在 AWS 上优化
多人游戏服务器性能

2017 年 4 月

版权声明

目录

摘要

本白皮书探讨了在 AWS 云中运行多人游戏服务器的绝佳使用案例，以及您可用于实现最高级别性能的优化措施。在本白皮书中，我们为您提供了充分利用 Amazon Elastic Compute Cloud (EC2) 系列实例所需的信息，帮助您实现在 AWS 中成功运行 Linux 多人游戏服务器所需的最强劲性能。

本文面向具备调整和优化 Linux 服务器经验的技术受众。

介绍

Amazon Web Services (AWS) 能够为您想得到的所有游戏工作负载带来好处，包括 PC/游戏机上的单人和多人游戏，以及手机游戏、社交平台游戏和网页游戏。在 AWS 云中运行 PC/游戏机多人游戏服务器清楚地表明了相比传统本地或托管数据中心，云模型能够更好地助您实现成功并且削减成本。

多人游戏服务器基于客户端/服务器网络架构，在这种模式中，游戏服务器保存来自所有客户端 (玩家) 的授权事件源。通常，在玩家将其操作发送到服务器之后，服务器使用所有这些操作模拟运行游戏世界，并将结果发送回各个客户端。

使用 Amazon Elastic Compute Cloud (Amazon EC2) 时，您可以创建并运行虚拟服务器 (称为*实例*) 来托管您的客户端/服务器多人游戏。[1] Amazon EC2 提供了大小可调的计算能力，并支持单根 I/O 虚拟化 (SR-IOV) 及高频率处理器。对于计算系列实例，我们即将在 2017 年推出 C5 计算优化实例类型，届时 Amazon EC2 将能够最多支持 72 个 vCPU (36 个物理内核)。

本白皮书探讨了如何优化 Amazon EC2 Linux 多人游戏服务器，以在实现最佳性能的同时，保持可扩展性、弹性和全球访问能力。首先，我们简要介绍计算优化实例系列的性能水平，然后深入探讨针对联网、CPU、内存和磁盘的优化技术。最后，我们会简单地介绍基准和测试。

Amazon EC2 实例类型注意事项

为了最大限度地发掘 Amazon EC2 实例的性能，请务必了解可用的计算选项。在此部分中，我们讨论使得 Amazon EC2 计算优化实例系列尤其适用于多人游戏服务器的功能。

Amazon EC2 计算优化实例功能

最新一代 C4 计算优化实例系列非常适合运行多人游戏服务器。[2] (在 AWS re:Invent 2016 大会上发布的 C5 实例类型，将会在正式推出后作为推荐游戏服务器平台。) 运行 C4 实例的硬件采用 Intel Xeon E5-2666 v3 (Haswell) 处理器。这是一款专门针对 AWS 设计的定制处理器。下表列出了 C4 系列中各个实例大小的功能。

实例大小	vCPU 数	RAM (GiB)	网络性能	EBS 优化：最大带宽 (Mbps)
c4.large	2	3.75	适中	500
c4.xlarge	4	7.5	适中	750
c4.2xlarge	8	15	高	1000
c4.4xlarge	16	30	高	2000
c4.8xlarge	36	60	10Gbps	4000

如表中所示，c4.8xlarge 实例提供了 36 个 vCPU。由于每个 vCPU 是完整物理 CPU 内核的一个超线程，因此使用此实例类型，您总共可以获得 18 个物理内核。每个内核以 2.9 GHz 基础频率运行，但可以在 3.2 GHz 全内核睿频下运行 (这表示所有内核可同步运行在 3.2 GHz 下，即使所有内核都在使用中也是如此)，睿频加速最高可达 3.5 GHz (只有当少量内核处于使用中时才可以)。

我们建议将 c4.4xlarge 和 c4.8xlarge 实例大小用于运行您的游戏服务器，因为这两种实例分别可以独占访问两个底层处理器插槽中的一个或全部。独占访问可以确保您的大部分工作负载获得 3.2 GHz 全内核睿频。主要的例外情况是运行高级矢量扩展 (AVX) 工作负载的应用程序。[3] 如果您在 c4.8xlarge 实例上运行 AVX 工作负载，则大多数情况下，您可以在运行不超过 3 个内核时实现 3.1 GHz 的最高频率。因此，请务必测试您的特定工作负载，验证您可以获得的性能。

下表显示了 c4.4xlarge 实例与 c4.8xlarge 实例在运行 AVX 和非 AVX 工作负载时的对比情况。

C4 实例大小和工作负载	最大内核睿频 (GHz)	全内核睿频 (GHz)	基础频率 (GHz)
C4.8xlarge — 非 AVX 工作负载	3.5 (活动的 vCPU 数量大致少于 4 个)	3.2	2.9
C4.8xlarge — AVX 工作负载	≤ 3.3	≤ 3.1，具体取决于工作负载和活动内核数	2.5
C4.4xlarge — 非 AVX 工作负载	3.2	3.2	2.9
C4.4xlarge — AVX 工作负载	3.2	≤ 3.1，具体取决于工作负载和活动内核数	2.5

其他计算实例选项

对于一些特殊情况，例如，某些角色扮演游戏 (RPG) 和多人在线竞技游戏 (MOBA)，您的游戏服务器会更多地受限于内存而非计算能力。在这些情况下，由于 M4 实例类型的内存与 vCPU 之比较高，因此可能是比 C4 实例类型更合适的选项。计算优化实例系列的 vCPU 与内存之比相比其他实例系列要高，而 M4 实例的内存与 vCPU 之比更高。M4 实例的 m4.10xlarge 和 m4.16xlarge 使用 Haswell 处理器；较小的使用的是 Broadwell 或 Haswell 处理器。M4 实例类型的联网性能与 C4 实例类型近似，可针对游戏服务器提供相当高的带宽。

性能优化

有众多针对 Linux 服务器的性能选项，而联网和 CPU 是其中最重要的两项。此部分中阐述的性能选项是 AWS 游戏客户发现的最有价值的选项和/或最适合在虚拟机 (VM) 上运行游戏服务器的选项。

这些性能选项分为四个部分：联网、CPU、内存和磁盘。这个性能优化选项列表并不是完整的，而且并不是所有选项都适合于每个游戏工作负载。我们强烈建议先对这些设置进行测试，然后再实施到生产环境中。

此部分假定您在使用 Amazon Virtual Private Cloud (VPC)[4] 创建的 VPC 中运行实例，其中使用了 Amazon 系统映像 (AMI)[5] 及硬件虚拟机 (HVM)。接下来的所有指令和设置都已在使用 4.4.23-31.54 内核的 Amazon Linux AMI 2016.09 上验证，不过它们也应适用于 Amazon Linux 未来的所有发行版本。

联网

联网是性能优化中最重要的因素。多人客户端/服务器游戏对延迟和丢包极为敏感。下表中提供了针对联网的性能优化选项。

性能优化选项	总结	备注	链接或命令
在靠近玩家的位置部署游戏服务器	靠近玩家是减少延迟的最佳方法	AWS 在全球范围内有多个区域。	AWS 区域列表
增强联网	提高联网性能	几乎所有工作负载都会获益。没有不利影响。	Linux/Windows
UDP			
接收缓冲区	有助于防止丢包	当客户端与服务器之间的延迟很高时非常有用。不利影响很小，但应进行测试。	将以下内容添加到 /etc/sysctl.conf： net.core.rmem_default = New_Value net.core.rmem_max = New_Value (建议从将系统的当前值集翻倍开始)
频繁轮询	减少传入数据包处理延迟	可提高 CPU 利用率	将以下内容添加到 /etc/sysctl.conf： net.core.busy_read = New_Value net.core.busy_poll = New_Value (建议先测试值 50，然后测试值 100)
内存	有助于防止丢包		将以下内容添加到 /etc/sysctl.conf： net.ipv4.udp_mem = New_Value New_Value New_Value (建议将系统的当前值集翻倍)
积压	有助于防止丢包		将以下内容添加到 /etc/sysctl.conf： net.core.netdev_max_backlog= New_Value (建议将系统的当前值集翻倍)
传输和接收队列	通过禁用超线程可能实现的性能提升		

以下建议涉及如何减少延迟、避免丢包以及为游戏服务器实现最佳联网性能。

在靠近玩家的位置部署游戏服务器

要想获得出色的玩家体验，将游戏服务器部署到尽可能靠近玩家的位置是一个关键要素。AWS 在全球范围内有多个区域，这使您可以将游戏服务器部署到靠近玩家的位置。有关最新的 AWS 区域和可用区列表，请参阅 https://aws.amazon.com/about-aws/global-infrastructure/。[6]

您可以对实例 AMI 进行打包，并将其部署到尽可能多的精选区域。客户通常会在几乎每个可用区中部署 AAA PC/游戏机游戏。如果您的玩家遍布全球各地，您可以决定在哪里部署游戏服务器，以便尽可能提供最佳的体验。

增强联网

增强联网是另一个性能优化选项。[7] 增强联网使用单根 I/O 虚拟化 (SR-IOV) 并将网卡直接向实例公开，而无需通过管理程序。[8] 这通常可以实现更高的 I/O 性能、更低的 CPU 占用率、更高的每秒数据包数 (PPS) 性能、更低的实例间延迟以及非常低的网络抖动。借助增强联网带来的性能提升，可以让多人游戏服务器实现质的飞跃。

