AWS 에서의 멀티 플레이어 게임 서버 성능 최적화

2017 년 4 월

Notices

This document is provided for informational purposes only. It represents AWS's current product offerings and practices as of the date of issue of this document, which are subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS's products or services, each of which is provided "as is" without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

공지사항

이 문서는 정보제공을 목적으로 제공됩니다. 문서에 설명된 제품들과 방법들은 본 문서가 공개된 날짜를 기준으로 유효한 내용이며 추후 사전 공지 없이 변경될 수 있습니다. 이 문서에 나와있는 서비스나 설명 및 방법들에 대한 최종 판단은 고객 여러분의 개별적인 판단에 달려있으며, 이를 어떤 종류로든 보장하지는 않습니다. 이 문서에 대한 내용에 대해서 AWS 에서는 어떠한 보장도 제공하지 않습니다. AWS 와 고객 여러분 간의 법적인 책임은 AWS 와의 합의(Agreement)에 따르며 이 문서는 AWS와 고객간의 맺는 합의(Agreement)에 해당하지 않으며 어떠한 영향도 주지 않습니다.

본 문서는 아래 문서의 번역본으로 약간의 오류 또는 잘못된 표현이 있을 수 있습니다. 보다 정확한 정보에 대해서는 아래 영어로 출판 된 원문을 참조하십시오.

원본 문서 : Optimizing Multiplayer Game Server Performance on AWS

한글번역: 김병수 솔루션즈 아키텍트 - AWS Solutions Architect team

목차

개요

이 백서는 AWS 클라우드에서 멀티 플레이어 게임 서버들을 구동하는 사용사례와 최고 수준의 성능을 달성하기 위한 최적화에 대해 설명합니다. 이 백서는 AWS 상에서 구동되는 Linux 기반의 멀티 플레이어 게임 서버를 성공적으로 구동하는데에 필요한 성능을 얻기 위하여 Amazon EC2(Elastic Compute Cloud)에서 활용해야 하는 정보들을 제공하고 있습니다.

이 백서는 Linux 베이스의 서버들을 튜닝하고 최적화한 기술적인 경험이 있는 독자들을 상정하고 있습니다.

소개

Amazon 웹 서비스(AWS)는 PC/콘솔 싱글 플레이어 및 멀티 플레이어 게임은 물론 소셜 기반 및 웹 기반 게임을 아우르는 생각할 수 있는 모든 게임 작업에 이점을 제공합니다. AWS 클라우드에서 PC/콘솔 멀티플레이어 게임 서버를 구동하는 것은 특히 전통적인 방식의 on-premise 데이터센터나 colocation 을 사용하는 것에 비해 클라우드 모델로 달성할 수 있는 비용절감 및 이점을 잘 보여줍니다.

멀티 플레이어 게임 서버는 게임서버가 모든 클라이언트(사용자)들에 대한 이벤트를 처리하는 클라이언트/서버 네트워크 아키텍처를 기반으로 합니다. 개략적으로 보면 게임서버는 플레이어 행위가 서버에 전송되면 서버는 이러한 모든 행위들을 사용하여 게임 세계의 시뮬레이션을 실행하고 결과를 각 클라이언트에 되돌려 보내는 방식으로 동작합니다.

Amazon EC2 (Amazon Elastic Compute Cloud)를 사용하면 클라이언트/서버 멀티 플레이어 게임을 호스트하는 가상 서버 (인스턴스라고 함)를 생성하고 실행할 수 있습니다.[1] Amazon EC2 는 조정 가능한 컴퓨팅 용량을 제공하며 단일 루트 I/O 가상화 (SR-IOV), 높은 clock 의 프로세서들을 지원합니다. Amazon EC2 연산 최적화 제품군은 2017 년에 소개된 C5 유형의 경우 72 개의 vCPU(36 개의 물리코어)까지 지원합니다.

이 백서는 확장성, 탄력성 및 글로벌 서비스를 위한 도달 거리 최소화를 만족하면서 동시에 최고의 성능을 달성하기 위해 Amazon EC2 Linux 멀티 플레이어 게임 서버를 최적화하는 방법에 대해 설명합니다. 연산 최적화된 인스턴스 제품군의 성능 기능에 대한 간략한 설명부터 시작하여 네트워킹, CPU, 메모리 및 디스크에 대한 최적화 테크닉에 대해 살펴 보겠습니다. 마지막으로 벤치마킹 및 테스트에 대해서도 간략하게 다루고 있습니다.

Amazon EC2 인스턴스 유형 고려사항

Amazon EC2 인스턴스에서 최대한의 성능을 얻으려면 사용 가능한 컴퓨팅 옵션을 살펴 보는 것이 중요합니다. 이 절에서는 멀티 플레이어 게임 서버에 이상적인 Amazon EC2 연산 최적화 인스턴스 제품군의 기능에 대해 설명합니다.

Amazon EC2 연산 최적화 인스턴스 기능

현재 세대의 C4 연산 최적화 인스턴스 제품군은 멀티 플레이어 게임 서버를 실행하는 데 이상적입니다.[2] (AWS re:Invent 2016 에서 발표된 C5 인스턴스 유형이 출시되면 C5 인스턴스 유형이 권장되는 게임 서버용 인스턴스 유형입니다.) C4 인스턴스는 Intel Xeon E5-2666 v3 (Haswell) 프로세서를 사용하는 하드웨어에서 구동됩니다. 이 프로세서는 AWS 를 위해 특별히 설계된 맞춤형 프로세서입니다. 다음 표는 C4 제품군의 각 인스턴스 크기별로 기능을 나열합니다.

인스턴스 크기	vCPU 개수	RAM (GiB)	네트워크 성능	EBS 최적화: 최대 대역폭(Mbps)
c4.large	2	3.75	중간	500
c4.xlarge	4	7.5	중간	750
c4.2xlarge	8	15	높음	1000
c4.4xlarge	16	30	높음	2000
c4.8xlarge	36	60	10 Gbps	4000

표에 표시된 것처럼 c4.8xlarge 인스턴스는 36 개의 vCPU 를 제공합니다. 각 vCPU 는 물리 CPU 코어의 하이퍼스레드이므로, 이 인스턴스 사이즈에서는 총 18 개의 물리적 코어가 제공됩니다. 각 코어는 기본 2.9GHz 에서 동작하지만 3.2GHz 의 모든 코어 터보 동작속도(모든 코어가 동시에 3.2GHz 에서 동작 가능) 및 3.5GHz 의 최대 터보 동작속도(몇 개의 코어만 사용 중일 때 동작 가능한 속도) 동작이 가능합니다.

게임 서버를 실행하는 용도로는 각각 한개 혹은 두개의 기본 프로세서 소켓에 대해 독점적인 액세스가 가능한 c4.4xlarge, c4.8xlarge 를 권장합니다. 독점적인 액세스를 통해 대부분의 작업 부하에 대해 3.2GHz 의 모든 코어 터보 모드를 사용할 수 있습니다. 다만 AVX (Advanced Vector Extension)[3]를 실행하는 응용 프로그램의 경우는 예외입니다. c4.8xlarge 인스턴스에서 AVX 위주 작업을 실행하는 경우 최대로 기대 가능한 코어 동작속도는 3 코어 이하를 사용할 경우, 대부분 3.1GHz 입니다. 달성 가능한 성능을 확인하기 위해서는 실제 워크로드를 가지고 테스트를 하는 것이 중요합니다.

다음 표에는 AVX 및 비 AVX 작업 부하에 대한 c4.4xlarge 인스턴스와 c4.8xlarge 인스턴스의 성능 비교입니다.

C4 인스턴스 크기 및 작업 부하	최대 Core Turbo 동작속도 (GHz)	모든 Core Turbo 동작속도 (GHz)	기본 동작속도 (GHz)
C4.8xlarge – AVX 비사용시 작업부하	3.5 (4 vCPU 미만의 코어가 동작중일 때)	3.2	2.9
C4.8xlarge – AVX 작업부하	≤ 3.3	≤ 3.1 (작업부하 및 동작중인 코어 수에 따라 다름)	2.5
C4.4xlarge – AVX 비사용시 작업부하	3.2	3.2	2.9
C4.4xlarge – AVX 작업부하	3.2	≤ 3.1 (작업부하 및 동작중인 코어 수에 따라 다름)	2.5

다른 인스턴스 유형 선택

예를 들어, 롤플레잉 게임(RPG)과 멀티 플레이어 온라인 배틀 아레나 게임(MOBAs)에서는 상황에 따라 게임 서버의 성능이 연산 속도보다 메모리 사용에 더 의존적일 수 있습니다. 이러한 경우, 메모리 대 vCPU 비율이 높은 M4 인스턴스 유형이 C4 인스턴스 유형보다 더 나은 선택일 수 있습니다. 연산 최적화 인스턴스 제품군은 다른 인스턴스 제품군보다 높은 vCPU 대 메모리 비율을 가지는 반면 M4 인스턴스는 높은 메모리 대 vCPU 비율을 제공합니다. M4 인스턴스는 m4.10xlarge 및 m4.16xlarge 의 경우 Haswell 프로세서를 사용하고, 더 작은 사이즈의 인스턴스들은 Broadwell 또는 Haswell 프로세서를 사용합니다. M4 인스턴스 유형은 네트워킹 성능에서는 C4 인스턴스 유형과 유사하며 게임 서버에 충분한 대역폭을 제공합니다.

성능 최적화

Linux 서버에는 가장 중요한 네트워킹과 CPU 의 성능 옵션 외에도 많은 성능 옵션이 있습니다. 이 절에서는 AWS 의 게임산업 고객들이 지금까지 가상 컴퓨터(VM)에서 게임 서버를 실행하면서 알게 된 가장 가치 있고 적합한 성능 옵션들을 기술합니다.

앞으로 기술할 성능 옵션은 네트워킹, CPU, 메모리 및 디스크의 네 가지 섹션으로 분류됩니다. 이는 모든 성능 조정 옵션 목록을 기술하지는 않으며, 기술된 모든 옵션이 모든 게임 작업부하에 적합하지는 않습니다. 프로덕션 환경에 이러한 설정을 적용하기 전에 반드시 먼저 테스트하는 것을 강력히 권장합니다.

