

CPS로 메모리 사용률 문제 해결

목차

[소개](#)

[사전 요구 사항](#)

[요구 사항](#)

[사용되는 구성 요소](#)

[배경 정보](#)

[문제](#)

[CPS로 메모리 사용률 문제를 해결하는 절차](#)

소개

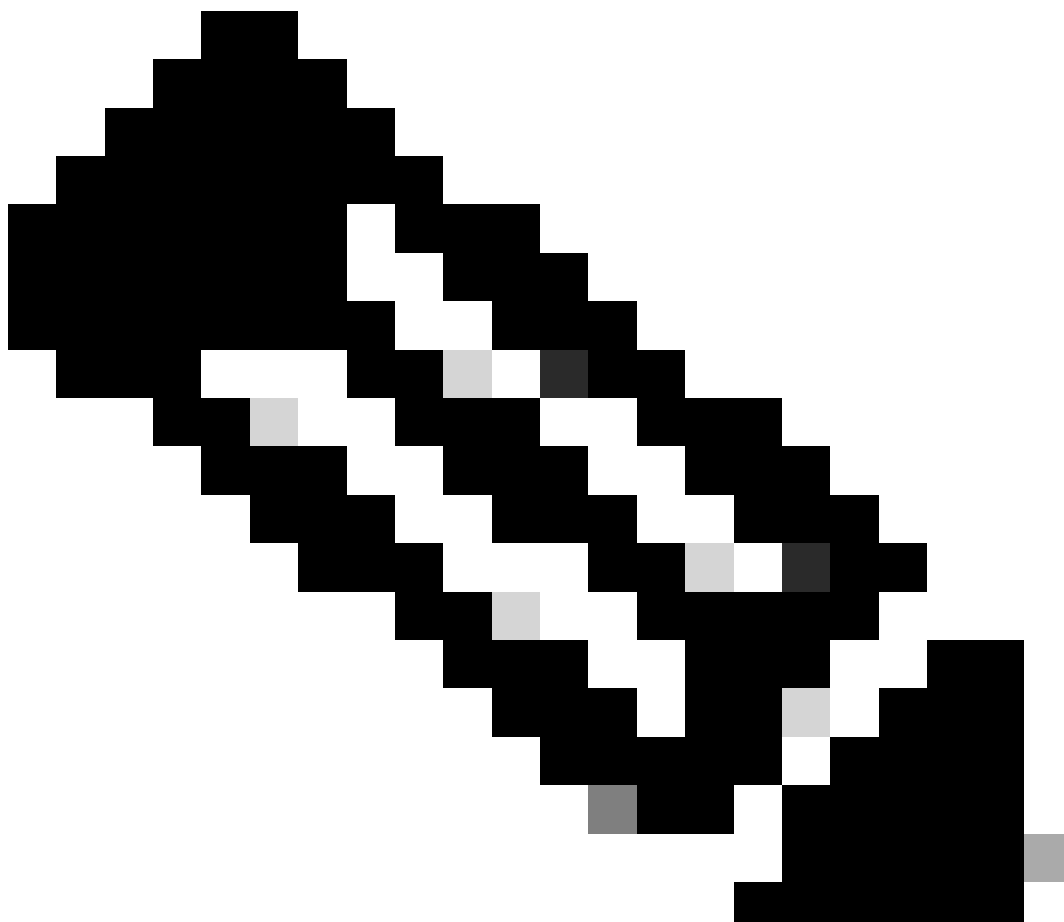
이 문서에서는 CPS(Cisco Policy Suite)의 메모리 사용률 문제 해결 절차에 대해 설명합니다.

사전 요구 사항

요구 사항

다음 주제에 대한 지식을 보유하고 있으면 유용합니다.

- Linux
- CPS
- 몽고DB



참고: CPS CLI에 대한 루트 액세스 권한이 있는 것이 좋습니다.

사용되는 구성 요소

이 문서의 정보는 다음 소프트웨어 및 하드웨어 버전을 기반으로 합니다.

- CPS 20.2
- UCS(Unified Computing System)-B
- MongoDB v3.6.17

이 문서의 정보는 특정 랩 환경의 디바이스를 토대로 작성되었습니다. 이 문서에 사용된 모든 디바이스는 초기화된(기본) 컨피그레이션으로 시작되었습니다. 현재 네트워크가 작동 중인 경우 모든 명령의 잠재적인 영향을 미리 숙지하시기 바랍니다.

배경 정보

Linux에는 소프트웨어 애플리케이션을 지원, 관리, 모니터링 및 구축할 수 있는 다양한 툴이 있습니

다.

제품 애플리케이션에 추가된 서비스 및 기능은 상당한 메모리를 소비할 수 있습니다. Linux 서버의 메모리 최적화는 애플리케이션의 실행을 더 원활하고 빠르게 만들 뿐만 아니라 데이터 손실 및 서버 충돌의 위험을 줄여줍니다.