增强联网仅适用于在 VPC 中使用 HVM AMI 运行的特定实例类型的实例，例如 C4、R4、R3、I3、I2、M4 和 D2。这些实例类型使用 Intel 82599 虚拟功能接口 (该接口使用“ixgbevf”Linux 驱动程序。) 此外，X1、R4、P2 和 M4.16xlarge (以及即将推出的 C5) 实例可使用 Elastic Network Adapter (ENA) 支持增强联网。

Amazon Linux AMI 默认包含这些必要的驱动程序。请按照 Linux 或 Windows 说明来安装其他 AMI 的驱动程序。[9,10] 您必须安装最新的 ixgbevf 驱动程序，该驱动程序可从 Intel 网站下载。[11] 建议的最低 ixgbevf 驱动程序版本为 2.14.2。

要检查您的实例上运行的驱动程序版本，请运行以下命令：

```
ethtool -i eth0
```

用户数据报协议 (UDP)

大部分第一人称射击游戏和其他类似的客户端/服务器多人游戏使用 UDP 作为客户端与游戏服务器之间的通信协议。以下部分列出了四个 UDP 优化，这些优化可以改进性能和减少丢包现象。

接收缓冲区

第一个 UDP 优化用于增加接收缓冲区的默认值。UDP 缓冲区空间太小会导致操作系统内核丢弃 UDP 数据包，造成丢包。增加此缓冲区空间有利于改善客户端与服务器之间存在高延迟的情况。Amazon Linux 上的 rmem_default 和 rmem_max 的默认值均为 212992。

要查看系统上的当前默认值，请运行以下命令：

```
cat /proc/sys/net/core/rmem_default
cat /proc/sys/net/core/rmem_max
```

一种分配合适缓冲区空间量的常用方法是，在开始时将这两个值翻倍，然后测试给您的游戏服务器带来的性能差异。根据结果，您可能需要减少或增加这些值。请注意，rmem_default 值不应超过 rmem_max 值。

要将这些参数配置为在重启之后保留，请在 /etc/sysctl.conf file 文件中设置新的 rmem_default 和 rmem_max 值：

```
net.core.rmem_default = New_Value
net.core.rmem_max = New_Value
```

只要对 sysctl.conf 文件进行更改，您就应运行以下命令来刷新配置：

```
sudo sysctl -p
```

频繁轮询

第二个 UDP 优化是频繁轮询，这可以通过让内核轮询传入数据包来帮助减少网络接收路径延迟。这可以提升 CPU 利用率，同时减少数据包处理延迟。

在大部分 Linux 发行版上，包括 Amazon Linux，默认情况下禁用频繁轮询。我们建议您在开始时为 busy_read 和 busy_poll 均使用值 50，然后测试给您的游戏服务器带来的性能差异。Busy_read 是在设备队列上等待套接字读取数据包的毫秒数，而 busy_poll 是在设备队列上等待套接字轮询和选择数据包的毫秒数。根据结果，您可能需要将值增加到 100。

要将这些参数配置为在重启之后保留，请将新的 busy_read 和 busy_poll 值添加到 /etc/sysctl.conf 文件：

```
net.core.busy_read = New_Value
net.core.busy_poll = New_Value
```

同样，在对 sysctl.conf 文件进行更改之后，应运行以下命令来刷新配置：

```
sudo sysctl -p
```

UDP 缓冲区

第三个 UDP 优化是更改 UDP 缓冲区可用于队列的内存量。udp_mem 选项配置 UDP 套接字可用于队列的分页数。这可在网络适配器非常繁忙时帮助减少丢包。

此设置是三个值的矢量，以分页 (4096 字节) 为单位测量。第一个值名为 *min*，是 UDP 开始限制内存使用量的最低阈值。第二个值名为 *pressure*，是 UDP 开始限制内存使用的内存阈值。最后一个值名为 *max*，是可供所有 UDP 套接字用于队列的分页数。默认情况下，c4.8xlarge 实例上的 Amazon Linux 使用矢量 1445727 1927636 2891454，而 c4.4xlarge 实例使用矢量 720660 960882 1441320。

要查看当前默认值，请运行以下命令：

```
cat /proc/sys/net/ipv4/udp_mem
```

一个可用于试验新值的非常不错的方法是先将此设置翻倍，然后测试给您的游戏服务器带来的性能差异。另一种很好的做法是，调整该值，使其为分页大小 (4096 字节) 的整数倍。要将这些参数配置为在重启之后保留，请将新的 UDP 缓冲区值添加到 /etc/sysctl.conf 文件：

```
net.ipv4.udp_mem = New_Value New_Value New_Value
```

在对 sysctl.conf 文件进行更改之后，应运行以下命令来刷新配置：

```
sudo sysctl -p
```

积压

最后一个 UDP 优化是增加积压值，这可以帮助减少丢包概率。此优化将增加传入数据包的队列大小，非常适合接口接收数据包的速度超过内核处理速度的情况。在 Amazon Linux 上，队列大小的默认值为 1000。

要检查默认值，请运行以下命令：

```
cat /proc/sys/net/core/netdev_max_backlog
```

我们建议您将系统的默认值翻倍，然后测试给您的游戏服务器带来的性能差异。要将这些参数配置为在重启之后保留，请将新的 backlog 值添加到 /etc/sysctl.conf 文件：

```
net.core.netdev_max_backlog = New_Value
```

在对 sysctl.conf 文件进行更改之后，应运行以下命令来刷新配置：

```
sudo sysctl -p
```

传输和接收队列

许多游戏服务器由于每秒处理的数据包数过多而对网络造成过高的压力，而不是对使用的总体带宽带来压力。此外，如果其中一个 vCPU 收到了大量的中断请求 (IRQ)，则 I/O 等待也会成为瓶颈。

接收端缩放 (RSS) 是解决这些联网性能问题的一种常用方法。[12] RSS 是一个硬件选项，可以在网络接收控制器 (NIC) 上提供多个接收队列。对于 Amazon Elastic Compute Cloud (Amazon EC2)，NIC 称为 Elastic Network Interface (ENI)。[13] RSS 在 C4 实例系列上启用，但不允许更改 RSS 配置。使用 Linux 时，C4 实例系列会为所有实例大小提供两个接收队列。每个队列具有单独的 IRQ 编号，并且映射到单独的 vCPU。

在 c4.8xlarge 实例上运行命令 `$ ls -1 /sys/class/net/eth0/queues` 将显示以下队列：

```
$ ls -l /sys/class/net/eth0/queues
total 0
drwxr-xr-x 2 root 0 Aug 18 21:00 rx-0
drwxr-xr-x 2 root root 0 Aug 18 21:00 rx-1
drwxr-xr-x 3 root root 0 Aug 18 21:00 tx-0
drwxr-xr-x 3 root root 0 Aug 18 21:00 tx-1
```

要查找队列使用了哪些 IRQ 以及 CPU 如何处理这些中断，请运行以下命令：

```
cat /proc/interrupts
```

或者，运行此命令以输出队列的 IRQ：

```
echo eth0; grep eth0-TxRx /proc/interrupts | awk '{printf
" %s\n", $1}'
```

接下来是在 c4.8xlarge 实例上运行 /proc/interrupts 所生成的完整内容中的部分输出，其中只显示了 etho 中断。第一列是每个队列的 IRQ。后两列是进程信息。在本例中，您可以看到 TxRx-0 和 TxRx-1 分别使用 IRQ 267 和 268。

```
     CPU0  CPU23  CPU33
267  634   2789   0      xen-pirq-msi-x  eth0-TxRx-0
268  600   0      2587   xen-pirq-msi-x  eth0-TxRx-1
```

要验证队列将中断发送到哪个 vCPU，请运行以下命令 (使用各个 TxRx 队列的 IRQ 来替换 IRQ_Number)：

```
$ cat /proc/irq/267/smp_affinity
00000000,00000000,00000000,00800000
$ cat /proc/irq/268/smp_affinity
00000000,00000000,00000002,00000000
```

上一个输出来自 c4.8xlarge 实例。它使用十六进制，需要转换为二进制以查找 vCPU 编号。例如，十六进制值 00800000 转换为二进制之后是 00000000100000000000000000000000。从右侧的 0 开始数，您可以得到 vCPU 23。另一个队列使用 vCPU 33。

由于 vCPU 23 和 33 位于不同处理器插槽中，因此在物理上位于不同的非一致性内存访问 (NUMA) 节点上。这里的一个问题是，默认情况下每个 vCPU 是一个超线程 (但在此特定案例中，它们是同一个内核的两个超线程)，因此将每个队列绑定到一个物理内核可以带来性能提升。

C4 实例系列上的 Amazon Linux 的两个队列的 IRQ 已经固定到位于 c4.8xlarge 实例上的单独 NUMA 节点上的特定 vCPU。此默认状态可能非常适合您的游戏服务器。但是，务必要验证您的 Linux 发行版上为 IRQ 和 vCPU 配置了两个队列 (位于单独的 NUMA 节点上)。对于 c4.8xlarge 以外的其他 C4 实例大小来说，NUMA 不是问题，因为其他大小只有一个 NUMA 节点。