이 절에서는 하드웨어 가상 머신(HVM)용 Amazon Machine Image (AMI)[4]를 사용하여 인스턴스를 생성하고 Amazon VPC (Virtual Private Cloud)[5]로 작성된 VPC 에서 인스턴스를 실행한다고 가정합니다. 이 절의 지침과 설정은 모두 4.4.23-31.54 커널을 사용하는 Amazon Linux AMI 2016.09 에서 검증되었지만 이후의 모든 Amazon Linux 배포판에서도 동작할 것입니다.

네트워킹

네트워킹은 성능 조정을 위한 가장 중요한 영역 중 하나입니다. 멀티 플레이어 클라이언트 / 서버 게임들은 지연시간과 패킷 손실에 매우 민감합니다. 네트워킹을 위한 성능 조정 옵션 목록은 다음 표에 기술되어 있습니다.

성능조정 옵션	개요	비고	링크 및 명령어
플레이어와 가까운 곳에 게임 서버 배포	플레이어와의 근접성은 지연시간 감소에 가장 좋은 방법.	AWS 는 전 세계에 걸쳐 수많은 지역에서 서비스 중.	AWS 리전 리스트
향상된 네트워킹	네트워킹 성능 개선	거의 모든 작업부하가 부정적인 효과 없이 성능 개선 효과.	Linux/Windows
UDP			
수신 버퍼	패킷 손실 방지에 도움	클라이언트와 서버 간 지연시간이 높을 때 유용. 부정적인 효과는 거의 없으나 테스트는 반드시 필요	/etc/sysctl.conf 에 설정: net.core.rmem_default = New_Value net.core.rmem_max = New_Value (현재 시스템 설정값의 두배 값으로 시작하는 것을 추천)
Busy polling	수신 패킷 처리 지연시간 감소	CPU 사용량이 증가할 수 있음.	/etc/sysctl.conf 에 설정: net.core.busy_read = New_Value net.core.busy_poll = New_Value (설정값 50 으로 우선 테스트하고 100 으로 다시 테스트해 보는 것을 추천)
UDP 버퍼 메모리	패킷 손실 방지에 도움		/etc/sysctl.conf 에 설정: net.ipv4.udp_mem = New_Value New_Value New_Value

성능조정 옵션	개요	비고	링크 및 명령어
			(현재 시스템 설정값의 두배 값으로 시작하는 것을 추천)
Backlog	패킷 손실 방지에 도움		/etc/sysctl.conf 에 설정: net.core.netdev_max_backlog= New_Value (현재 시스템 설정값의 두배 값으로 시작하는 것을 추천)
송신/수신 대기열	하이퍼스레드 비활성화로 성능 향상될 수 있음		

다음 권장사항들은 지연시간 감소, 패킷 손실 방지 및 게임 서버를 위한 최적의 네트워킹 성능을 얻는 방법에 대해 다룹니다.

플레이어와 가까운 곳에 게임 서버 배포

플레이어와 가능한 가깝게 게임 서버를 배치하는 것은 좋은 플레이어 경험을 위한 핵심 요소입니다. AWS 는 전 세계에 걸쳐 수많은 서비스 지역을 보유하고 있어 플레이어와 가까운 곳에 게임 서버를 배포할 수 있습니다. AWS 지역 및 가용 영역의 최신 목록은 https://aws.amazon.com/about-aws/global-infrastructure/[6]을 참조하십시오.

당신만의 인스턴스 AMI(Amazon Machine Image)[7]를 패키지로 만들어 원하는 만큼 많은 지역에 배포할 수 있습니다. AWS 상에서 서비스되는 AAA PC / 콘솔 게임 고객들의 경우 거의 모든 서비스 가능 지역을 사용하는 것이 자주 관찰됩니다. 플레이어가 세계 어디에 있는지 파악하면 가능한 최고의 경험을 제공하기 위해 게임 서버를 어디에 배포할지 결정할 수 있습니다.

향상된 네트워킹

향상된 네트워킹[8] 은 또 다른 성능 튜닝 옵션입니다. 향상된 네트워킹은 단일 루트 I/O 가상화 (SR-IOV) [9]를 사용하여 하이퍼바이저를 거치지 않고 네트워크 카드를 인스턴스에 직접 노출합니다. 이는 일반적으로 더 높은 I/O 성능, 더 낮은 CPU 사용량,더 높은 PPS (초당 패킷 수) 성능 향상, 짧은 인스턴스간 지연 및 매우 낮은 네트워크 지터(jitter)가 가능하게 합니다. 향상된 네트워킹으로 인한 성능 향상은 멀티 플레이어 게임 서버에 큰 차이를 가져올 수 있습니다.

향상된 네트워킹은 HVM AMI 를 사용하고 VPC 안에서 실행되는 C4, R4, R3, I3, I2, M4 및 D2 와 같은 특정 인스턴스 유형에서만 사용할 수 있습니다. 이러한 인스턴스 유형은 "ixgbevf" Linux 드라이버를 사용하는 가상함수 인터페이스(Intel 82599 Virtual Function Interface)를 사용합니다. 추가적으로 X1, R4, P2 및 M4.16xlarge 인스턴스(그리고 곧 출시될 C5 인스턴스)의 경우는 ENA(Elastic Network Adapter)를 사용하여 향상된 네트워킹 기능을 지원합니다.

Amazon Linux AMI 에는 기본적으로 필요한 드라이버가 포함되어 있습니다. 다른 AMI 들의 경우에는 Linux [10], 또는 Windows [11] 지침에 따라 드라이버를 설치하십시오. 드라이버 설치시에는 Intel 웹사이트 [12] 에서 최신 ixgbevf 드라이버를 다운로드하는 것이 중요합니다. ixgbevf 드라이버의 최소 권장 버전은 2.14.2 입니다.

인스턴스에서 실행중인 드라이버 버전을 확인하려면 다음 명령을 실행하십시오:

```
ethtool -i eth0
```

User Datagram Protocol (UDP)

대부분의 1 인칭 슈팅 게임 및 다른 유사한 클라이언트/서버 멀티 플레이어 게임들은 클라이언트와 게임 서버 사이의 통신 프로토콜로 UDP 를 사용합니다. 다음 절에서는 성능을 향상시키고 패킷 손실의 발생을 줄일 수 있는 네 가지 UDP 최적화를 설명합니다.

수신 버퍼

첫번째 UDP 최적화는 수신 버퍼의 기본값을 늘리는 것입니다. UDP 버퍼 공간이 너무 작으면 운영 체제 커널이 UDP 패킷을 버려 패킷 손실을 초래할 수 있습니다. 이 버퍼 공간을 늘리면 클라이언트와 서버 사의 지연시간이 긴 경우에 도움이 됩니다. Amazon Linux 에서 rmem_default 및 rmem_max 의 기본값은 212992 입니다.

시스템의 현재 기본값을 확인하려면 다음 명령을 실행하십시오.:

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

적절한 양의 버퍼 공간을 할당하는 일반적인 방법은 먼저 두 값을 두 배로 늘린 다음 게임 서버의 성능 차이를 테스트하는 것입니다. 테스트 결과에 따라 이러한 값을 줄이거나 늘려야 할 수 있습니다. rmem_default 값은 rmem_max 값을 초과하지 않아야 합니다.

이 매개변수가 재부팅된 후에도 지속되도록 설정하려면 /etc/sysctl.conf 파일에 새로운 rmem_default 및 rmem_max 값을 다음과 같이 설정하십시오.:

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

sysctl.conf 설정값을 시스템에 갱신하기 위해서는 파일을 변경할 때마다 다음 명령을 실행하여야 합니다.:

```
sudo sysctl -p
```

Busy Polling

두 번째 UDP 최적화는 busy polling 으로, 커널이 들어오는 패킷을 폴링하여 네트워크 수신 경로 지연을 줄일 수 있습니다. 이는 CPU 사용률은 증가시키지만 패킷 처리 지연을 줄일 수 있습니다.

Amazon Linux 를 포함하여 대부분의 Linux 배포판에서는 busy polling 이 기본적으로 비활성화되어 있습니다. busy read 와 busy poll 값 모두 50 으로 설정하여 게임 서버에서 어떤 차이가 있는지 테스트해보는 것을 추천합니다. Busy read 는 소켓에서 패킷을 읽기 위해 장치 대기열에서 대기하는 시간(마이크로 초)이며 busy poll 은 소켓 폴링 및 선택을 위해 장치 대기열에서 대기하는 시간(마이크로 초)입니다. 결과에 따라 설정값을 100 으로 늘려야 할 수도 있습니다.

재부팅시에도 이 매개 변수 설정값이 유지되도록 설정하려면 새 busy_read 및 busy_poll 값을 /etc/sysctl.conf 파일에 다음과 같이 추가합니다.:

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

이 경우에도 sysctl.conf 설정값을 시스템에 갱신하기 위해서는 파일을 변경할 때마다 다음 명령을 실행하여야 합니다.:

```
sudo sysctl -p
```

UDP 버퍼

세번째 UDP 최적화는 UDP 버퍼가 대기열에 사용하는 메모리 크기를 변경하는 것입니다. udp_mem 옵션은 UDP 소켓이 대기열에 사용할 수 있는 페이지 수를 설정합니다. 이는 네트워크 어댑터의 사용량이 매우 많을 때 발생할 수 있는 패킷 손실을 줄이는데 도움이 됩니다.

이 설정은 페이지 단위(4096 바이트)의 세 값의 벡터로 설정합니다. 첫번째 min 값은 UDP 가 메모리 사용을 시작하는 최소 임계 값입니다. 두번째 pressure 값은 UDP 가 메모리 소모를 조정하는 메모리 임계값입니다. 마지막 max 값은 모든 UDP 소켓에서 대기열에 사용할 수 있는 최대 페이지 값입니다. 기본적으로 c4.8xlarge 인스턴스에서 구동되는 Amazon Linux 는 1445727 1927636 2891454 의 벡터를 사용하고 c4.4xlarge 인스턴스에서는 720660 960882 1441320 의 벡터를 사용합니다.

