定，你可以不必把节点添加到这一项。除了这种方法以外，你还可以用主机群的方式简化hostlist项的配置。方法就是，使用qconf -ahgrp @allhosts创建一个主机群，你也可以用过qconf -mhgrp @allhosts修改现有的主机群来添加你的新节点。如果使用这种方法，那么上面配置文件的第二行就应当写为

hostlist @allhosts

用哪种方法配置取决于你自己的选择，对于小型队列，直接写主机列表可能就够了。

用于显示GridEngine所有主机列表，以及每个主机属于哪个队列的指令是qhost -q，结果示例如下

# qhost -q

HOSTNAME ARCH NCPU LOAD MEMTOT MEMUSE SWAPTO SWAPUS

-------------------------------------------------------------------------------

global - - - - - - -

a01.clsp.jhu.edu lx26-amd64 24 12.46 126.2G 11.3G 86.6G 213.7M

all.q BIP 0/6/20

a02.clsp.jhu.edu lx26-amd64 24 16.84 126.2G 12.4G 51.3G 164.5M

all.q BIP 0/18/20

<省略若干行>

如果你在"BIP"的位置看见的是"E"，那说明这个节点出现问题了。其他你不会想看到的标志是，"a"表示警告（通常表示节点的情况不太好），"u"标志状态无法获取，"d"表示这个节点被GridEngine管理员禁用了。有时候运行任务出现问题时，节点会出现"E"标志，这通常是由NFS或其他挂载的问题导致的。你可以清除错误标志

qmod -c all.q@a01

但是，如果节点真的存在严重问题，最好先修复它。这时候你可以禁用它

qmod -d all.q@a01

然后用qmod -e all.q@a01来恢复。

GridEngine出现问题的一种典型症状是当你觉得节点可用时，任务一直在等待。最简单的调试方法是用qstat查看任务标志，然后用qstat -j <任务标志>查看任务没有被执行的具体原因。

你可以用下面的指令查看所有用户提交的任务

qstat -u '\*'

**让你的集群保持稳定**

在这一部份，我们给出一些小提示，以便让你的集群保持稳定并更好的配合Kaldi的使用。

**内存耗尽（OOM）**

计算集群崩溃的一个重要原因就是内存耗尽，Linux自带的OOM杀手并不太好，当你内存耗尽时，可能会杀掉某个重要的系统进程，导致很难诊断的后果。即便没有杀掉任何进程，malloc()的调用也会失败，而很少有程序能友善的处理这个问题。在CLSP集群中，我们自己编写了一个OOM杀手，使用root执行。并编写了相应的初始化脚本。当我们的OOM杀手检测到内存过载时，它会杀掉任意非系统用户进程中占用内存最大的那个。通常这样做是很安全的。这些脚本在我们的kluster项目中开源了，你可以用下面的方式获取并添加到你的系统中。要想让下面的指令正常工作，你需要确认你的系统初始化脚本是LSB风格的，例如Debian Wheezy。在Debian的下一个发布版，jesse中，根本不存在初始化脚本，而是使用systemd。如果你搞清楚了如何在systemd中使用下面的工具，请告诉我们。首先执行

sudo bash

然后作为root执行

apt-get install -y subversion

svn cat https://svn.code.sf.net/p/kluster/code/trunk/scripts/sbin/mem-killer.pl > /sbin/mem-killer.pl

chmod +x /sbin/mem-killer.pl

cd /etc/init.d

svn cat https://svn.code.sf.net/p/kluster/code/trunk/scripts/etc/init.d/mem-killer >mem-killer

chmod +x mem-killer

update-rc.d mem-killer defaults

service mem-killer start

如果某个进程被杀，mem-killer.pl会告知管理员和那个用户。但前提是你的系统有邮件发送功能。

**共享文件系统（NFS）**

我们不会在这里给出NFS安装教程，但是我们希望说明一些潜在的问题，并给出一些可行的解决方法。首先，NFS不是唯一的共享文件系统，还有其他更新的分布式文件系统，但是我们并没有相关经验。

如果配置不当，NFS的性能可能会非常糟糕。下面是我们使用的配置，是用/etc/fstab获取的。其实这不是我们真正使用的方法（我们使用NIS和automount），但是下面的方法更简单。

# grep a05 /etc/fstab

a05:/mnt/data /export/a05 nfs rw,vers=3,rsize=8192,wsize=8192,acdirmin=5,acdirmax=8,hard,proto=tcp 0 0

"vers=3"表示我们使用NFS版本3，我们尝试过版本4，但是经常崩溃。

acdirmin=5和acdirmin=8选项指定了NFS重读缓存目录信息的最短和最长等待时间。默认值是30和60秒。这一点对于Kaldi脚本很重要，因为我们用GridEngine提交的脚本在被提交之前才刚刚生成，所以默认的值会导致脚本的执行节点上无法获取。在前面我们提到了/var/lib/gridengine/default/common/prolog.sh这个脚本指定等待14秒。很明显，14秒大于8秒，也就是说prolog脚本等待的时间比NFS最长的目录刷新周期还长。

hard选项也很重要，它表示如果服务器忙，客户端会等待而不是返回一个错误（例如fopen返回错误）。如果指定soft，Kaldi会崩溃。hard是默认选择，所以可以不用管它。

proto=tcp也是目前Debian的默认值，另外一种选择是proto=udp。如果网络比較拥挤，TCP协议更稳定，这是我们的经验。

rsize=8192,wsize=8192是数据包的大小，它们对于NFS的性能来说是关键因素。Kaldi的读写操作通常连在一起，并且通常不在一个文件内索引，因此大的数据包更好。