可以改进 RSS 性能的一个选项是禁用超线程。如果您在 Amazon Linux 上禁用超线程，则默认情况下队列将固定到物理内核 (同样也位于 c4.8xlarge 实例的单独 NUMA 节点上)。有关如何禁用超线程的更多信息，请参阅本白皮书中的“超线程”部分。

如果不将游戏服务器进程固定到内核，您可以阻止 Linux 计划程序将游戏服务器进程分配到 RSS 队列的 vCPU (或内核)。为此，您需要配置两个选项。

首先，在您的文本编辑器中，编辑 /boot/grub/grub.conf 文件。对于以“kernel”开头的第一个条目 (可能有多个内核条目，您只需要编辑第一个)，在行的末尾添加 isolcpus=NUMBER，其中 NUMBER 是 RSS 队列的 vCPU 数。例如，如果队列使用 vCPU 3 和 4，则将 NUMBER 替换为“3-4”。

```
# created by imagebuilder
default=0
timeout=1
hiddenmenu
title Amazon Linux 2014.09 (3.14.26-24.46.amzn1.x86_64)
root (hd0,0)
kernel /boot/vmlinuz-3.14.26-24.46.amzn1.x86_64 root=LABEL=/
console=ttyS0 isolcpus=NUMBER
initrd /boot/initramfs-3.14.26-24.46.amzn1.x86_64.img
```

使用 isolcpus 可防止计划程序在您指定的 vCPU 上运行游戏服务器进程。问题是这还会阻止 irqbalance 将 IRQ 分配给这些 vCPU。要解决此问题，您需要使用 IRQBALANCE_BANNED_CPUS 选项来封禁其余的所有 CPU。当前版本的 Amazon Linux 上的 irqbalance 1.1.10 版或更高版本优先使用 IRQBALANCE_BANNED_CPUS 选项，并且将 IRQ 分配给在 isolcpus 中指定的 vCPU，以便避开 IRQBALANCE_BANNED_CPUS 指定的 vCPU。这样，举例而言，如果您使用 isolcpus 隔离了 vCPU 3-4，则您需要在实例上使用 IRQBALANCE_BANNED_CPUS 封禁其他 vCPU。

为此，您需要在 /etc/sysconfig/irqbalance 文件中使用 IRQBALANCE_BANNED_CPUS 选项。这是 64 位的十六进制位掩码。找出值的最佳方法是，以十进制格式写出您希望包括在此值中的 vCPU，然后转换为十六进制。这样，在前面的实例中我们使用了 isolcpus 来排除 vCPU 3-4，然后我们希望使用 IRQBALANCE_BANNED_CPUS 排除 vCPU 1、2 和 5-14 (假设我们使用的是 c4.4xlarge 实例)，其二进制为 1111111111100111，转换为十六进制最终是 FFE7n。使用您常用的编辑器将以下行添加到 /etc/sysconfig/irqbalance 文件：

```
IRQBALANCE_BANNED_CPUS="FFE7n"
```

结果是，游戏服务器进程不使用 vCPU 3 和 4，但它们将由 RSS 队列以及系统所用的其他一些 IRQ 使用。

与其他所有情况类似，所有这些值应该在游戏服务器上进行测试，以便确定性能差异。

带宽

C4 实例系列为多人游戏服务器提供了足够的带宽。c4.4xlarge 实例提供了高网络性能，在使用增强联网并且处于相同置放群组中的两个 c4.8xlarge 实例 (或其他大型实例大小，例如 m4.10xlarge) 之间可以实现最高 10 Gbps。[14] 对于我们所见过的所有游戏服务器使用案例而言，c4.4xlarge 和 c4.8xlarge 实例提供的带宽均绰绰有余。

您可以轻松地确定 C4 实例上工作负载的联网性能与相同可用区中的其他实例以及另一个可用区中的其他实例的对比；最重要的是，与 Internet 的联网性能的对比。Iperf 可能是 Linux 上确定网络性能的最佳工具之一，[15] 而 Nttcp 则是适合 Windows 的工具。[16] 前面的链接还提供了开展网络性能测试的说明。在置放群组之外，您需要使用 Iperf 或 Nttcp 等工具来确定您的游戏服务器所能实现的确切网络性能。

CPU

CPU 是游戏服务器中最重要的两个性能优化因素之一。

性能优化选项	总结	备注	链接或命令
时钟源	使用 tsc 作为时钟源可以改进游戏服务器的性能	Amazon Linux 上的默认时钟源为 Xen。	将以下条目添加到 /boot/grub/grub.conf 文件的内核行上： tsc=reliable clocksource=tsc
C 状态和 P 状态	C 状态和 P 状态选项默认情况下优化，但 c4.8xlarge 上的 C 状态例外。在 c4.8xlarge 上将 C 状态设置为 C1 应该可以改进 CPU 性能。	只有 c4.8xlarge 上可以更改。不利之处在于 3.5 GHz 最大睿频不可用。不过，3.2 GHz 全内核睿频可用。	将以下条目添加到 /boot/grub/grub.conf 文件的内核行上： intel_idle.max_cstate=1
Irqbalance	在未将游戏服务器固定到 vCPU 时，irqbalance 可以帮助改进 CPU 性能。	在 Amazon Linux 上默认安装并运行。检查您的发行版本以确定它是否在运行。	NA
超线程	每个 vCPU 是内核的一个超线程。禁用超线程可能会改进性能。		将以下条目添加到 /boot/grub/grub.conf 文件的内核行上： Maxcpus=X (其中 X 是实例中的实际内核数)
CPU 固定	将游戏服务器进程固定到 vCPU 在一些情况下可带来性能改进。	CPU 固定不是游戏公司中的常见做法。	"numactl --physcpubind $phys_cpu_core --membind $associated_numa_node ./game_server_executable"
Linux 计划程序	有三个特殊的 Linux 计划程序配置选项对游戏服务器有帮助。		sudo sysctl -w 'kernel.sched_min_granularity_ns=New_Value' (建议在开始时将系统的当前值集翻倍) sudo sysctl -w 'kernel.sched_wakeup_granularity_ns=New_Value' sudo sysctril －w (建议在开始时将系统的当前值集减半) 'kernel.sched_migration_cost_ns=New_Value' (建议在开始时将系统的当前值集翻倍)

时钟源

时钟源使得 Linux 可以访问时间线，这样进程可以确定自身在时间方面的进度。对于多人游戏服务器来说，考虑到服务器是事件的权威来源，而每个客户端都有自己的时间体系和事件流，因此时间极为重要。kernel.org 网站深入地介绍了时钟源。[17]

查找当前的时钟源：

```
$cat
/sys/devices/system/clocksource/clocksource0/current_clocksource
```

默认情况下，运行 Amazon Linux 的 C4 实例上时钟源设置为 xen。

查看可用的时钟源：

```
cat
/sys/devices/system/clocksource/clocksource0/available_clocksource
```

对于运行 Amazon Linux 的 C4 实例，此列表默认显示 xen、tsc、hpet 和 acpi_pm。对于大多数游戏服务器，最佳时钟源选项是 TSC (时间戳计数器)，这是各处理器上的 64 位寄存器。大多数情况下，TSC 是速度最快、精度最高的时间流动测量标准，并且稳定不变。请查看此 xen.org 文章，其中详细地讨论了采用 XEN 虚拟化时的 TSC。[18] 同步功能面向处于所有功耗状态的所有处理器提供，因此将 TSC 视为已同步并且不变。这意味着 TSC 将按照恒定速率递增。

TSC 可以使用 rdtsc 或 rdtscp 指令访问。Rdtscp 通常是比 rdtsc 更好的选项，因为 rdtscp 考虑到了 Intel 处理器有时会使用乱序执行，这可能会影响到获取准确的时间读数。

对于游戏服务器，建议将时钟源更改为 TSC。不过，应该针对您的工作负载进行全面的测试。要将时钟源设置为 TSC，请使用选定编辑器来编辑 /boot/grub/grub.conf 文件。对于以“kernel”开头的第一个条目 (请注意，可能有多个内核条目，您只需要编辑第一个)，在行的末尾添加 `tsc=reliable clocksource=tsc`。

```
# created by imagebuilder
default=0
timeout=1
hiddenmenu
title Amazon Linux 2014.09 (3.14.26-24.46.amzn1.x86_64)
root (hd0,0)
kernel /boot/vmlinuz-3.14.26-24.46.amzn1.x86_64 root=LABEL=/
console=ttyS0 tsc=reliable clocksource=tsc
initrd /boot/initramfs-3.14.26-24.46.amzn1.x86_64.img
```

处理器状态控制 (C 状态和 P 状态)

处理器状态控制只能在 c4.8xlarge 实例上修改 (也可在 d2.8xlarge、m4.10xlarge 和 x1.32xlarge 实例上配置)。[19] C 状态控制当内核处于空闲状态时可以进入的休眠级别，而 P 状态控制内核的所需性能 (以 CPU 频率的形式)。C 状态是空闲节能状态，而 P 状态是执行节能状态。