시스템의 현재 기본값을 확인하려면 다음 명령을 실행하십시오.:

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

이 설정의 새로운 값을 실험할 때 설정값을 기본값의 두 배로 늘린 다음 게임 서버의 차이점을 테스트하는 것으로 시작하는 것이 좋습니다. 값을 조정할 때에는 페이지 크기 (4096 바이트)의 배수가 되도록 하는 것이 좋습니다. 재부팅시에도 이 설정값이 유지되도록 설정하려면 /etc/sysctl.conf 파일에 새 UDP 버퍼값을 다음과 같이 추가하십시오.:

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

sysctl.conf 설정값을 변경한 후에 이를 시스템에 갱신하기 위해서는 다음 명령을 실행하십시오.:

```
sudo sysctl -p
```

Backlog

패킷 손실의 가능성을 줄이는 데 도움이 되는 마지막 UDP 최적화는 backlog 값을 증가시키는 것입니다. 이 최적화는 네트워크 인터페이스가 커널이 처리할 수 있는 것보다 빠른 속도로 패킷을 수신하는 상황에서 들어오는 패킷의 대기열 크기를 증가시킵니다. Amazon Linux 에서 대기열 크기의 기본값은 1000 입니다.

시스템의 현재 기본값을 확인하려면 다음 명령을 실행하십시오.:

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

여기에서도 시스템의 기본값을 두 배로 늘린 다음 게임 서버의 차이점을 테스트하는 것을 추천합니다. 재부팅시에도 이 설정값을 유지하도록 설정하려면 /etc/sysctl.conf 파일에 새 backlog 값을 다음과 같이 추가하십시오.:

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

sysctl.conf 설정값을 변경한 후에 이를 시스템에 갱신하기 위해서는 다음 명령을 실행하십시오.:

```
sudo sysctl -p
```

송신 및 수신 대기열

많은 게임 서버에서 전체 대역폭 사용량보다 초당 처리 패킷수를 통해 네트워크에 부하가 걸립니다. 또한 vCPU 중 하나가 많은 양의 인터럽트 요청(IRQ)을 받으면 I/O 대기가 병목이 될 수 있습니다.

RSS(Receive Side Scaling) [13] 는 이러한 네트워킹 성능 문제를 해결하는 데 사용되는 일반적인 방법입니다. RSS 는 네트워크 인터페이스 컨트롤러 (NIC)에서 복수의 수신대기열을 제공할 수 있는 하드웨어 옵션입니다. Amazon EC2 에서 NIC 는 ENI(Elastic Network Interface) [14] 라고 합니다. RSS 는 C4 인스턴스 제품군에서 활성화되어 있지만 RSS 설정 변경은 허용되지 않습니다. C4

인스턴스 제품군은 Linux 를 사용할 경우 모든 인스턴스 크기에 대해 두 개의 수신 대기열을 제공합니다. 이 두개의 대기열에는 각각 별도의 IRQ 번호가 있으며 별도의 vCPU 에 매핑됩니다.

c4.8xlarge 인스턴스에서 $ ls -1 /sys/class/net/eth0/queues 명령을 실행하면 다음과 같이 대기열이 표시됩니다. :

```
$ 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
```

어느 IRQ 가 어느 대기열에서 사용되고 있고 CPU 가 해당 인터럽트들을 어떻게 처리하고 있는지 확인하기 위해서는 다음 명령어를 실행합니다.:

```
cat /proc/interrupts
```

다른 방법으로, 대기열들의 IRQ 만을 확인할 때에는 다음 명령어를 사용할 수 있습니다. :

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

다음은 c4.8xlarge 에서 /proc/interrupts 의 전체 내용 중 eth0 인터럽트만을 출력하도록 한 결과값입니다. 첫번째 컬럼은 각 대기열의 IRQ 입니다. 마지막 두 컬럼은 프로세스 정보입니다. 이 경우, TxRx-0 과 TxRx-1 대기열이 각각 IRQ 267 과 268 을 사용하고 있는 것을 알 수 있습니다.

```
     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
```

대기열이 어느 vCPU 로 인터럽트를 보내고 있는지 확인하기 위해서는 다음 명령어를 실행합니다. (IRQ_Number 를 각 TxRx 대기열의 IRQ 번호로 대치하여 실행하여야 합니다.)

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

앞의 결과는 c4.8xlarge 인스턴스에서 실행한 결과값입니다. Hex 값으로 출력되므로 vCPU 번호를 확인하기 위해서는 이진값으로 변환하여야 합니다. 예를 들어, hex 값 00800000 을 이진값으로 변환하면 00000000100000000000000000000000 이고, 오른쪽에서부터 CPU0 을 나타내므로 vCPU 23 번을 사용중임을 알 수 있습니다. 같은 방법으로 다른 대기열은 vCPU 33 번을 사용중임을 알 수 있습니다.

vCPU 23 번과 33 번은 각각 다른 프로세서 소켓에 속하기 때문에 이들은 물리적으로 서로 다른 NUMA(non-uniform memory access) 노드에 위치하게 됩니다. 여기서 각 vCPU 가 기본적으로는 물리코어가 아닌 하이퍼스레드라는 문제가 있습니다. (이 경우만 놓고 보자면 두 vCPU 는 같은 코어의 하이퍼스레드들입니다.) 그러므로 각 대기열을 한 물리코어에 할당하여 성능을 향상시킬 수 있습니다.

C4 인스턴스 제품군에서 구동되는 Amazon Linux 에서는 이 두 대기열의 IRQ 들이 이미 분리된 특정 vCPU 에 고정되어 있고 C4.8xlarge 인스턴스의 경우 이 vCPU 는 개별 NUMA 노드에 위치해 있습니다. 이 기본 설정은 게임서버를 실행하는데 이상적일 것입니다. 하지만, 다른 Linux 배포판을 사용할 경우, 두개의 대기열이 각각의 NUMA 노드에 위치한 vCPU 및 IRQ 로 설정되어 있는지 반드시 확인해봐야 합니다. C4.8xlarge 외 다른 C4 인스턴스들의 경우 하나의 NUMA 노드만 가지고 있기 때문에 NUMA 는 문제가 되지 않습니다.

RSS 의 성능을 향상시킬 수 있는 한 가지 옵션은 하이퍼스레딩을 비활성화하는 것입니다. Amazon Linux 에서 하이퍼스레딩을 비활성화한 경우 기본적으로 대기열은 (c4.8xlarge 인스턴스의 경우 개별 NUMA 노드에 위치한) 물리 코어들에 고정됩니다. 하이퍼스레딩을 비활성화하는 방법에 대한 자세한 내용은 이 백서의 하이퍼스레딩 단원을 참조하십시오.

만약 게임 서버 프로세스가 특정 코어에 고정되지 않는 경우, Linux 스케줄러가 게임 서버 프로세스를 RSS 대기열이 사용하는 vCPU (또는 코어)에 할당하지 못하도록 설정할 수 있습니다. 이렇게 하려면 두 가지 옵션을 설정해야 합니다.

먼저, 텍스트 편집기에서 /boot/grub/grub.conf 파일을 편집합니다. "kernel"로 시작하는 첫번째 항목(두 개 이상의 커널 항목이 있을 수 있지만 첫번째 항목 만 편집하면 됩니다)의 끝 부분에 isolcpus=NUMBER 를 추가합니다. 여기서 NUMBER 는 RSS 대기열이 사용하는 vCPU 의 번호입니다. 예를 들어 RSS 대기열이 vCPU 3 과 4 를 사용하는 경우 NUMBER 를 "3-4"로 바꿉니다.

```
# 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
```

isolcpus 를 사용하면 스케줄러가 지정한 vCPU 에서 게임 서버 프로세스를 실행하지 못하게 됩니다. 문제는 이 설정으로 인해 irqbalance 도 이렇게 설정한 vCPU 에 IRQ 를 할당하지 못하게 된다는 것입니다. 이 문제를 해결하려면 IRQBALANCE_BANNED_CPUS 옵션을 사용하여 나머지 모든 CPU 들이 IRQ 를 할당받지 못하도록 설정해야 합니다. 현재 버전의 Amazon Linux 에서 구동되는 irqbalance 버전 1.1.10 이상은 IRQBALANCE_BANNED_CPUS 설정을 우선하므로 IRQBALANCE_BANNED_CPUS 로 지정된 vCPU 를 사용하지 않기 위해 isolcpus 에 지정된 vCPU 에 IRQ 를 할당하게 됩니다. 그러므로 예를 들어 isolcpus 를 사용하여 vCPU 3-4 를 격리한 경우 IRQBALANCE_BANNED_CPUS 설정을 사용하여 인스턴스의 다른 vCPU 들이 IRQ 를 할당받지 못하도록 설정할 필요가 있습니다.

이렇게 하려면 /etc/sysconfig/irqbalance 파일에서 IRQBALANCE_BANNED_CPUS 옵션을 설정해야 합니다. 이 값은 64 비트 16 진수 bitmask 입니다. 올바른 값을 찾는 가장 좋은 방법은 이 값에 포함시키려는 vCPU 들을 10 진수 형식으로 먼저 준비한 다음 16 진수로 변환하는 것입니다. 이전 예제에서 isolcpus 를 사용하여 vCPU 3-4 를 제외시켰으므로,

IRQBALANCE_BANNED_CPUS 에는 vCPU 1, 2 및 5-14 (c4.4xlarge 인스턴스를 사용중이라고 가정)를 설정해야 합니다. 따라서 bitmask 로는 1111111111100111 이 되고, 이 값을 1 마지막으로 16 진수로 변환하면 FFE7 이 됩니다. 원하는 편집기를 사용하여 다음 행을 /etc/sysconfig/irqbalance 파일에 추가하십시오.:

```
IRQBALANCE_BANNED_CPUS="FFE7n"
```

결과적으로 vCPU 3 및 4 는 게임 서버 프로세스에서 사용되지 않지만 RSS 대기열과 시스템에서 사용하는 몇 가지 다른 IRQ 에서 사용됩니다.

다른 모든 설정들과 마찬가지로, 이러한 모든 값은 실제로 성능 차이를 보이는지 판단하기 위해 실제 게임 서버로 테스트해봐야 합니다.

대역폭

C4 인스턴스 제품군은 멀티 플레이어 게임 서버에 충분한 대역폭을 제공합니다. c4.4xlarge 인스턴스의 경우 높은 네트워크 성능을 제공하며, 향상된 네트워킹을 통해 동일한 배치 그룹[15] 에 있는 두 개의 c4.8xlarge 인스턴스 (혹은 m4.10xlarge 와 같은 다른 대형 인스턴스) 사이에서 최대 10Gbps 를 달성 할 수 있습니다. c4.4xlarge 와 c4.8xlarge 인스턴스는 우리가 알고 있는 모든 게임 서버 사례들에서 충분한 대역폭을 제공했습니다.