另外一个你可以调节的是NFS服务器的线程数。用如下方法查看

/$ head /etc/default/nfs-kernel-server

# Number of servers to start up

RPCNFSDCOUNT=64

更改这个文件，然后重启服务（Debian中：service nfs-kernel-server restart）。显然，最好不要让这个数字小于可能同时访问NFS服务的客户端的数目，你可以用下面的指令，通过retrans观察你的线程数是不是过小了

nfsstat -rc

Client rpc stats:

calls retrans authrefrsh

434831612 6807461 434979729

这个例子中的retrans非常大，理想状态下应该是0。当我们大部分节点在工作时，我们设置的NFS线程数是24，这个值小于可能访问的客户端的数量，所以retrans值比較高。如果有很多客户端同时访问，并且都在写大量数据，它们会占满全部线程，当另一个客户端企图连接时，你会在客户端的日志里发现如下错误nfs: server a02 not responding, still trying.。有些时候这会导致任务失败。如果你使用automount，甚至可能导致一些并没有连接NFS服务的任务失败或被延迟（automount有个脑残的，单线程的设计，所以一个automount请求的失败会导致其他所有automount请求被挂起）。

**其他常见问题**

这一部份我们介绍在计算集群中观察到的一些现象，以及如何配置和管理计算集群。

在CLSP，我们使用多个NFS主机，不仅仅用一两个。实际上，我们大部分的节点还用NFS输出数据。如果你也这样，应该考虑使用我们的mem-killer.pl，否则你会遇到由于用户误操作导致的大量内存耗尽问题。对于很多人共享的任务队列，使用多个文件服务器是个更好的选择，因为用户会难以避免的过载文件服务器。如果只有一两个文件服务器，那么你的队列的性能会收到很大影响。CLSP的单个文件服务器的网络带宽是相当低的，因为成本问题，我们使用1G以太网。但是互相之间用一个强大（昂贵）的Cisco路由连接，所以总的来说网络不会成为瓶颈。这意味着单个文件服务器可能很容易崩溃，但是由于存在大量独立文件服务器，通常只会影响使用那个崩溃的服务器的用户。

当我们检测到某个文件服务器负载较大（比如响应缓慢，或者用dmesg输出nfs: server a15 not responding之类的错误），我们尝试追踪问题的缘由。通常，这都是由用户的不良习惯导致的，例如，用户提交了太多大量占用IO的任务。通常，通过对比iftop和qstat的输出，我们可以知道是哪个用户导致了这个问题。为了保持队列问题，有必要纠正用户的使用习惯，限制类似任务的提交数量。通过给用户发送邮件，提醒他们修改自己的配置。如果不这样做的话，队列资源是根本无法使用的，因为用户会坚持他们的不良习惯。

与我们类似，很多其他的机构也有多个文件服务器，但仍然将所有的流量指向某一个服务器，因为服务器是逐步购买扩充的，而他们总是在最新买的那个服务器上分配新的空间。这样做很蠢。在CLSP，我们将所有的NFS服务器设为全局可写，告诉用户如何用自己的帐户在自己选择的服务器上创建文件夹，避免每个人都去找管理员开辟空间的麻烦。我们编写了脚本，当任何一个文件服务器的用量超过95%时管理员会收到邮件通知，内容包括哪个文件夹占用了大量空间。然后可以告诉相关的用户，让他门删除一些自己的数据，或者根据需要由管理员删除，比如用户已经离开。我们还编写了一个脚本，用来检查在队列中哪个用户占用了大部分资源，并用邮件通知他们在哪里占用了多少资源。另外，当出现一些特殊情况时，管理员也会要求用户清理空间，比如某个高年级学生无缘由地使用了大量的存储空间。存储空间任何时候都不应当成为瓶颈，因为在大部分情况下都有95%是垃圾，没人关心，甚至没有人记得。这个机制只是为了找到有责任清理的那个人，告诉他们自己都保存了哪些没用的东西，让他门的工作效率更高。

另外一个有用的技巧是，在一个不用于实验的文件服务器上分配home目录，这可以保证用户总能得到快速响应，即便其他的用户在做一些愚蠢的事。这也使我们的备份工作更容易进行。我们只备份home目录，而不是那些用于搭建实验环境的NFS卷，并且我们明确告诉用户，这些内容是不进行备份的（当然，有时候学生们会忘记备份他们的重要数据，导致数据丢失）。禁止在home目录下进行实验是一个必要的、需要强调的措施。我们不得不频繁的告诉用户停止那些使用home目录下数据的并行任务。这是导致计算集群出错的最常见的原因。