C 状态从 C0 (最浅休眠状态，此时内核实际上在执行各项功能) 开始，一直到 C6 (最深休眠状态，此时内核基本上已关闭)。c4.8xlarge 实例的默认 C 状态为 C6。对于 C4 系列中的所有其他实例大小，默认值为 C1。这就是 3.5 GHz 最大睿频仅在 c4.8xlarge 实例上可用的原因。一些 vCPU 需要进入比 C1 更深层的休眠状态，这样内核才能达到 3.5 GHz。

对于 c4.8xlarge 实例，一种选择是将 C1 设置为最深层 C 状态，从而阻止内核进入休眠状态。这可以减少处理器的反应延迟，但同时会阻止内核在仅有几个内核处于活动状态的情况下达到 3.5 GHz 睿频加速；不过仍允许 3.2 GHz 全内核睿频加速。因此，您可以减少反应延迟，代价是当只有几个内核在运行时达到 3.5 GHz 的可能性会降低。您可获得的结果将取决于您的测试和应用程序工作负载。如果 3.2 GHz 全内核睿频加速可接受，并且您计划利用 C4.8xlarge 实例上的所有或大多数内核，请将 C 状态更改为 C1。

P 状态从 P0 开始 (此时将启用睿频模式)，一直到 P15 (表示可能的最低频率)。P0 提供了最高基准频率。所有 C4 实例大小的默认 P 状态为 P0。完全没理由为游戏工作负载进行更改。睿频加速模式是非常可取的状态。

下表描述了 c4.4xlarge 和 c4.8xlarge 的 C 状态与 P 状态。

实例大小	默认最大 C 状态	建议的设置	默认 P 状态	建议的设置
c4.4xlarge 及更小	1	1	0	0
c4.8xlarge	6[a]	1	0	0

a) 运行 cat /sys/module/intel_idle/parameters/max_cstate 时，最大 C 状态显示为 9。它实际上设置为 6，这是可能的最大值。

使用 turbostat 可查看 C 状态以及可在 c4.8xlarge 实例上达到的最大睿频。同样，这些指令已使用 Amazon Linux AMI 进行测试，仅适用于 c4.8xlarge 实例而不适用于 C4 系列中的任何其他实例大小。

首先，运行以下 turbostat 命令，在系统上安装 stress。 (如果系统上未安装 turbostat，也请安装它。)

```
sudo yum install stress
```

以下命令对两个内核 (即，两个不同的物理内核的两个超线程) 进行压力测试：

```
sudo turbostat --debug stress -c 2 -t 60
```

下面是运行此命令所获得结果的输出内容节选：

Core	CPU	Avg_MHz	%Busy	Bzy_MHz	TSC_MHz	SMI	CPU%c1	CPU%c3	CPU%c6
-	-	188	5.50	3428	2893	0	9.82	0.00	84.68
0	0	4	0.11	3259	2893	0	4.48	0.01	95.40
0	18	4	0.12	3274	2893	0	4.47		
1	1	4	0.11	3270	2893	0	4.45	0.00	95.44
1	19	4	0.11	3280	2893	0	4.45		
2	2	2	0.06	3408	2893	0	99.93	0.00	0.01
2	20	3327	97.20	3431	2893	0	2.79		
3	3	4	0.11	3278	2893	0	4.45	0.00	95.44
3	21	4	0.11	3279	2893	0	4.45		
4	4	3	0.11	3276	2893	0	3.57	0.00	96.32
4	22	4	0.11	3276	2893	0	3.57		
5	5	3	0.10	3277	2893	0	4.50	0.00	95.40
5	23	4	0.11	3279	2893	0	4.49		
6	6	4	0.11	3277	2893	0	4.45	0.00	95.44
6	24	3	0.10	3280	2893	0	4.45		
7	7	4	0.11	3279	2893	0	3.57	0.00	96.32
7	25	4	0.11	3280	2893	0	3.57		
8	8	4	0.11	3266	2893	0	7.09	0.00	92.80
8	26	4	0.12	3276	2893	0	7.08		
0	9	3324	97.13	3430	2893	0	2.86	0.00	0.01
0	27	2	0.05	3407	2893	0	99.94		
1	10	4	0.11	3276	2893	0	4.47	0.00	95.42
1	28	3	0.11	3277	2893	0	4.47		
2	11	4	0.11	3268	2893	0	4.46	0.00	95.43
2	29	4	0.11	3265	2893	0	4.45		
3	12	3	0.11	3270	2893	0	4.46	0.00	95.43
3	30	4	0.11	3270	2893	0	4.46		
4	13	4	0.14	3275	2893	0	3.59	0.00	96.27
4	31	3	0.11	3280	2893	0	3.62		
5	14	4	0.12	3277	2893	0	4.45	0.00	95.43
5	32	3	0.10	3278	2893	0	4.47		
6	15	3	0.11	3276	2893	0	7.11	0.00	92.79
6	33	4	0.12	3264	2893	0	7.09		
7	16	4	0.11	3276	2893	0	4.49	0.00	95.40
7	34	4	0.11	3280	2893	0	4.48		
8	17	4	0.12	3272	2893	0	4.45	0.00	95.43
8	35	3	0.11	3272	2893	0	4.46		

定义：

AVG_MHz：按已用时间划分的已执行周期数。

%Busy：处于“C0”状态的时间百分比。

Bzy_MHz：CPU 繁忙 (处于“c0”状态) 时的平均时钟频率。

TSC_MHz：在整个时间间隔内，TSC 运行的平均 MHz。

此输出指示 vCPU 9 和 20 大部分时间处于 C0 状态 (%Busy)，并且接近最大睿频 3.5 GHz (Bzy_MHz)。这些内核的其他超线程 vCPU 2 和 27 的 C 状态为 C1 (CPU%c1)，表示正在等待指令。由于 c4.8xlarge 实例上的默认 C 状态为 C6，因此已达到接近 3.5 GHz 的频率，并且大多数内核处于 C6 状态 (CPU%c6)。

接下来，尝试对全部 36 个 vCPU 进行压力测试，了解 3.2 GHz 全内核睿频加速的情况：

```
sudo turbostat --debug stress -c 36 -t 60
```

下面是运行此命令所获得结果的输出内容节选：

Core	CPU	Avg_MHz	%Busy	Bzy_MHz	TSC_MHz	SMI	CPU%c1	CPU%c3	CPU%c6
-	-	3189	99.90	3200	2893	0	0.06	0.00	0.05
0	0	3188	99.85	3200	2893	0	0.13	0.00	0.02
0	18	3192	99.98	3200	2893	0	0.01		
1	1	3191	99.94	3200	2893	0	0.01	0.00	0.05
1	19	3188	99.86	3200	2893	0	0.09		
2	2	3191	99.95	3200	2893	0	0.01	0.00	0.04
2	20	3187	99.81	3200	2893	0	0.15		
3	3	3189	99.88	3200	2893	0	0.09	0.00	0.03
3	21	3191	99.96	3200	2893	0	0.01		
4	4	3192	99.99	3200	2893	0	0.00	0.00	0.01
4	22	3189	99.90	3200	2893	0	0.09		
5	5	3190	99.93	3200	2893	0	0.01	0.00	0.06
5	23	3187	99.81	3200	2893	0	0.13		
6	6	3191	99.97	3200	2893	0	0.01	0.00	0.02
6	24	3189	99.89	3200	2893	0	0.09		
7	7	3187	99.83	3200	2893	0	0.09	0.00	0.08
7	25	3190	99.91	3200	2893	0	0.01		
8	8	3187	99.84	3200	2893	0	0.09	0.00	0.07
8	26	3190	99.92	3200	2893	0	0.01		
0	9	3188	99.84	3200	2893	0	0.08	0.00	0.07
0	27	3190	99.91	3200	2893	0	0.01		
1	10	3188	99.84	3200	2893	0	0.09	0.00	0.07
1	28	3190	99.92	3200	2893	0	0.01		
2	11	3188	99.85	3200	2893	0	0.09	0.00	0.06
2	29	3190	99.93	3200	2893	0	0.01		
3	12	3188	99.86	3200	2893	0	0.09	0.00	0.05
3	30	3191	99.94	3200	2893	0	0.01		
4	13	3191	99.95	3200	2893	0	0.01	0.00	0.04
4	31	3186	99.81	3200	2893	0	0.15		
5	14	3192	99.98	3200	2893	0	0.00	0.00	0.01
5	32	3189	99.89	3200	2893	0	0.09		
6	15	3189	99.88	3200	2893	0	0.09	0.00	0.03
6	33	3191	99.96	3200	2893	0	0.01		
7	16	3187	99.82	3200	2893	0	0.09	0.00	0.09
7	34	3189	99.90	3200	2893	0	0.01		
8	17	3192	99.97	3200	2893	0	0.01	0.00	0.02
8	35	3185	99.75	3200	2893	0	0.23		