C4 인스턴스에서 실행되는 실제 작업부하의 네트워크 성능을 동일한 가용영역의 다른 인스턴스나 다른 가용영역의 다른 인스턴스, 인터넷으로부터 쉽게 테스트하고 비교하여 결정할 수 있습니다. Iperf[16]는 Linux 에서 네트워크 성능을 테스트하고 결정하는데 가장 좋은 도구 중 하나이고. Nttcp[17]는 Windows 에서 사용가능한 좋은 도구입니다. 위 링크에서는 네트워크 성능 테스트에 대한 지침도 제공하고 있습니다. 배치 그룹 외부에서부터 Iperf 또는 Nttcp 와 같은 도구를 사용하여 게임 서버에서 달성가능한 정확한 네트워크 성능을 테스트하고 결정해야 합니다.

CPU

CPU 는 네트워킹과 함께 게임 서버에서 가장 중요한 두 가지 성능 조정 영역 중 하나입니다.

성능 조정 옵션	개요	비고	링크 및 명령어
클럭 소스	클럭소스로 tsc 를 사용하여 게임서버 성능 개선 가능	Xen 이 Amazon Linux 이 기본 클럭소스임.	/boot/grub/grub.conf 파일 커널 항목에 다음 항목 추가: tsc=reliable clocksource=tsc
C-State 와 P-State	C-State 및 P-State 옵션들은 c4.8xlarge 인스턴스의 C-State 를 제외하고 기본적으로 최적화되어 있음. C4.8xlarge 인스턴스의 경우 C-State 를 C1 으로 설정하여 CPU 성능개선 가능	C4.8xlarge 인스턴스에서만 변경 가능. 설정 변경시 3.5GHz 의 최대 터보 모드 지원이 안될 수 있으나, 3.2GHz 의 모든 코어 터보 모드 동작이 가능해짐.	/boot/grub/grub.conf 파일의 커널 항목에 다음 항목 추가: intel_idle.max_cstate=1
Irqbalance	게임 서버를 특정 vCPU 에 고정하지 않을 경우 irqbalance 를 사용하면 CPU 성능 개선 가능	Amazon Linux 에서는 기본적으로 설치되어 실행중. 다른 배포판을 사용할 경우 설치 및 동작중인지 확인 필요.	NA
하이퍼스레딩	각 vCPU 는 물리코어의 하이퍼스레드임. 하이퍼스레딩 비활성화로 성능이 개선될 수 있음		/boot/grub/grub.conf 파일의 커널 항목에 다음 항목 추가: Maxcpus=X (X 는 사용중인 인스턴스의 실제 물리코어 수)
CPU 고정	특정상황에서는 게임 서버 프로세스를 vCPU 에 고정하는 것이 성능상 이득을 얻을 수 있음	게임 산업에서는 CPU 고정이 일반적인 용례는 아님.	"numactl --physcpubind $phys_cpu_core --membind $associated_numa_node ./game_server_executable"
Linux 스케줄러	게임서버 성능에 도움이 될 수 있는 세가지 Linux 스케줄러 설정 옵션이 있음		sudo sysctl -w 'kernel.sched_min_granularity_ns=New_Value'

성능 조정 옵션	개요	비고	링크 및 명령어
			(시스템 현재값의 두배 설정으로 시작하는 것을 추천) sudo sysctl -w 'kernel.sched_wakeup_granularity_ns=New_Value' sudo sysctril –w (시스템 현재값으로부터 시작하는 것을 추천) 'kernel.sched_migration_cost_ns=New_Value' (시스템 현재값의 두배 설정으로 시작하는 것을 추천)

클럭 소스

클럭 소스는 Linux 에서 타임라인에 액세스 할 수 있도록 하여 프로세스가 현재의 시간정보를 파악할 수 있도록 해줍니다. 멀티 플레이어 게임서버에서는 서버가 모든 이벤트들을 최종적으로 처리하는 원천이고 각 클라이언트가 자체의 시간과 흐름으로 이 이벤트들을 보여주게 되므로 시간정보가 매우 중요합니다. kernel.org 웹사이트에 클럭 소스에 대한 좋은 설명이 게재되어 있습니다. [18]

현재의 클럭소스 설정을 확인하려면 다음 명령을 실행합니다.:

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

기본적으로 C4 인스턴스에서 구동되는 Amazon Linux 의 경우 이것이 xen 으로 설정되어 있습니다.

설정가능한 모든 클럭소스를 확인하려면 다음 명령을 실행합니다.:

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

Amazon Linux 가 실행되는 C4 인스턴스에서는 기본적으로 xen, tsc, hpet 및 acpi_pm 이 표시되어야 합니다. 대부분의 게임 서버의 경우 각 프로세서의 64 비트 레지스터 인 TSC (Time Stamp Counter)가 클럭소스로 가장 적합합니다. 대부분의 경우, TSC 가 가장 빠르고 가장 정밀한 시간 경과에 대한 측정값 이고 단조함수이며 불변입니다. XEN 가상화와 관련하여 TSC 에 대한 좋은 토론이 게재된 xen.org 문서를 참조하십시오.[19] 어느 power state 에 있던 모든 프로세서에 동기화가 제공되므로 TSC 는 동기화되고 불변인 것으로 간주됩니다. 이것은 TSC 가 항상 일정한 비율로 증가한다는 것을 의미합니다.

TSC 는 rdtsc 또는 rdtscp 명령어를 사용하여 액세스 할 수 있습니다. rdtscp 는 Intel 프로세서가 비순차적 명령어 처리를 하는 경우가 있다는 것을 고려하고 있으므로 rdtsc 보다 rdtscp 가 보통 더 나은 옵션입니다.

게임 서버에서는 클럭 소스를 TSC 로 변경하는 것을 권장합니다만, 실제 작업 부하에 대해 클럭 소스를 변경한 후 전체적으로 테스트해봐야 합니다. 클럭 소스를 TSC 로 설정하려면 원하는 편집기로 /boot/grub/grub.conf 파일을 수정해야 합니다. "kernel"로 시작하는 첫번째 항목(두 개 이상의 kernel 항목이 있을 수 있지만 첫번째 항목만 수정하면 됩니다)의 끝부분에 tsc=reliable clocksource=tsc 를 추가하십시오.

```
# 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
```

프로세서 상태 제어(C-States 및 P-States)

프로세서 상태 제어는 C4 인스턴스 제품군에서는 c4.8xlarge 인스턴스에서만 가능합니다. (다른 인스턴스 제품군에서는 d2.8xlarge, m4.10xlarge 및 x1.32xlarge 인스턴스에서 설정 가능)[20] C-State 는 코어가 유휴 상태일 때 들어갈 수 있는 슬립 레벨을 제어하고 P-State 는 코어가 동작중일 때 원하는 성능(CPU 동작 클럭)을 제어합니다. C-State 는 유휴시 절전 상태이고 P-State 는 실행시 절전 상태입니다.

C-State 는 코어가 실제로 기능을 수행하는 가장 얕은 상태인 C0 에서 시작하여 코어가 본질적으로 꺼지는 가장 깊은 상태인 C6 까지 존재합니다. c4.8xlarge 인스턴스의 경우 기본 C-State 는 C6 입니다. C4 제품군의 c4.8xlarge 외 다른 인스턴스 크기의 경우 기본값은 C1 입니다. 이것이 3.5GHz 최대 터보 모드가 c4.8xlarge 인스턴스에서만 사용가능한 이유입니다. 코어가 3.5GHz 에 도달하려면 일부 vCPU 가 C1 보다 더 깊은 슬립 상태여야 합니다.

c4.8xlarge 인스턴스에서 사용가능한 조정 옵션은 코어가 휴면상태가 되지 않도록 가장 깊은 C-State 를 C6 가 아닌 C1 으로 설정하는 것입니다. 이 설정은 프로세서 반응 지연시간을 줄여주지만 또한 몇 개의 코어만 활성화되어 있을 때 도달할 수 있는 3.5GHz 터보 부스트로 동작하는 것도 막게 됩니다; 하지만 이 경우에도 여전히 3.2GHz 의 모든 코어 터보 모드는 사용 가능합니다. 따라서 프로세서의 반응 지연을 줄이는 것과 몇 개의 코어만 동작 중일 때 해당 코어들이 3.5GHz 의 최대 터보 모드로 동작하는 것 중에서 선택해야 합니다. 선택은 실제 Application 작업부하의 테스트 결과에 따라 다릅니다. 3.2GHz 의 모든 코어 터보 모드가 납득할 수 있는 수준의 성능이고 C4.8xlarge 인스턴스에서 대부분의 시간 동안 모든 코어 또는 대부분의 코어를 사용하게 될 것이라면 C-State 를 C1 으로 변경하십시오.

P-State 는 터보 모드가 활성화되는 P0 에서 시작하여 동작 가능한 최저 클럭 모드를 나타내는 P15 까지 존재합니다. P0 가 최대치의 동작속도를 제공합니다. 모든 C4 인스턴스 크기의 기본 P-State 는 P0 입니다. 게임 작업부하의 경우 이 설정을 변경할 이유는 없습니다. 이미 설정된 터보 부스트 모드가 바람직한 상태입니다.

다음 표는 c4.4xlarge 와 c4.8xlarge 의 C-State 와 P-State 를 설명합니다.

	기본 최대 C-State	추천 설정	기본 P-State	추천 설정
c4.4xlarge 및 더 작은 인스턴스	1	1	0	0
c4.8xlarge	6[a]	1	0	0

a) cat /sys/module/intel_idle/parameters/max_cstate 명령어를 실행하면 최대 C-state 가 9 인 것으로 표시되지만 실제로는 가능한 최대 값인 6 으로 설정됩니다.

turbostat 을 사용하여 c4.8xlarge 인스턴스에서 얻을 수 있는 C-State 및 최대 터보 모드 클럭을 확인할 수 있습니다. 다시 한번 언급하지만, 이 지침은 Amazon Linux AMI 를 사용하여 테스트되었으며 c4.8xlarge 인스턴스에서만 동작하고 C4 제품군의 다른 인스턴스 크기에서는 동작하지 않습니다.