Linux 시스템에 대한 메모리를 최적화하려면 먼저 Linux에서 메모리가 작동하는 방식을 이해해야 합니다. 일부 메모리 용어로 시작하고 Linux에서 메모리를 처리하는 방법을 설명한 다음 메모리 문제를 해결하고 방지하는 방법을 배웁니다.

한 시스템에 포함될 수 있는 총 메모리 양은 운영 체제의 아키텍처를 기반으로 합니다.

Linux의 전체 메모리를 가상 메모리라고 합니다. 여기에는 물리적 메모리(RAM - Random Access Memory라고도 함)와 스왑 공간이 포함됩니다. RAM을 더 추가하지 않으면 시스템의 물리적 메모리를 늘릴 수 없습니다. 그러나 하드 디스크에서 스왑 공간을 사용하여 가상 메모리를 늘릴 수 있습니다.

RAM은 시스템이 높은 메모리 소비 프로세스를 처리할 수 있는지 여부를 결정합니다.

사용자, 컴퓨터 프로세스 및 HDD(Hard Disk Drive)의 데이터가 RAM으로 전송됩니다. 필요한 경우 RAM이 이를 저장하고 사용자 또는 HDD에 다시 전송합니다. 데이터가 지속적이어야 하는 경우 RAM은 데이터를 CPU(Central Processing Unit)로 전송합니다.

컴퓨터에서 사용 가능한 여유 공간을 확인하려면 free 명령을 사용합니다.

```
[root@installer ~]# free -h
total used free shared buff/cache available
Mem: 11Gi 1.3Gi 2.9Gi 105Mi 7.4Gi 10Gi
Swap: 0B 0B 0B
[root@installer ~]#
```

문제

Linux 서버는 다양한 이유로 상당한 양의 메모리를 사용할 수 있습니다. 신속한 문제 해결을 효과적으로 수행하려면 먼저 가장 가능성 있는 이유를 배제해야 합니다.

Java 프로세스:

Java를 사용하여 구현된 애플리케이션이 여러 개 있으며, 이러한 애플리케이션의 구현 또는 컨피그레이션이 잘못되면 서버의 메모리 사용량이 많아질 수 있습니다. 캐싱과 세션 캐싱 안티패턴에서 컨피그레이션이 잘못된 것이 가장 일반적인 두 가지 원인입니다.

캐싱은 응용 프로그램의 높은 성능을 얻을 수 있는 일반적인 방법이지만 잘못 적용하면 시스템 성능이 저하될 수 있습니다. 컨피그레이션이 잘못되면 캐시가 너무 빨리 증가하여 시스템에서 실행 중인 다른 프로세스의 메모리가 줄어들 수 있습니다.

세션 캐싱은 애플리케이션의 중간 상태를 저장할 때 자주 사용됩니다. 개발자가 세션당 사용자를

저장할 수 있으며 데이터 개체 값을 쉽게 저장하거나 가져올 수 있습니다. 그러나 개발자는 세션 캐싱 데이터를 나중에 정리하는 것을 잊는 경향이 있습니다.

Java에서 데이터베이스 작업을 수행할 때 일반적으로 최대 절전 세션을 사용하여 서버와 데이터베이스 간의 연결을 만들고 세션을 관리합니다. 그러나 개발자가 최대 절전 모드로 작업할 때 자주 발생하는 오류가 있습니다. 스레드 안전을 위해 격리되는 대신 최대 절전 모드 세션은 동일한 HTTP(Hypertext Transfer Protocol) 세션에 포함됩니다. 이로 인해 응용 프로그램은 필요 이상으로 더 많은 상태를 저장할 수 있으며, 소수의 사용자만 메모리 사용량이 크게 증가합니다.

데이터베이스:

높은 메모리 소비 프로세스를 설명할 때는 데이터베이스를 언급해야 합니다. 응용 프로그램이 사용자 요청을 처리하는 동안 데이터베이스에 대한 읽기 및 쓰기가 많으므로 데이터베이스에서 상당한 메모리를 사용할 수 있습니다.

MongoDB 데이터베이스를 참조로 사용: 고성능을 달성하기 위해 데이터를 캐시하고 인덱싱하기 위한 버퍼 메커니즘을 적용합니다. 데이터베이스에 대한 여러 요청이 있을 때 최대 메모리를 사용하도록 데이터베이스를 구성하면 Linux 서버의 메모리가 곧 초과될 수 있습니다.