您会看到对于所有 vCPU，99% 以上的时间都处于 C0 状态 (%Busy)，并且它们在处于 C0 状态时全部达到 3.2 GHz (Bzy_MHz)。

要将 C 状态设置为 C1，请使用选定编辑器来编辑 /boot/grub/grub.conf 文件。对于以“kernel”开头的第一个条目 (可能有多个内核条目，您只需要编辑第一个内核条目)，在行的结尾处添加 `intel_idle.max_cstate=1` 以将 C1 设为空闲内核的最深层 C 状态：

```
# created by imagebuilder
default=0
timeout=1
hiddenmenu
title Amazon Linux 2014.09 (3.14.26-24.46.amzn1.x86_64)
root (hd0,0)
kernel /boot/vmlinuz-3.14.26-24.46.amzn1.x86_64 root=LABEL=/
console=ttyS0 intel_idle.max_cstate=1
initrd /boot/initramfs-3.14.26-24.46.amzn1.x86_64.img
```

保存文件并退出您的编辑器。重启实例以启用新的内核选项。现在，重新运行 turbostat 命令，查看在将 C 状态设置为 C1 后所发生的更改：

```
sudo turbostat --debug stress -c 2 -t 10
```

下面是运行此命令所获得结果的输出内容节选：

Core	CPU	Avg_MHz	%Busy	Bzy_MHz	TSC_MHz	SMI	CPU%c1	CPU%c3	CPU%c6
-	-	178	5.59	3200	2893	0	94.41	0.00	0.00
0	0	1	0.03	3201	2893	0	99.97	0.00	0.00
0	18	1	0.03	3200	2893	0	99.97		
1	1	1	0.03	3201	2893	0	99.97	0.00	0.00
1	19	1	0.03	3201	2893	0	99.97		
2	2	2	0.05	3200	2893	0	99.95	0.00	0.00
2	20	3192	99.99	3200	2893	0	0.01		
3	3	2	0.06	3201	2893	0	99.94	0.00	0.00
3	21	1	0.04	3201	2893	0	99.96		
4	4	1	0.03	3202	2893	0	99.97	0.00	0.00
4	22	1	0.04	3200	2893	0	99.96		
5	5	1	0.04	3201	2893	0	99.96	0.00	0.00
5	23	1	0.03	3200	2893	0	99.97		
6	6	1	0.04	3201	2893	0	99.96	0.00	0.00
6	24	1	0.04	3200	2893	0	99.96		
7	7	1	0.03	3201	2893	0	99.97	0.00	0.00
7	25	1	0.04	3200	2893	0	99.96		
8	8	1	0.03	3201	2893	0	99.97	0.00	0.00
8	26	1	0.03	3200	2893	0	99.97		
0	9	1	0.04	3201	2893	0	99.96	0.00	0.00
0	27	1	0.03	3200	2893	0	99.97		
1	10	1	0.03	3201	2893	0	99.97	0.00	0.00
1	28	1	0.04	3200	2893	0	99.96		
2	11	1	0.03	3201	2893	0	99.97	0.00	0.00
2	29	1	0.04	3200	2893	0	99.96		
3	12	1	0.03	3201	2893	0	99.97	0.00	0.00
3	30	1	0.03	3201	2893	0	99.97		
4	13	1	0.04	3201	2893	0	99.96	0.00	0.00
4	31	1	0.03	3201	2893	0	99.97		
5	14	1	0.04	3201	2893	0	99.96	0.00	0.00
5	32	1	0.04	3200	2893	0	99.96		
6	15	1	0.03	3202	2893	0	99.97	0.00	0.00
6	33	1	0.04	3201	2893	0	99.96		
7	16	3192	99.99	3200	2893	0	0.01	0.00	0.00
7	34	1	0.04	3200	2893	0	99.96		
8	17	1	0.03	3201	2893	0	99.97	0.00	0.00
8	35	1	0.03	3201	2893	0	99.97		

上表中的输出指示，所有内核现在都处于 C 状态 C1。已进行压力测试的两个 vCPU (上述示例中的 vCPU 16 和 2) 的最高平均频率为 3.2 GHz (Bzy_MHz)。由于所有 vCPU 都处于 C1 状态，最大睿频 3.5 GHz 不再可用。

还可以通过运行以下命令来验证 C 状态是否设置为 C1：

```
cat /sys/module/intel_idle/parameters/max_cstate
```

最后，您可能想知道当内核从 C6 切换到 C1 时的性能成本。您可以查询 cpuidle 文件，了解各种 C 状态的退出延迟 (以微秒为单位)。当 CPU 在各个 C 状态之间切换时，会出现延迟。

在默认 C 状态下，cpuidle 指示从 C6 切换到 C0 需要 133 微秒：

```
$ find /sys/devices/system/cpu/cpu0/cpuidle -name latency -o -
name name | xargs cat
POLL
0
C1-HSW
2
C1E-HSW
10
C3-HSW
33
C6-HSW
133
```

在将 C 状态默认值更改为 C1 后，您会发现 CPU 空闲时间的差异。现在，我们可以看到从 C1 切换到 C0 只需要 2 微秒。通过将 vCPU 设置为 C1，我们已将延迟减少了 131 微秒。

```
$ find /sys/devices/system/cpu/cpu0/cpuidle -name latency -o -
name name | xargs cat
POLL
0
C1-HSW
2
```

上述指令只适用于 c4.8xlarge 实例。对于 c4.4xlarge 实例 (以及 C4 系列中的更小实例大小)，C 状态已为 C1，并且全内核睿频加速 3.2 GHz 在默认情况下可用。Turbostat 将不会显示处理器超出基准 2.9 GHz 的情况。存在一个问题，即在对 turbostat 使用调试选项时，c4.4xlarge 实例不会显示 Avg_MHz 或 Bzy_MHz 值，如上面的 c4.8xlarge 实例输出中所示。

一种验证 c4.4xlarge 实例上的 vCPU 达到 3.2 GHz 全内核睿频加速的方法是使用来自 Brendan Gregg 的 showboost 脚本。[20]

要使此方法适用于 Amazon Linux，您需要安装 msr 工具。为此，请运行以下命令：

```
sudo yum groupinstall "Development Tools"
wget https://launchpad.net/ubuntu/+archive/primary/+files/msr-tools_1.3.orig.tar.gz
tar -zxvf msr-tools_1.3.orig.tar.gz
sudo make
sudo make install
cd msr-tools_1.3
wget https://raw.githubusercontent.com/brendangregg/msr-cloud-tools/master/showboost
chmod +x showboost
sudo ./showboost
```

此输出中只显示 vCPU 0，不过您可以修改选项部分来更改将显示的 vCPU。要显示 CPU 频率，请运行您的游戏服务器或使用 turbostat stress，然后运行 showboost 命令以查看 vCPU 的频率。

Irqbalance

Irqbalance 是一项服务，可将中断分布在系统中的各个内核上以改进性能。建议对大多数使用案例使用 Irqbalance，不过将游戏服务器固定到特定 vCPU 或内核的情况除外。在这种情况下，禁用 irqbalance 可能会很有用。请针对特定工作负载进行测试，了解是否存在差异。默认情况下，irqbalance 在 C4 实例系列上运行。

要检查 irqbalance 是否在您的实例上运行，请运行以下命令：

```
sudo service irqbalance status
```

可以在 /etc/sysconfig/irqbalance 文件中配置 Irqbalance。

您希望看到中断非常均匀地分布在所有 vCPU 上。您可以通过运行以下命令查看中断的状态，了解它们是否正确地分布在各个 vCPU 上：

```
cat /proc/interrupts
```

超线程

C4 实例系列上的每个 vCPU 均为物理内核的超线程。如果您确定会对应用程序性能带来不利影响，可以禁用超线程。不过，许多游戏客户未发现有必要禁用超线程。

下表显示了每个 C4 实例大小的物理内核数。

实例名称	vCPU 数	物理内核数
c4.large	2	1
c4.xlarge	4	2
c4.2xlarge	8	4
c4.4xlarge	16	8
c4.8xlarge	36	18

可以运行以下命令来查看所有 vCPU：

```
cat /proc/cpuinfo
```

要获取更具体的输出，可使用以下命令：

```
egrep '(processor|model name|cpu MHz|physical id|siblings|core
id|cpu cores)' /proc/cpuinfo
```

在此输出中，“processor”为 vCPU 编号。“physical id”显示处理器插槽 ID。对于 c4.8xlarge 之外的任何其他 C4 实例，此值将为 0。“core id”为物理内核编号。每个具有相同“physical id”和“core id”的条目将为同一内核的超线程。

查看每个内核的 vCPU 对（即，超线程）的另一种方式是查看每个内核的 thread_siblings_list。这将显示两个代表每个内核的 vCPU 的编号。将“cpuX”中的 X 更改为要查看的 vCPU 编号。

```
cat /sys/devices/system/cpu/cpuX/topology/thread_siblings_list
```

要禁用超线程，请使用选定编辑器编辑 /boot/grub/grub.conf 文件。对于以“kernel”开头的第一个条目 (可能有多个内核条目，您只需要编辑第一个)，在行的末尾添加 maxcpus=NUMBER，其中 NUMBER 是您使用的 C4 实例大小中的实际内核数。参阅上表以了解每个 C4 实例大小中的物理内核数。

```
# created by imagebuilder
default=0
timeout=1
hiddenmenu
title Amazon Linux 2014.09 (3.14.26-24.46.amzn1.x86_64)
root (hd0,0)
kernel /boot/vmlinuz-3.14.26-24.46.amzn1.x86_64 root=LABEL=/
console=ttyS0 maxcpus=18
initrd /boot/initramfs-3.14.26-24.46.amzn1.x86_64.img
```