먼저 다음 명령을 실행하여 시스템에 stress 를 설치합니다. (turbostat 이 시스템에 설치되어 있지 않은 경우 turbostat 도 설치합니다.)

```
sudo yum install stress
```

다음 명령은 2 개의 코어 (즉, 2 개의 서로 다른 물리적 코어의 2 개의 하이퍼스레드)에 스트레스를 인가합니다.

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

다음은 명령을 실행한 결과의 일부입니다.

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		

정의:

AVG_MHz: 실행 주기를 경과시간으로 나눈 평균 동작 클럭

%Busy: "C0" state 에 들어가 있던 비율.

Bzy_MHz: CPU 가 동작중("c0 state)이었던 동안 평균 동작 클럭

TSC_MHz: 전체 실행시간중 TSC 가 동작한 평균 동작 클럭

위 결과값은 vCPU 9 와 20 이 대부분의 시간 동안 C0 상태(%Busy)로 동작했고 3.5GHz (Bzy_MHz)의 최대 터보모드에 근접하여 동작한 것을 보여줍니다. 이 코어의 다른 하이퍼스레드인 vCPU 2 와 27 은 명령을 기다리는 상태인 C1 상태(CPU%c1)에 머물러 있음을 보여줍니다. c4.8xlarge 인스턴스의 기본 C-State 가 C6 이고 대부분의 코어가 C6 상태 (CPU%c6)에 있었기 때문에 3.5GHz 에 가까운 동작속도가 달성되었음을 알 수 있습니다.

다음으로 3.2GHz 의 모든 코어 터보 모드를 확인하기 위해 36 개의 vCPU 전부에 스트레스를 인가해 봅니다:

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

다음은 위 명령어를 실행한 결과값의 일부입니다.:

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		

모든 vCPU 가 99 %이상의 시간 동안 C0(%Busy)상태에 있고, C0 상태에 있는 동안 모두 3.2GHz(Bzy_MHz)으로 동작하고 있음을 볼 수 있습니다.

C-State 를 C1 로 설정하려면 텍스트 편집기로 /boot/grub/grub.conf 파일을 편집해야 합니다. "kernel"으로 시작하는 첫번째 항목(두개 이상의 커널 항목이 있을 수 있지만 첫 번째 항목만 편집하면 됩니다)의 줄 끝 부분에 intel idle.max cstate=1 을 추가하여 유휴 상태의 코어가 도달할 수 있는 가장 깊은 휴면상태의 CS-State 를 C1 으로 설정합니다.:

```
# 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
```

파일을 저장하고 편집기를 종료하십시오. 새로운 커널 옵션을 반영하기 위해 인스턴스를 재부팅하십시오. 이제 turbostat 명령을 다시 실행하여 C-State 를 C1 으로 설정한 후 변경된 사항을 확인합니다.

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

다음은 위 명령어를 실행한 결과값의 일부입니다.:

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		

위 결과는 모든 코어가 현재 C1 C-State 에 있음을 보여줍니다. 부하를 인가받은 두 vCPU - 위의 예에서는 vCPU 16 및 2 - 의 최대 평균 동작 클럭은 3.2GHz (Bzy_MHz)입니다. 모든 vCPU 가 C1 상태에 있으므로 3.5GHz 의 최대 터보 모드는 더 이상 사용할 수 없습니다.

C- State 가 C1 으로 설정되어 있는지 확인하는 또 다른 방법은 다음 명령을 실행하는 것입니다.:

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

마지막으로, 코어가 C6 상태에서 C1 상태로 전환될 때 어떤 성능상의 비용이 무엇인지 궁금할 것입니다. cpuidle 파일을 조회하여 다양한 C-State 에 대한 종료 지연 시간을 마이크로 초 단위로 확인할 수 있습니다. CPU 의 C-State 상태가 전환될 때마다 지연 시간이 발생합니다.

예를 들어 기본 C-State 설정상에서 cpuidle 은 C6 상태에서 C0 상태로 전환시 133 마이크로 초의 시간이 필요함을 알 수 있습니다.:

```
$ 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
```

C- State 기본값을 C1 상태로 변경하면 CPU 유휴상태 간의 차이를 볼 수 있습니다. 이제 C1 상태에서 C0 상태로 전환하는 데 2 마이크로 초 밖에 걸리지 않습니다. vCPU 를 C1 상태로 설정하면 상태 전환 지연시간을 133 마이크로초에서 2 마이크로초로 131 마이크로 초 줄일 수 있습니다.

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

위의 지침은 c4.8xlarge 인스턴스에만 유효합니다. c4.4xlarge 인스턴스 (및 C4 제품군의 더 작은 크기의 인스턴스)의 경우 C-State 는 이미 C1 상태로 설정되어 있으므로 모든 코어 터보 3.2GHz 는 기본 설정으로 사용 가능하여야 합니다. Turbostat 는 프로세서들이 2.9GHz 의 기본 동작 클럭을 넘어서서 동작하는 것을 보여주지 않을 것입니다. 문제는 turbostat 에 디버그 옵션을 사용하더라도 c4.4xlarge 인스턴스에서는 c4.8xlarge 인스턴스에서 위 출력과 같이 Avg_MHz 또는 Bzy_MHz 값을 표시하지 않는다는 것입니다.

c4.4xlarge 인스턴스의 vCPU 들이 3.2GHz 의 모든 코어 터보 모드로 동작하는지 확인하는 한 가지 방법은 Brendan Gregg 의 showboost 스크립트[21]를 사용하는 것입니다. Amazon Linux 에서 이를 사용하려면 msr 도구를 설치해야합니다. 이를 위해 다음 명령을 실행하십시오.:

```
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
```

결과에서 vCPU 0 만 표시되지만 옵션 섹션을 수정하여 표시되는 vCPU 를 변경할 수 있습니다. CPU 동작클럭을 표시하려면 게임 서버를 실행하거나 turbostat stress 를 사용하고 showboost 명령을 실행하여 vCPU 의 동작클럭을 확인합니다.

Irqbalance

Irqbalance 는 성능을 향상시키기 위해 인터럽트들을 시스템의 코어들에 분배하는 서비스입니다. Irqbalance 는 게임 서버를 특정 vCPU 또는 코어에 고정하는 경우를 제외하고는 대부분의 경우에서 권장됩니다. 게임 서버를 특정 vCPU 들이나 코어들에 고정시키는 경우에는 irqbalance 를 비활성화하는 것이 나을 수 있습니다. Irqbalance 의 활성화/비활성화 간 차이점이 있는지 당신의 실제 작업부하로 테스트하십시오. 기본적으로 C4 인스턴스 제품군에서 irqbalance 는 실행되도록 설정되어 있습니다.

인스턴스에서 irqbalance 가 실행 중인지 확인하려면 다음 명령을 실행합니다.:

```
sudo service irqbalance status
```

Irqbalance 는 / etc / sysconfig / irqbalance 파일에서 설정할 수 있습니다.

인터럽트들이 모든 vCPU 에 대해 균등하게 분배되는지 확인하고 싶을 경우, 다음 명령을 실행하여 인터럽트들의 상태들을 확인하고 인터럽트들이 vCPU 들에 제대로 분배되고 있는지 확인할 수 있습니다.:

```
cat /proc/interrupts
```

하이퍼스레딩

C4 인스턴스 제품군의 각 vCPU 는 물리 코어의 하이퍼스레드입니다. 만약 하이퍼스레딩이 당신의 응용 프로그램의 성능에 악영향을 미친다고 판단되면 하이퍼스레딩을 비활성화할 수 있습니다. 그러나 많은 게임 고객들의 사례에서 하이퍼스레딩을 비활성화해야 하는 경우는 거의 없었습니다.

아래 표는 각 C4 인스턴스 크기의 물리 코어 수를 보여줍니다.

인스턴스 크기	vCPU 수	물리 코어 수
c4.large	2	1
c4.xlarge	4	2
c4.2xlarge	8	4
c4.4xlarge	16	8
c4.8xlarge	36	18

다음 명령어를 실행하여 모든 vCPU 의 정보를 확인할 수 있습니다.:

```
cat /proc/cpuinfo
```

더 상세한 정보를 얻기 원한다면 다음 명령어를 사용할 수 있습니다.:

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

이 명령어의 결과값에서 "processor"는 vCPU 번호입니다. "physical id"는 프로세서 소켓 ID 를 나타냅니다. c4.8xlarge 이외의 C4 인스턴스의 경우 하나의 소켓만을 사용하므로 이 값은 0 이됩니다. "core id"는 물리 코어 번호입니다. 동일한 "physical id" 및 "core id"를 가진 각 항목은 동일한 코어의 하이퍼스레드들입니다.

각 코어의 vCPU 쌍 (즉, 하이퍼스레드)을 보는 또 다른 방법은 각 코어에 대한 thread_siblings_list 를 보는 것입니다. 이 정보는 각 코어에 대한 vCPU 번호 두 개를 나타냅니다. 다음 명령어에서 "cpuX"의 X 를 확인하고자 하는 vCPU 의 번호로 변경하십시오.

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

하이퍼스레딩을 비활성화하려면 텍스트 편집기로 /boot/grub/grub.conf 파일을 편집하십시오. "kernel"로 시작하는 첫번째 항목 (두개 이상의 커널 항목이 있을 수 있지만 첫 번째 항목만 편집하면 됨)의 줄 끝에 maxcpus=NUMBER 를 추가합니다. 여기서 NUMBER 는 사용중인 C4 인스턴스의 실제 코어 수입니다. 각 C4 인스턴스 크기별 물리 코어 수에 대해서는 위의 표를 참조하십시오.

```
# 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
```

파일을 저장하고 편집기를 종료하십시오. 새로운 커널 옵션을 적용하려면 인스턴스를 재부팅하십시오.

다시 말하지만, 이들은 게임 성능이 향상되는지 여부를 결정하기 위해 실제 작업부하로 테스트해봐야 하는 설정 중 하나입니다. 이 설정은 성능 향상을 위해 프로세스의 CPU 고정과 함께 설정해야 할 수 있습니다. 실제로, 프로세스의 CPU 고정을 사용하지 않고 하이퍼스레딩을 비활성화하면 성능이 저하될 수 있습니다.