保存文件并退出您的编辑器。重启实例以启用新的内核选项。

同样，这是您为了确定是否能够实现游戏性能提升而应测试的设置之一。此设置可能需要与 CPU 固定结合使用，才能带来任何性能提升。实际上，在不使用固定的情况下禁用超线程可能会降低性能。AWS 上运行的许多主要 AAA 游戏实际上不禁用超线程。如果无性能提升，您可以不使用此设置，避免因必须在每个游戏服务器上维护此设置而产生的管理开销。

CPU 固定

我们发现许多游戏服务器进程通常都有一个主线程和多个辅助线程。将每个游戏服务器的进程固定到内核 (vCPU 或物理内核) 绝对是一种选择，但并不是我们常见的配置。通常，当游戏引擎真正需要对内核进行独占访问时，将执行内核固定。游戏公司一般只允许 Linux 计划程序处理此工作。同样，应该对此进行测试，但如果在不使用内核固定的情况下性能便已足够，则无需考虑内核固定，这样可为您节省管理开销。

如 NUMA 部分中将讨论的一样，您可以将进程固定到 CPU 内核和 NUMA 节点，方式是运行以下命令 (替换 $phys_cpu_core 和 $associated_numa_node 的值以及 game_server_executable 名称)：

```
“numactl -physcpubind $phys_cpu_core -membind
$associated_numa_node ./game_server_executable”
```

Linux 计划程序

默认的 Linux 计划程序称作完全公平计划程序 (CFS)，[21] 它负责通过控制 CPU 资源的分配来执行进程。CFS 的主要目标是最大程度地提高 vCPU 利用率，实现最佳的整体性能。如果您未将游戏服务器进程固定到 vCPU，则 Linux 计划程序将为这些进程分配线程。

有几个参数可用来优化 Linux 计划程序，它们对游戏服务器非常有用。下面介绍的三个参数的主要目标是，考虑到任务活动的情况下，在合理范围内将任务保留在处理器上尽可能长的时间。我们重点关注计划程序最小粒度、计划程序唤醒粒度以及计划程序迁移成本值。

要查看所有 kernel.sched 选项的默认值，请运行以下命令：

```
sudo sysctl -A | grep -v "kernel.sched_domain" | grep
"kernel.sched"
```

计划程序最小粒度值可配置一个任务在由另一个任务替代之前需在 CPU 上运行的时间。默认情况下，在运行 Amazon Linux 时，C4 实例系列上的此值设置为 3 毫秒。可以增大此值，以使任务在处理器上运行更长时间。一种选项是将此设置倍增至 6 毫秒。与本白皮书中的所有其他性能建议一样，应该对您的游戏服务器全面测试这些设置。由于此计划程序命令和其他两个计划程序命令在重新启动后不保留设置，因此需要在启动脚本中保留设置：

```
sudo sysctl -w 'kernel.sched_min_granularity_ns=New_Value
```

计划程序唤醒粒度值将影响唤醒任务来替代当前正在运行的任务的能力。此值越小，就越容易强制删除任务。默认情况下，在运行 Amazon Linux 时，C4 实例系列上的此值设置为 4 毫秒。您可以选择将此值减半为 2 毫秒并测试结果。进一步减小此值也可能会提高游戏服务器的性能。

```
sudo sysctl -w 'kernel.sched_wakeup_granularity_ns= New_Value'
```

计划程序迁移成本值设定了任务上次执行后的持续时间，在这段时间内，当计划程序做出迁移决定时，任务仍将被视为处于“热缓存”状态。处于“热缓存”状态的任务不太可能会迁移，这有助于降低任务被迁移的可能性。默认情况下，在运行 Amazon Linux 时，C4 实例系列上的此值设置为 4 毫秒。您可以选择将此值倍增至 8 毫秒并进行测试。

```
sudo sysctril  -w 'kernel.sched_migration_cost_ns= New_Value'
```

内存

任何在 c4.8xlarge 实例上运行游戏服务器的客户必须密切关注 NUMA 信息。

性能优化选项	总结	备注	链接或命令
NUMA	在 c4.8xlarge 上，NUMA 会成为一个问题，因为有两个 NUMA 节点。	任何小于 c4.8xlarge 的 C4 实例大小将不会遇到 NUMA 问题，因为它们都具有一个 NUMA 节点。	提供了三个用于处理 NUMA 的选项：CPU 固定、NUMA 平衡和 numad 进程。
虚拟内存	少量虚拟内存调整可提升一些游戏服务器的性能。		将以下内容添加到 /etc/sysctl.conf： vm.swappiness = New_Value (建议在开始时将系统的当前值集减半) 将以下内容添加到 /etc/sysctl.conf： vm.dirty_ratio = New_Value (在 Amazon Linux 上，建议继续使用默认值 20) 将以下内容添加到 /etc/sysctl.conf： vm.dirty_background_ratio = New_Value (在 Amazon Linux 上，建议继续使用默认值 10)

NUMA

当前一代的所有 EC2 实例都支持 NUMA。NUMA 是多处理系统中使用的一个内存架构，可让线程访问本地内存、其他处理器的本地内存或共享内存平台。此处的重要考虑因素是，使用远程内存时的访问速度要比本地内存慢得多。当线程访问远程内存时，会发生性能下降，并且存在互连争用问题。

对于无法利用 NUMA 的应用程序，您需要确保处理器尽可能多地只使用本地内存。这只是 c4.8xlarge 实例面临的问题，因为您能够访问两个处理器插槽，每个插槽均代表一个单独的 NUMA 节点。对于 C4 系列中的小型实例来说，由于仅限您使用单个 NUMA 节点，因此 NUMA 不会成为问题。此外，NUMA 拓扑将在实例的生命周期内保持不变。

c4.8xlarge 实例具有 2 个 NUMA 节点。要查看有关这些节点以及与每个节点关联的 vCPU 的详细信息，请运行以下命令：

```
numactl --hardware
```

要查看 NUMA 策略设置，请运行：

```
numactl --show
```

您还可以在以下目录中查看此信息 (只需在每个 NUMA 节点目录中查看)：

```
/sys/devices/system/node
```

使用 numastat 工具可以查看进程和操作系统的每 NUMA 节点内存统计信息。利用 -p 选项，您可以查看单个进程的此信息，而 -v 选项提供了更详细的数据。

```
numastat -p process_name
numastat -v
```

CPU 固定

下面是在解决潜在 NUMA 性能问题时建议的三个选项。第一个选项是使用 CPU 固定，第二个选项是自动 NUMA 平衡，第三个选项是使用 numad。应测试这三个选项，确定哪个选项能够为您的游戏服务器提供最佳性能。

首先，我们来看看 CPU 固定。这涉及到将游戏服务器进程同时绑定到 vCPU (或核心) 和 NUMA 节点。可使用 numactl 执行此操作。对于实例上运行的每个游戏服务器，在以下命令中更改 $phys_cpu_core 和 $associated_numa_node 的值以及 game_server_executable 名称。有关其他选项，请参阅 numactl 手册页。[22]

```
numactl --physcpubind=$phys_cpu_core --
membind=$associated_numa_node game_server_executable
```

自动 NUMA 平衡

下一个选项是使用自动 NUMA 平衡。此功能尝试在线程或进程所使用的内存所在的处理器插槽中保留这些线程或进程。它还尝试将应用程序数据转移到处理器插槽以供任务访问。从 Amazon Linux Ami 2016.03 开始，默认情况下禁用自动 NUMA 平衡。[23]

要检查是否已在您的实例上启用自动 NUMA 平衡，请运行以下命令：

```
cat /proc/sys/kernel/numa_balancing
```

要永久启用或禁用 NUMA 平衡，请将 `Value` 参数设置为 0 (禁用) 或 1 (启用)，并运行以下命令：

```
sudo sysctl -w 'kernel.numa_balancing=Value'
echo 'kernel.numa_balancing = Value' | sudo tee
/etc/sysctl.d/50-numa-balancing.conf
```

这些指令同样适用于 Amazon Linux。一些版本可能在 /etc/sysctl.conf 文件中进行此设置。

Numad

最后，我们来看看 Numad 选项。Numad 是一个守护程序，可监控 NUMA 拓扑并确保在核心的 NUMA 节点上保留进程。可以根据系统条件的变化做出调整。NUMA 内存管理揭秘一文说明了自动 NUMA 平衡和 numad 选项之间的性能差异。[24]

要使用 numad，您需要先禁用自动 NUMA 平衡。要在 Amazon Linux 上安装 numad，请访问 Fedora numad 网站，然后下载最新提交的稳定版本。[25] 从 numad 目录中，运行以下命令以安装 numad：

```
sudo yum groupinstall "Development Tools"
wget https://git.fedorahosted.org/cgit/numad.git/snapshot/numad-
0.5.tar.gz
tar -zxvf numad-0.5.tar.gz
cd numad-0.5
make
sudo make install
```

可在 /var/log/numad.log 中找到 numad 的日志，并且 /etc/numad.conf 中有一个配置文件。

可通过多种方式运行 numad。numad -u 选项可设置节点的最高使用百分比。默认值为 85%。“NUMA 揭秘”一文中建议的设置为 -u100，此设置会将最大值配置为 100%。这会强制进程保留在本地 NUMA 节点上，直至完全达到其内存要求。

```
sudo numad -u100
```

可使用以下命令来终止 numad：

```
sudo /usr/bin/numad -i0
```

最后，由于您的远程内存访问仍会出现问题，因此完全禁用 NUMA 不是一个好的选择，而使用 NUMA 拓扑更可取。对于 c4.8xlarge 实例，建议对大多数游戏服务器采取某项措施。建议测试我们之前讨论的可用选项，确定哪个选项可提供最佳性能。虽然所有这些选项都无法消除进程对远程 NUMA 节点执行内存调用，但它们均将为游戏服务器提供更佳的体验。

可以通过以下方式测试某个选项所发挥的作用：在实例上运行您的游戏服务器，并使用以下命令查看是否存在任何 numa_foreign (即，已分配给其他 NUMA 节点但应该用于此节点的内存) 和 numa_miss (即，已分配给此节点但应该用于其他 NUMA 节点的内存) 条目：

```
numastat -v
```

测试 NUMA 问题的更常见方式是，使用 stress 等工具，然后运行 numastat 来查看是否存在 foreign/miss 条目：

```
stress --vm-bytes $(awk '/MemFree/{printf "%d\n", $2 * 0.097;}'
< /proc/meminfo)k --vm-keep -m 10
```

虚拟内存

此外，我们发现客户使用的几项虚拟内存调整措施可以提升性能。同样，应该全面测试这些调整措施，确定其是否可提升游戏性能。

虚拟机内存交换

虚拟机内存交换用于控制系统如何考虑匿名内存或分页缓存。较低的值将减少交换进程出现内存不足的次数，这可减少延迟，但会降低 I/O 性能。可能的值为 0 到 100。在 Amazon Linux 上，默认值为 60。建议先将该值减半，然后进行测试。进一步减小此值也可能有助于改进游戏服务器性能。

要查看当前值，请运行以下命令：

```
cat /proc/sys/vm/swappiness
```

要将此参数配置为在重启之后保留，请将带有新值的以下项添加到 /etc/sysctl.conf 文件：

```
vm.swappiness = New_Value
```

虚拟机脏页比率

使用虚拟机脏页比率，可强制进程在特定百分比的系统内存变脏时，阻止写入内存并将脏页写出到磁盘。可能的值介于 0 到 100 之间。在 Amazon Linux 上，默认值为 20 (建议的值)。

要查看当前值，请运行以下命令：

```
cat /proc/sys/vm/dirty_ratio
```

要将此参数配置为在重启之后保留，请将带有新值的以下项添加到 /etc/sysctl.conf 文件：

```
vm.dirty_ratio = New_Value
```

虚拟机脏页后台比率

使用虚拟机脏页后台比率，可强制系统在特定百分比的系统内存变脏时将数据写入磁盘。可能的值为 0 到 100。在 Amazon Linux 上，默认值为 10，这是建议值。

要查看当前值，请运行以下命令：

```
cat /proc/sys/vm/dirty_background_ratio
```

要将此参数配置为在重启之后保留，请将带有建议值的以下项添加到 /etc/sysctl.conf 文件：

```
dirty_background_ratio=10
```

磁盘

由于磁盘很少会成为多玩家游戏服务器的瓶颈，因此磁盘性能调整最不重要。我们尚未发现客户在 C4 实例系列上遇到任何磁盘性能问题。C4 实例系列仅将 Amazon Elastic Block Store (EBS) 用于无本地实例存储的存储；因此，默认情况下，C4 实例已经过 EBS 优化。[26] 如果需要，Amazon EBS 可提供最高 48000 IOPS。您可以执行标准磁盘性能步骤，例如，使用单独的引导卷和操作系统/游戏 EBS 卷。

性能优化选项	总结	备注	链接或命令
EBS 性能	C4 实例在默认情况下会进行 EBS 优化。可以配置 IOPS 以便满足游戏服务器的要求。		NA

基准测试和测试

基准测试

可通过多种方式对 Linux 进行基准测试。您可能会发现 Phoronix 测试套件是很有用的一个选项。[27] 这个基于 Python 的开源套件提供了大量基准测试 (和测试) 选项。您可以针对现有基准进行测试，以便在连续测试后比较结果。您可以将结果上传到 OpenBenchmarking.org 以便在线查看和比较。[28]

提供了许多基准，其中大多数基准可在 OpenBenchmarking.org 测试站点上找到。[29] 对于在为游戏服务器准备工作期间执行的基准测试来说，一些很有用的测试包括 cpu、[30] 多核心、[31] 处理器[32] 和通用测试。[33] 这些测试通常涉及多个子测试。请注意，一些子测试可能不适于下载或无法正常运行。

在开始前，您需要先安装一些必备组件：

```
sudo yum groupinstall "Development Tools" -y
sudo yum install php-cli php-xml  -y
sudo yum install {libaio,pcre,popt}-devel glibc-{devel,static} -y
```

接下来，下载并安装 Phoronix：

```
wget https://github.com/phoronix-test-suite/phoronix-test-suite/archive/master.zip
解压缩 master.zip
cd phoronix-test-suite-master
./install-sh ~/directory-of-your-choice/phoronix-tester
```

要安装测试组件，请从您在运行 install-sh 命令时指定的目录的 bin 子目录运行以下命令：

```
phoronix-test-suite install <测试或套件名称>
```

要安装并运行测试组件，请使用：

```
phoronix-test-suite benchmark <测试或套件名称>
```

您可以选择将结果上传到 Openbenchmark.org。在测试开始时，将显示此选项。如果您选择“yes”，则可指定测试运行。最后，将提供 URL 以查看所有测试结果。在上传结果后，您可以使用以前测试的基准结果数值重新运行基准测试，这样结果将与以前的结果并行显示。您可以重复执行此过程，将多个测试的结果一起显示。通常，您需要进行少量更改，然后重新运行基准测试。您也可以选择不上传测试结果，而是在命令行输出中查看这些结果。

```
phoronix-test-suite benchmark TEST-RESULT-NUMBER
```

以下屏幕截图展示了 OpenBenchmarking.org 上为 c4.8xlarge 实例上运行的一组多核心基准测试显示的输出示例：

multicoretest

ptsh	test1	test2	test3	test4	test5	test6	test7
hmmer: Pfam Database Search	8.06	7.99	7.98	8.01	8.23	8.22	8.20
mafft: Multiple Sequence Alignment	4.08	3.98	4.25	4.23	4.27	3.88	3.85
gcrypt: CAMELLIA256-ECB Cipher	2250	2253	2247	2260	2260	2253	2257
john-the-ripper: Test: Blowfish	11404	11404	11404	11404	11411	7289	7300
x264: H.264 Video Encoding	295.01	297.65	295.59	296.63	286.41	249.39	249.40
himeno: Poisson Pressure Solver	1640.80	1646.83	1642.67	1638.73	1633.64	1640.49	1636.61
compress-7zip: Compress Speed Test	37151	37226	37259	37356	36301	28343	28418
build-apache: Time To Compile	24.84	24.75	24.76	24.82	24.85	25.61	25.53
build-php: Time To Compile	17.75	17.71	17.65	17.74	17.88	20.35	20.27
c-ray: Total Time	13.26	13.27	13.28	13.27	13.26	13.93	13.93
compress-pbzip2: 256MB File Compression	6.21	5.64	5.71	5.62	6.29	7.27	7.24
smallpt: Global Illumination Renderer; 100 Samples	48	47	48	48	48	62	62
bullet: Test: 1000 Convex	6.44	6.43	6.45	6.45	6.46	6.46	6.44
crafty: Elapsed Time	81.23	81.14	81.26	81.55	81.38	81.43	81.41
minion: Benchmark: Solitaire	95.28	93.71	95.50	95.25	95.58	94.68	94.10
minion: Benchmark: Quasigroup	156.10	154.39	156.19	155.48	156.41	154.58	154.68
povray: Total Time	75.58	75.52	75.76	75.99	75.64	90.61	90.58
openssl: RSA 4096-bit Performance	1067.27	1068.37	1067.83	1068.23	1068.10	973.00	973.13

OpenBenchmarking.org

CPU 性能分析

用于 CPU 性能分析的最佳工具之一是 Linux perf 命令。[34] 借助此命令，您可以分别使用 perf record 和 perf report 来记录和分析性能数据。性能分析不在本白皮书的讨论范围内，不过有几个绝佳的资源值得参考：kernel.org wiki 和 Brendan Gregg 的 perf 资源。[35] 下一部分将介绍如何使用 perf 生成火焰图来分析 CPU 使用率。