AWS 에서 실행되는 많은 메이저 AAA 게임이 실제로 하이퍼스레딩을 비활성화하지 않습니다. 하이퍼스레딩이 실제로 성능을 향상시키지 않는다면 각 게임 서버에서 이를 관리해야하는 관리 부담을 피하기 위해 이 설정을 하지 않는 것이 좋습니다.

CPU 고정

많은 게임 서버 프로세스들이 보통 주 스레드와 몇 가지 부 스레드로 구성되어 있습니다. 각 게임 서버의 프로세스를 코어(vCPU 또는 물리 코어)에 고정하는 것은 확실히 선택지가 될 수 있지만 자주 보게 되는 구성은 아닙니다. 일반적으로 CPU 고정은 게임 엔진이 정말로 코어에 독점적으로 접근해야 하는 상황에서 설정합니다. 게임 회사들에서는 단순하게 Linux 스케줄러가 이를 처리하도록 하는 경우가 많습니다. 다시 말하지만, 이 설정도 실제 작업부하로 테스트해봐야 할 항목입니다. 하지만 CPU 고정 없이도 충분한 성능이 나온다면 이와 같이 별도 CPU 고정 없이 스케줄러가 처리하도록 하는 것이 CPU 고정에 대한 고려를 해야하는 관리 부담을 줄일 수 있습니다.

이후 NUMA 절에서 다룰 내용이지만 다음 명령을 실행하여 CPU 코어와 NUMA 노드 모두에 프로세스를 고정 할 수 있습니다 (game_server_executable 이름 및 $phys_cpu_core 및 $associated_numa_node 의 값을 실제 값으로 변경하여 사용하십시오.):

```
"numactl –physcpubind $phys_cpu_core –membind $associated_numa_node
./game_server_executable"
```

Linux 스케줄러

기본 Linux 스케줄러는 Complete Fair Scheduler(CFS)[22]라고 하며, CPU 자원 할당을 고려하여 프로세스들을 실행하는 역할을 합니다. CFS 의 기본 목표는 vCPU 의 활용도를 극대화하고 그 결과로 전체적인 성능을 최대화하는 것입니다. 게임 서버 프로세스들을 특정 vCPU 에 고정하지 않을 경우 Linux 스케줄러가 게임 서버 프로세스들의 스레드들을 적절한 vCPU 에 할당하게 됩니다.

게임 서버들에 도움이 되도록 Linux 스케줄러를 조정하기 위한 몇 가지 매개변수들이 있습니다. 아래에 기술한 세가지 매개변수의 주된 목표는 작업의 활동에 따라 가능한 한 프로세서에서 작업들이 수행되도록 유지하는 것입니다. 여기에서는 스케줄러의 최소 세분성, 스케줄러 wakeup 세분성 및 스케줄러 마이그레이션 비용에 중점을 두겠습니다.

모든 kernel.sched 옵션의 기본값을 확인하려면 다음 명령을 실행하십시오.:

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

스케줄러 최소 세분값은 한 작업이 다른 작업으로 대체되기 전까지 CPU 에서 실행되는 시간을 설정합니다. Amazon Linux 로 구동되는 C4 인스턴스 제품군에서는 기본값으로 3ms 로 설정됩니다. 이 값을 늘리면 프로세서에서의 작업을 더 오래 유지할 수 있습니다. 이 설정값을 6ms 로 두배로 설정하는 것이 하나의 선택지가 될 수 있습니다. 이 백서의 다른 모든 성능 권장사항과 마찬가지로 이러한 설정은 실제 게임 서버에서 철저히 테스트해봐야 합니다. 이 명령과 다른 두 스케줄러 명령어는 다시 부팅할 경우 설정 유지되지 않으므로 시작 스크립트에서 실행되도록 설정되어야 합니다.:

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

스케줄러 wakeup 세분값은 실행중인 현재 작업을 대체하기 위하여 작업이 wakeup 되는 기능에 영향을 줍니다. 이 값이 작을수록 작업의 강제 종료가 쉽게 일어납니다. Amazon Linux 로 구동되는 C4 인스턴스 제품군에서 이 값은 기본적으로 4ms 로 설정됩니다. 이 값을 2ms 로 반으로 줄이고 결과를 테스트해보는 것이 선택지 중 하나입니다. 게임 서버의 특성에 따라 더 작은 값으로 설정하면 성능이 더 향상될 수 있습니다.

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

스케줄러 마이그레이션 비용값은 스케줄러가 마이그레이션을 결정할 때 각 작업이 여전히 "캐시 핫 (cache hot)"인지 결정할 때 사용되는 작업의 마지막 실행 후 경과시간을 설정합니다. "캐시 핫"인 작업은 마이그레이션 될 가능성이 적기 때문에 이 설정을 통해 작업이 마이그레이션될 가능성을 줄일 수 있습니다. Amazon Linux 로 구동되는 C4 인스턴스 제품군에서 이 값은 기본적으로 4ms 로 설정됩니다. 이 값을 8ms 로 두 배로 하고 테스트해 보는 것이 하나의 선택지입니다.

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

메모리

c8.8xlarge 인스턴스에서 게임 서버를 실행하는 경우, NUMA 정보에 주의해야 합니다.

성능 조정 옵션	개요	비고	링크 및 명령어
NUMA	c4.8xlarge 인스턴스는 두개의 NUMA 노드를 가지고 있으므로 NUMA 가 이슈가 될 수 있음	C4.8xlarge 보다 작은 크기의 인스턴스는 하나의 NUMA 노드만 가지므로 NUMA 이슈 없음	NUMA 이슈를 처리하는 세가지 선택지가 있음: CPU 고정, NUMA 밸런싱 및 numad 프로세스
가상 메모리	가상 메모리 옵션 조정으로 일부 게임서버들은 성능 개선 효과를 볼 수 있음		다음 항목을 /etc/sysctl.conf 에 추가: vm.swappiness = New_Value (시스템의 현재값으로부터 시작하여 값을 변경하여 테스트해보는 것을 추천) 다음 항목을 /etc/sysctl.conf 파일에 추가: vm.dirty_ratio = New_Value (Amazon Linux 기본값인 20 으로 설정하는 것을 추천) 다음 항목을 /etc/sysctl.conf 파일에 추가: vm.dirty_background_ratio = New_Value (Amazon Linux 기본값인 10 으로 설정하는 것을 추천)

NUMA

현재 세대의 모든 EC2 인스턴스는 NUMA 를 지원합니다. NUMA 는 스레드가 로컬 메모리, 다른 프로세서의 로컬 메모리 또는 공유 메모리 플랫폼에 모두 액세스 할 수 있게 해주는 멀티프로세서 시스템에서 사용되는 메모리 아키텍처입니다. 여기서 중요한 문제는 원격 메모리 사용이 로컬 메모리보다

훨씬 느린 액세스를 제공한다는 것입니다. 스레드가 원격 메모리에 액세스 할 때 성능상의 불이익이 발생하며, 상호 연결 경합 문제가 발생할 수 있습니다.

NUMA 의 이점을 활용할 일이 없는 응용 프로그램의 경우, 프로세서가 가능한 한 로컬 메모리만 사용하도록 하고 싶을 것입니다. 각각 별도의 NUMA 노드가 되는 두 프로세서 소켓에 접근이 가능한 c4.8xlarge 인스턴스의 경우에는 이것이 문제가 됩니다. 인스턴스 제품군의 더 작은 인스턴스들의 경우는 단일 NUMA 노드에만 접근이 가능하므로 NUMA 가 문제가 되지 않습니다. 추가적으로 NUMA 토폴로지는 인스턴스의 수명 동안 고정된 상태로 유지됩니다.

c4.8xlarge 인스턴스에는 두 개의 NUMA 노드가 있습니다. 이들 노드와 각 노드와 연관된 vCPU 들에 대한 세부 정보를 확인하려면 다음 명령을 실행하십시오.:

```
numactl --hardware
```

NUMA 정책 설정을 확인하려면 다음 명령어를 실행하십시오.:

```
numactl --show
```

다음 디렉토리에서 이 정보를 확인할 수도 있습니다. (각각의 NUMA 노드 디렉토리를 살펴보십시오):

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

numastat 도구를 사용하여 프로세스 및 운영체제에 대한 per-NUMA-node 메모리 통계를 확인할 수 있습니다. -p 옵션을 사용하면 특정 프로세스에 대해 이 정보를 확인할 수 있으며 -v 옵션을 사용하면 더 자세한 정보를 얻을 수 있습니다.

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

CPU 고정

잠재적인 NUMA 성능 문제를 해결하기위해서는 세가지 권장 옵션이 있습니다. 첫번째는 CPU 고정을 사용하는 것이고 두번째는 자동 NUMA 밸런싱이고 마지막은 numad 를 사용하는 것입니다. 이러한 옵션 중 어느 옵션이 당신의 게임 서버에 가장 최상의 성능을 제공하는지 확인하기 위해서는 각 옵션들이 실제 게임 서버로 테스트되어야 합니다.

먼저 CPU 고정을 살펴 보겠습니다. 여기에는 게임 서버 프로세스를 vCPU (또는 코어)와 NUMA 노드에 바인딩하는 작업이 포함됩니다. 이 작업은 numactl 을 사용하여 수행할 수 있습니다. 인스턴스에서 실행중인 각 게임 서버에 대해 다음 명령에서 $phys_cpu_core 및 $associated_numa_node 의 값과 game_server_executable 이름을 변경하여 실행하십시오. 추가 옵션은 numactl man 페이지[23]를 참조하십시오.

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

자동 NUMA 밸런싱

다음 옵션은 자동 NUMA 밸런싱을 사용하는 것입니다. 이 기능은 스레드나 프로세스를 현재 사용중인 메모리가 있는 프로세서 소켓에 유지하려고 시도합니다. 또한 응용 프로그램 데이터를 그것에 액세스하는 작업이 실행중인 프로세서 소켓으로 이동하려고 시도합니다. Amazon Linux AMI 2016.03 버전에서 자동 NUMA 밸런싱은 기본적으로는 비활성화되어 있습니다.[24]

인스턴스에서 자동 NUMA 밸런싱이 사용 가능한지 확인하려면 다음 명령을 실행하십시오.:

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

NUMA 밸런싱을 영구적으로 활성화 또는 비활성화하려면 다음 명령에서 Value 매개변수의 값을 0(비활성화)이나 1(활성화)로 설정하여 다음 명령을 실행하십시오.:

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

위 지침은 Amazon Linux 용입니다. 다른 배포판의 경우 /etc/sysctl.conf 에서 이를 설정해야 할 수 있습니다.

Numad

마지막으로 살펴볼 옵션은 Numad 입니다. Numad 는 NUMA 토폴로지를 모니터링하고 프로세스들이 실행되는 코어에 대한 NUMA 노드에서 계속 실행되도록 유지하는 데몬입니다. Numad 는 시스템 조건의 변화에 따라 조정이 가능합니다. NUMA 메모리 관리의 신비라는 기사가 자동 NUMA 밸런싱과 numad 간의 성능 차이를 설명하고 있습니다.[25]

numad 를 사용하려면 먼저 자동 NUMA 밸런싱을 비활성화해야합니다. Amazon Linux 에 numad 를 설치하려면 Fedora numad 사이트[26]를 방문한 다음 가장 최근의 안정 버전을 다운로드하십시오. numad 를 설치하려면 numad 디렉토리에서 다음 명령을 실행하십시오:

```
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
```

numad 에 대한 로그는 /var/log/numad.log 에서 찾을 수 있고 /etc/numad.conf 이 설정 파일입니다.

numad 를 실행하는 방법은 여러 가지가 있습니다. numad -u 옵션은 한 노드의 최대 사용률을 설정합니다. 기본값은 85 %입니다. NUMA 메모리 관리의 신비 기사에서 권장하는 설정은 -u100 이고 이는 최대 사용률을 100%로 설정합니다. 이는 프로세스들을 그들의 메모리 요구사항의 최대 100%까지 로컬 NUMA 노드에 남아있게 강제합니다.

```
sudo numad –u100
```

Numad 는 다음 명령을 사용하여 종료 할 수 있습니다.:

```
sudo /usr/bin/numad –i0
```

마지막으로, NUMA 를 완전히 비활성화하는 것은 원격 메모리 액세스에 문제를 발생시켜 좋은 선택지가 아니며 비활성화보다는 NUMA 토폴로지로 작업하는 것이 낫습니다. c4.8xlarge 인스턴스의 경우 대부분의 게임 서버에 대해 몇 가지 작업을 진행하는 것을 추천합니다. 최고의 성능을 제공하는 설정을 결정하기 위해 지금까지 논의한 옵션들을 실제로 테스트해 보는 것을 권장합니다. 이들 옵션들 중 어느 것도 프로세스의 원격 NUMA 노드에 대한 메모리 호출을 완전히 제거하지는 못하지만 각 설정들은 당신의 게임 서버에 더 나은 성능을 제공할 수 있습니다.

인스턴스에서 게임 서버를 실행하고 다음 명령을 사용하여 numa_foreign (즉, 이 노드에 할당되었어야 하지만 다른 NUMA 노드에 할당된 메모리) 및 numa_miss (즉, 이 노드에 할당되었지만 다른 NUMA 노드에 할당되었어야 하는 메모리) 항목들을 확인하는 것으로 어느 옵션이 가장 적합한지 테스트해볼 수 있습니다.:

```
numastat -v
```

NUMA 문제를 테스트하는 좀 더 일반적인 방법은 스트레스와 같은 도구를 사용하여 numastat 를 실행하여 foreign/miss 항목들이 있는지 확인하는 것입니다.:

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

가상 메모리

또, 고객들이 성능 향상을 위해 사용하는 몇가지 가상 메모리 조정 옵션들이 있습니다. 다시 한번 강조하지만, 이들이 당신의 게임 서버 성능을 향상시킬 수 있는지에 대해서는 전체적으로 테스트되어야 합니다.

가상 메모리 스와핑

가상 메모리 스와핑는 시스템이 익명 메모리 또는 페이지 캐시를 선호하는 정도를 제어합니다. 값이 낮으면 메모리 부족으로 인한 프로세스들의 스와핑 발생이 줄어들어 지연시간을 줄여주지만 I/O 성능은 떨어질 수 있습니다. 가능한 값은 0 ~ 100 입니다. Amazon Linux 의 기본값은 60 입니다. 해당값을 절반으로 줄인 다음 테스트하는 것으로 시작해보는 것을 추천합니다. 값을 더 줄이는 것도 게임 서버 성능에 도움이 될 수 있습니다.

시스템의 현재값을 확인하려면 다음 명령어를 실행하십시오.:

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

이 매개변수를 재부팅시에도 유지되도록 설정하려면 /etc/sysctl.conf 파일에 다음과 같이 새 값을 추가하십시오.:

```
vm.swappiness = New_Value
```

가상 메모리 Dirty Ratio

가상 메모리 Dirty Ratio 는 시스템 가용 메모리가 특정 비율 이상 dirty page 상태가 되면 프로세스를 블록하고 사용중인 페이지를 디스크에 기록하도록 강제합니다. 가능한 값은 0 에서 100 까지입니다. Amazon Linux 의 기본값은 20 이며 이것이 권장되는 값입니다.

시스템의 현재값을 확인하려면 다음 명령어를 실행하십시오.:

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

이 매개변수를 재부팅시에도 유지되도록 구성하려면 다음과 같이 /etc/sysctl.conf 파일에 새 값을 추가하십시오.:

```
vm.dirty_ratio = New_Value
```

가상 메모리 Dirty Background Ratio

가상 메모리 Dirty Background Ratio 는 시스템 가용 메모리가 특정 비율 이상 dirty page 상태가 되면 시스템이 강제로 디스크에 데이터 쓰기를 시작하도록 강제합니다. 가능한 값은 0 에서 100 까지입니다. Amazon Linux 의 기본값은 10 이며 이것이 권장되는 값입니다.

시스템의 현재값을 확인하려면 다음 명령어를 실행하십시오.:

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

이 매개변수를 재부팅시에도 유지되도록 구성하려면 다음과 같이 /etc/sysctl.conf 파일에 권장값을 추가하십시오.:

```
dirty_background_ratio=10
```

디스크

멀티 플레이어 게임 서버에서 디스크가 병목 현상이 되는 경우는 거의 없으므로 디스크 성능 튜닝은 게임 서버의 성능에 가장 영향을 덜 미칩니다. 고객이 실제로 C4 인스턴스 제품군에서 고객들이 디스크 성능 문제를 경험하는 경우가 보고된 적은 없습니다. C4 인스턴스 제품군은 로컬 인스턴스 저장소가 없고 Amazon Elastic Block Store(EBS) [27]만을 저장소로 사용하고, 따라서 C4 인스턴스들은 기본적으로 EBS 최적화되어 있습니다. Amazon EBS 는 필요한 경우 최대 48,000 IOPS 를 제공할 수 있습니다. 부팅 및 OS/게임 EBS 볼륨 분리 사용과 같은 표준 디스크 성능 최적화 작업들을 수행할 수 있습니다.

성능 조정 옵션	개요	비고	링크 및 명령어
EBS 성능	C4 는 EBS 를 사용하고 기본적으로 EBS 최적화되어 있음. 게임서버 요구사항에 적합하도록 IOPS 설정 가능		NA

벤치마킹 및 테스트

벤치마킹

Linux 를 벤치마킹하는 데는 많은 방법이 있습니다. 유용한 선택지 중 하나는 Phoronix Test Suite 입니다.[28] 이 Python 기반의 오픈소스 제품은 많은 벤치마킹 (및 테스트) 옵션들을 제공합니다. 연속적인 테스트를 진행한 후에 결과들을 기존 벤치마크 결과들과 비교할 수 있습니다. 결과들을 온라인으로 확인하거나 비교할 수 있도록 OpenBenchmarking.org 에 업로드할 수도 있습니다.[29]

이미 많은 수의 벤치마크 정보들이 존재하고 대부분 OpenBenchmarking.org 테스트 사이트에서 찾을 수 있습니다.[30] 이들 중 게임 서버를 준비할 때 유용한 테스트들은 주로 CPU,[31] 멀티코어,[32] 프로세서[33] 및 종합[34] 테스트들입니다. 이 테스트들은 대개 여러 개의 하위 테스트들을 포함하고 있습니다. 일부 하위 테스트들은 다운로드할 수 없거나 제대로 실행되지 않을 수 있습니다.

시작하려면 먼저 필수 구성 요소를 설치해야 합니다.:

```
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
```

다음으로 Phoronix 를 다운로드 및 설치합니다.:

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

테스트를 설치하려면 install-sh 명령을 실행할 때 지정한 디렉토리 하위의 bin 디렉토리에서 다음을 실행하십시오.:

```
phoronix-test-suite install <test or suite name>
```

테스트를 실행하려면 다음을 수행하십시오.:

```
phoronix-test-suite benchmark <test or suite name>
```

결과를 Openbenchmark.org 에 업로드할지 여부를 선택할 수 있습니다. 이 옵션은 테스트 시작시 표시됩니다. "yes"를 선택하면 테스트의 이름을 지정할 수 있습니다. 실행 종료 후에는 모든 테스트 결과를 볼 수 있는 URL 이 제공됩니다. 한번 결과가 업로드되면 이전 테스트의 벤치 마크 결과 번호를 사용하여 벤치 마크를 재실행 할 수 있고 이전 결과들이 나란히 표시됩니다. 이 과정을 반복하여 많은 테스트 결과를 함께 표시 할 수 있습니다. 보통 작은 변경을 하고 벤치마크를 다시 돌리게 됩니다. 테스트 결과를 업로드하지 않고 명령줄 출력에서 확인할 수도 있습니다.

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

아래의 스크린샷은 c4.8xlarge 인스턴스에서 멀티코어 벤치마크 테스트 세트를 실행한 후 OpenBenchmarking.org 에서 표시된 출력의 예입니다.:

multicoretest

	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 성능 분석

CPU 성능 분석 또는 프로파일링을 위한 최상의 도구 중 하나는 Linux perf 명령어입니다.[35] perf record 및 perf report 명령어를 사용하면 각각 성능 데이터를 기록하고 분석할 수 있습니다. 성능 분석에 대한 상세한 내용은 이 백서의 범위를 벗어나지만, kernel.org wiki 와 Brendan Gregg 의 perf 리소스에서 훌륭한 성능 분석 관련 리소스를 찾을 수 있습니다.[36] 다음 절에서는 CPU 사용량을 분석하기 위해 perf 를 사용하여 flame graph 를 생성하는 방법에 대해 설명합니다.