可视化 CPU 分析

游戏服务器测试期间出现的一个常见问题是，多台游戏服务器正在运行 (通常不固定到 vCPU)，一个 vCPU 接近 100% 利用率，而其他 vCPU 表现出低利用率。解决此类性能问题和其他类似的 CPU 问题是一个复杂且耗时的过程。基本上，此过程涉及查看 CPU 上运行的功能并查找占用 CPU 最多的代码路径。利用 Brendan Gregg 的火焰图，您可以执行可视化分析并解决潜在的 CPU 性能问题。[36] 您还可以快速轻松地标识可视化分析时段内最常使用的功能。

有多种类型的火焰图，包括内存泄漏图，不过我们将重点关注 CPU 火焰图。[37] 我们将使用 perf 命令生成基础数据，然后生成火焰图来创建直观的表示。

首先，安装必备组件：

```
# 安装 Perf
sudo yum install perf

# 让运行 perf record 时不再需要使用 root 身份
sudo sh -c 'echo 0 >/proc/sys/kernel/perf_event_paranoid'

# 下载火焰图
wget
https://github.com/brendangregg/FlameGraph/archive/master.zip

# 最后，您需要解压缩已下载的文件。这将创建一个名为 FlameGraph-master 的目录，其中包含火焰图可执行文件
解压缩 master.zip
```

要查看火焰图中的有用数据，您需要运行游戏服务器或 CPU stress 工具。运行后，再运行 perf 分析记录。您可以对所有 vCPU、特定 vCPU 或特定 PID 运行 perf record。下面是包含各种选项的表：

选项	备注
-F	perf record 的频率。对于大多数使用案例，99 Hz 通常已足够。
-g --	用于捕获堆栈跟踪 (与 CPU 函数或指令相对)。
-C	用于指定要跟踪的 vCPU。
-a	用于指定应跟踪的所有 vCPU。
sleep	指定 perf record 的运行时间 (以秒为单位)。

下面是用于对火焰图运行 perf record 的常见命令，具体取决于您是查看所有 vCPU 还是仅查看一个 vCPU。从 FlameGraph-master 目录运行这些命令：

```
# 在所有 vCPU 上运行 perf record。
perf record -F 99 -a -g -- sleep 60

# 在由 -C 选项后的数字指定的特定 vCPU 上运行 perf record。
perf record -F 99 -C CPU_NUMBER -g -- sleep 60
```

在 perf record 完成后，运行以下命令以生成火焰图：

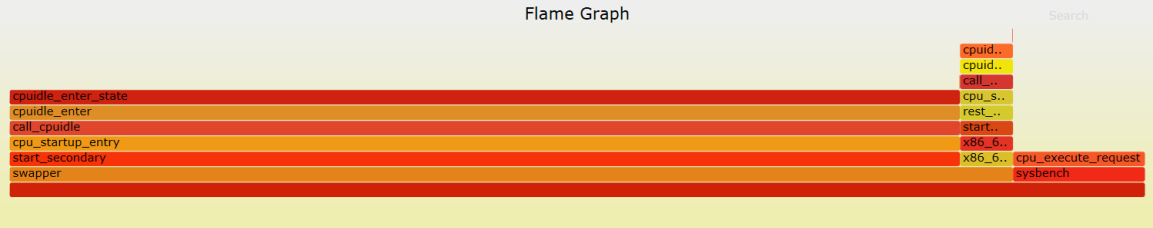
```
# 创建 perf 文件。当您运行此命令时，将显示错误“no symbols found”。由于
我们正在为火焰图生成此项，因此可忽略该错误。
perf script > out.perf


# 使用 stackcollapse 程序将堆栈样本放置到单行中。
./stackcollapse-perf.pl out.perf > out.folded

# 使用 flamegraph.pl 渲染 SVG。
./flamegraph.pl out.folded > kernel.svg
```

最后，使用 WinSCP 等工具将 SVG 文件复制到桌面，以便您查看。

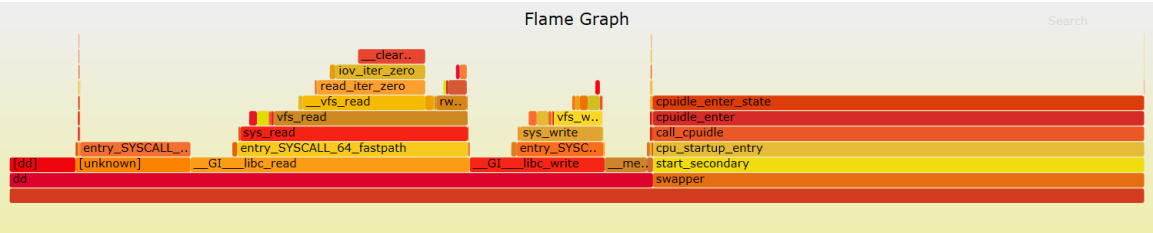
下面是火焰图的两个示例。第一个示例是在运行 sysbench 时使用以下选项在 c4.8xlarge 实例上生成的，时间长度为 60 秒 (for each in 1 2 4 8 16; do sysbench --test=cpu --cpu-max-prime=20000 --num-threads=$each run; done)。您可以看到实例上实际用于 sysbench 的总 CPU 处理时间非常少。您可以将鼠标悬停在火焰图的各个元素的上方，获取有关每个元素的样本数和所花费百分比的详细信息。



第二个示例是在运行以下脚本时在同一 c4.8xlarge 实例上生成的，时间长度为 60 秒：

```
(fulload() { dd if=/dev/zero of=/dev/null |dd if=/dev/zero
of=/dev/null |dd if=/dev/zero of=/dev/null |dd if=/dev/zero
of=/dev/null |dd if=/dev/zero of=/dev/null | dd if=/dev/zero
of=/dev/null | dd if=/dev/zero of=/dev/null | dd if=/dev/zero
of=/dev/null | dd if=/dev/zero of=/dev/null | dd if=/dev/zero
of=/dev/null | dd if=/dev/zero of=/dev/null | dd if=/dev/zero
of=/dev/null  & }; fulload; read; killall dd)
```

输出表示在后台发生的一系列更有用的操作：



结论

本白皮书的目的在于向您说明如何优化您的 EC2 实例，从而在 AWS 上以最佳状态运行游戏服务器。其中重点说明了在 Linux 上运行游戏服务器时，如何实现 C4 实例系列上的网络、CPU 和内存的性能优化。由于磁盘很少会在运行游戏服务器时成为瓶颈，因此对磁盘性能的关注较少。

本白皮书意在对计算实例的相关信息起到提纲契领的作用，为您在 AWS 上运行游戏服务器提供帮助。我们希望本指南中包含的重要信息、性能建议和注意事项可帮助您快速使用 AWS 启动并运行您的游戏服务器，让您的游戏尽可能地取得最大的成功，从而为您节省大量时间。

贡献者

以下为对此文档有贡献的个人和组织：

- Greg McConnel，Amazon Web Services 解决方案架构师
- Todd Scott，Amazon Web Services 解决方案架构师
- Dhruv Thukral，Amazon Web Services 解决方案架构师

备注

[1] https://aws.amazon.com/ec2/

[2] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/c4-instances.html

[3] https://en.wikipedia.org/wiki/Advanced_Vector_Extensions

[4] https://aws.amazon.com/vpc/

[5] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html

[6] https://aws.amazon.com/about-aws/global-infrastructure/

[7] https://aws.amazon.com/ec2/faqs/#Enhanced_Networking

[8] https://en.wikipedia.org/wiki/Single-root_input/output_virtualization

[9] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/enhanced-networking.html

[10] http://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/enhanced-networking-windows.html

[11] https://downloadcenter.intel.com/download/18700/Network-Adapter-Virtual-Function-Driver-for-10-Gigabit-Network-Connections

[12] https://www.kernel.org/doc/Documentation/networking/scaling.txt

[13] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-eni.html

[14] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/placement-groups.html

[15] https://aws.amazon.com/premiumsupport/knowledge-center/network-throughput-benchmark-linux-ec2/

[16] https://aws.amazon.com/premiumsupport/knowledge-center/network-throughput-benchmark-windows-ec2/

[17] https://www.kernel.org/doc/Documentation/timers/timekeeping.txt

[18] https://xenbits.xen.org/docs/4.3-testing/misc/tscmode.txt

[19] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/processor_state_control.html

[20] https://raw.githubusercontent.com/brendangregg/msr-cloud-tools/master/showboost

[21] https://en.wikipedia.org/wiki/Completely_Fair_Scheduler

[22] http://linux.die.net/man/8/numactl

[23] https://aws.amazon.com/amazon-linux-ami/2016.03-release-notes/

[24] http://rhelblog.redhat.com/2015/01/12/mysteries-of-numa-memory-management-revealed/#more-599

[25] https://git.fedorahosted.org/git/numad.git

[26] https://aws.amazon.com/ebs/

[27] http://www.phoronix-test-suite.com/

[28] http://openbenchmarking.org/

[29] http://openbenchmarking.org/tests/pts

[30] http://openbenchmarking.org/suite/pts/cpu

[31] http://openbenchmarking.org/suite/pts/multicore

[32] http://openbenchmarking.org/suite/pts/processor

[33] http://openbenchmarking.org/suite/pts/universe

[34] https://perf.wiki.kernel.org/index.php/Main_Page

[35] http://www.brendangregg.com/perf.html

[36] http://www.brendangregg.com/flamegraphs.html

[37] http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html