시각적 CPU 프로파일링

게임 서버 테스트 중에 발생하는 일반적인 문제 중 하나는 복수의 게임 서버가 실행 중일 때(많은 경우, vCPU 고정을 사용하지 않았을 때) 한 vCPU 가 거의 100%의 사용률을 기록하는 반면 다른 vCPU 는 사용률이 낮다는 것입니다. 이러한 유형의 성능 문제 및 기타 유사한 CPU 문제를 해결하는 작업은 복잡하고 시간이 많이 소요되는 작업일 수 있습니다. 이러한 작업은 기본적으로 CPU 에서 실행되는 함수를 살펴서 CPU 를 가장 많이 사용하는 코드 경로를 찾는 과정을 포함합니다. Brendan Gregg 의 flame graph 를 사용하면 잠재적 CPU 성능 문제를 시각적으로 분석하고 문제를 해결할 수 있습니다.[37] Flame graph 는 가장 자주 사용되는 함수들을 쉽고 빠르게 식별할 수 있게 해 줍니다.

메모리 누수에 대한 그래프를 포함하여 여러 유형의 flame graph 들이 있지만, 이 문서에서는 CPU flame graphs 에 초점을 맞출 것입니다.[38] perf 명령어를 사용하여 분석 대상 데이터들을 생성한 다음 flame graph 를 이용하여 시각화 작업을 수행합니다.

먼저 필수 구성 요소를 설치합니다.:

```
# Perf 명령어를 설치합니다
sudo yum install perf

# perf record 명령어 실행을 위해 root 계정을 사용해야 할 필요성을 제거합니다
sudo sh -c 'echo 0  >/proc/sys/kernel/perf_event_paranoid'

# Flamegraph 를 다운로드합니다.
wget https://github.com/brendangregg/FlameGraph/archive/master.zip
```

```
# 마지막으로 다운로드한 파일의 압축을 해제합니다. Flame graph 실행 파일이 위치한
FlameGraph-master 라는 디렉토리가 생성됩니다.
unzip master.zip
```

Flame graph 로 의미 있는 데이터를 보려면 실제 게임 서버나 CPU 스트레스 도구를 실행해야 합니다. 게임 서버 혹은 CPU 스트레스 도구 실행 중에 perf 프로파일을 기록합니다. 모든 vCPU 나 특정 vCPU, 또는 특정 PID 에 대해 perf record 명령을 실행할 수 있습니다. 다음은 perf 명령어의 다양한 옵션에 대한 표입니다.:

옵션	비고
-F	Perf record 주기. 대부분의 사용 사례에서 99 Hz 면 충분
-g --	(CPU 함수 및 명령어들 대신)Stack trace 를 기록할 때 사용
-C	추적할 vCPU 를 특정할 때 사용
-a	모든 vCPU 가 추적 대상일 때 사용
sleep	Perf record 명령어가 실행될 시간을 초 단위로 지정

다음은 모든 vCPU 들의 성능지표를 추적하거나 하나의 vCPU 의 성능지표를 추적할 때 일반적으로 사용하는 perf record 명령어들입니다. FlameGraph-master 디렉토리에서 다음 명령을 실행하십시오:

```
# 모든 vCPU 들에 대해 perf record 를 실행합니다.
perf record -F 99 -a -g -- sleep 60

# -C 옵션 뒤에 번호를 설정하여 해당 vCPU 에 대하여 perf record 를 실행합니다..
perf record -F 99 -C CPU_NUMBER -g -- sleep 60
```

실행이 완료되면 다음 명령어를 실행하여 flame graph 를 생성합니다.:

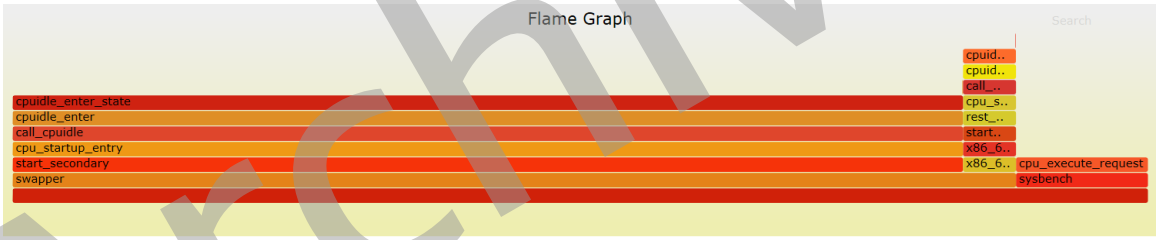
```
# perf 파일을 생성합니다. 이 명령어 실행시 "no symbol found" 에러가 발생하지만,
이 경우에는 flame graph 를 생성하기 위해 실행하는 것이므로 무시해도 됩니다.
perf script > out.perf
```

```
# stackcollapse 프로그램을 사용하여 샘플 스택을 한 줄로 만듭니다.
./stackcollapse-perf.pl out.perf > out.folded

# flamegraph.pl 를 실행하여 SVG vector 로 렌더링합니다.
./flamegraph.pl out.folded > kernel.svg
```

마지막으로 WinSCP 와 같은 도구를 사용하여 생성된 SVG 파일을 데스크탑에 복사하면 결과를 확인할 수 있습니다.

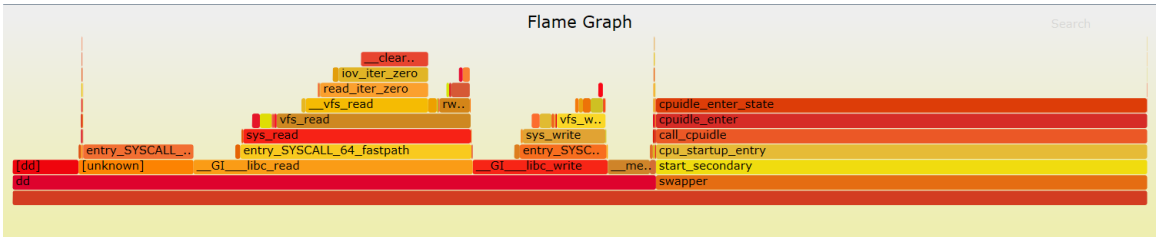
다음은 flame graph 의 두 가지 예입니다. 첫번째 예는 c4.8xlarge 인스턴스에서 sysbench 를 다음 옵션(for each in 1 2 4 8 16; do sysbench --test=cpu --cpu-max-prime=20000 --num- threads=$each run; done)으로 실행하고 perf 명령어로 60 초간 추적한 결과물을 시각화한 것입니다. 인스턴스에서 CPU 처리량이 sysbench 에 실제로 얼마나 적게 사용되고 있는지 알 수 있습니다. Flame graph 의 다양한 element 들 위로 마우스를 가져가면 샘플 수 및 각 영역에 소비된 CPU 리소스 비율에 대한 추가 세부사항들을 확인할 수 있습니다.



두번째 graph 는 동일한 c4.8xlarge 인스턴스에서 다음 스크립트를 실행하고 perf 명령어로 60 초간 추적한 결과물을 시각화하였습니다.:

```
(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)
```

이 graph 는 시스템 하부에서 일어나는 더 흥미로운 일련의 동작들을 보여줍니다



결론

이 백서의 목적은 AWS 에서 게임 서버를 최적으로 실행하기 위해 EC2 인스턴스를 어떻게 튜닝할 수 있는지 보여주는 것입니다. C4 인스턴스 제품군에서 Linux 상에서 게임서버들을 구동할 경우에 대한 네트워크, CPU 및 메모리의 성능 최적화에 중점을 두고 기술하고 있습니다. 게임 서버 운영에 있어 디스크가 병목 현상을 일으키는 경우는 거의 없으므로 디스크 성능은 영향을 적게 미칩니다.

이 백서는 AWS 에서 게임 서버를 운영할 때 도움이 될만한 컴퓨팅 인스턴스에 대한 정보들을 모아서 요약한 것입니다. 이 가이드가 당신의 게임 런칭을 가능한 성공적으로 만들기 위해 AWS 를 사용하여 게임서버를 빠르게 준비하고 운영하기 위한 핵심 정보, 성능 권장 사항 및 주의사항들을 상기시켜 당신의 시간을 많이 절약해주기를 바랍니다.

이 문서의 기여자

다음 개인들 및 단체들이 이 문서에 기여했습니다.:

- Greg McConnel, Solutions Architect, Amazon Web Services
- Todd Scott, Solutions Architect, Amazon Web Services
- Dhruv Thukral, Solutions Architect, Amazon Web Services
- Byungsu Kim, Solutions Architect, Amazon Web Services
- Seungmo Koo, Solutions Architect, Amazon Web Services

Notes

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

5 https://aws.amazon.com/vpc/

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

7 http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html

8 https://aws.amazon.com/ec2/faqs/#Enhanced_Networking

9 https://en.wikipedia.org/wiki/Single-root_input/output_virtualization

10 http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/enhanced-networking.html

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

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

13 https://www.kernel.org/doc/Documentation/networking/scaling.txt

14 http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-eni.html

15 http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/placement-groups.html

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

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

18 https://www.kernel.org/doc/Documentation/timers/timekeeping.txt

19 https://xenbits.xen.org/docs/4.3-testing/misc/tscmode.txt

20 http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/processor_state_control.html

21 https://raw.githubusercontent.com/brendangregg/msr-cloud-tools/master/showboost

22 https://en.wikipedia.org/wiki/Completely_Fair_Scheduler

23 http://linux.die.net/man/8/numactl

24 https://aws.amazon.com/amazon-linux-ami/2016.03-release-notes/

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

26 https://git.fedorahosted.org/git/numad.gi t

27 https://aws.amazon.com/ebs/

28 http://www.phoronix-test-suite.com/

29 http://openbenchmarking.org/

30 http://openbenchmarking.org/tests/pts

31 http://openbenchmarking.org/suite/pts/cpu

32 http://openbenchmarking.org/suite/pts/multicore

33 http://openbenchmarking.org/suite/pts/processor

34 http://openbenchmarking.org/suite/pts/universe

35 https://perf.wiki.kernel.org/index.php/Main_Page

36 http://www.brendangregg.com/perf.html

37 http://www.brendangregg.com/flamegraphs.html

38 http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html