CPS 메모리 소비는 Grafana 그래프 또는 기타 모니터링 툴에서 적절한 KPI를 사용하여 모니터링할 수 있습니다. CPS VM(가상 머신)에서 메모리 사용량이 기본 임계값인 90%를 초과하여 증가하면 CPS에서 해당 VM에 대해 Low Memory 경보를 생성할 수 있습니다. 이 임계값은 free_mem_per 설정을 사용하여 CPS 구축 템플릿에서 구성할 수 있습니다.

메모리 사용량이 많은 프로세스/유틸리티를 식별합니다.

1. 메모리 부족 경보를 발생시킨 VM에 로그인합니다.

2. 디렉토리로 /var/log 이동하여 파일top_memory_consuming_processes을 체크인하여 메모리 소비량이 높은 프로세스 ID(PID)를 식별합니다.

```
***** Date: Tue May 16 05:06:01 UTC 2023 *****
PID PPID CMD %MEM %CPU RSS PRI STAT PSR WCHAN NI P
9435 1 /usr/bin/java -XX:OnOutOfMe 26.7 77.9 4353796 5 S<1 2 - -15 *
24139 1 /usr/java/default/bin/java 1.0 0.0 174636 20 S1 3 - 0 *
2905 2862 /usr/sbin/collectd -C /etc/ 1.0 0.2 169104 20 S1 1 hrtimer_nanos1 0 *
913 1 /usr/lib/systemd/systemd-jo 0.4 0.1 69364 20 Ss 5 do_epoll_wait 0 *
1513 1 /usr/libexec/platform-pytho 0.1 0.0 27912 20 Ssl 5 - 0 *
3379 2905 /usr/sbin/collectd -C /etc/ 0.1 0.0 23716 20 S1 3 - 0 *
3377 2905 /usr/sbin/collectd -C /etc/ 0.1 0.0 23712 20 S1 4 - 0 *
3378 2905 /usr/sbin/collectd -C /etc/ 0.1 0.0 23712 20 S1 5 - 0 *
3380 2905 /usr/sbin/collectd -C /etc/ 0.1 0.0 23712 20 S1 5 - 0 *
***** END *****
```

3. 응용 프로그램 또는 데이터베이스 프로세스인지 여부에 관계없이 이 명령을 사용하여 프로세스를 검증합니다.

<#root>

```
#ps -ef | grep <PID>
```

CPS로 메모리 사용률 문제를 해결하는 절차

Linux에서 메모리를 최적화하는 것은 복잡하며, 과부하 메모리를 수정하려면 상당한 노력이 필요합니다.

접근 방식 1.

캐시된 메모리 탐지 및 재확보:

경우에 따라 메모리 부족 경보는 Linux 메모리 관리에서 캐시에 객체를 할당한 결과일 수 있습니다.

VM이 캐시한 메모리 양을 평가하고 캐시된 메모리 중 일부를 해제하도록 Linux를 트리거합니다.

1. 둘 이상의 CPS VM에 캐시된 메모리의 양을 비교하여 각 VM에서 free -m 명령을 실행합니다.

```
[root@dc1-qns01 ~]# free -m
total used free shared buff/cache available
Mem: 15876 5262 4396 808 6217 9628
Swap: 4095 0 4095
[root@dc1-qns01 ~]#
```

2. 비활성 캐시 메모리 중 일부를 재확보하려면 이 명령을 실행합니다.

```
#free && sync && echo 3 > /proc/sys/vm/drop_caches && echo "" && free
```

```
[root@dc1-qns01 ~]# free -m
total used free shared buff/cache available
Mem: 15876 5016 8782 872 2076 9809
Swap: 4095 0 4095
[root@dc1-qns01 ~]#
```

참고:

1. 이 명령은 IO(입력 출력) 및 CPU(중앙 처리 장치) 사용량이 일시적으로 증가할 수 있는 캐시 개체를 무시하므로 사용량이 적은 시간/유지 관리 창에서 이 명령을 실행하는 것이 좋습니다.
2. 이것은 비파괴 명령이며 사용 중이 아닌 사용 가능한 메모리입니다.

메모리 부족 경보가 여전히 해결되지 않으면 Approach 2를 계속 진행합니다.

접근 2.

QNS

등과 같은 애플리케이션 프로세스로 인해 메모리 소비가 많은 경우

1. 프로세스를 다시 시작합니다.

<#root>

Command Syntax:

```
#monit restart <process name>
```

2. 명령에 의한 메모리 사용량 감소를 free-m 확인합니다.

메모리 부족 경보가 여전히 해결되지 않으면 접근 방식 3으로 진행합니다.

접근 3.

일반적으로 VM(디스크 메모리 CPU)에 대한 리소스를 늘리기 위해 VM 재시작을 수행하므로 경보가 생성된 VM을 재시작합니다.

sessionmgr VM에 대한 높은 메모리 사용률을 확인한 경우 접근 방식 4를 진행합니다.

접근 4.

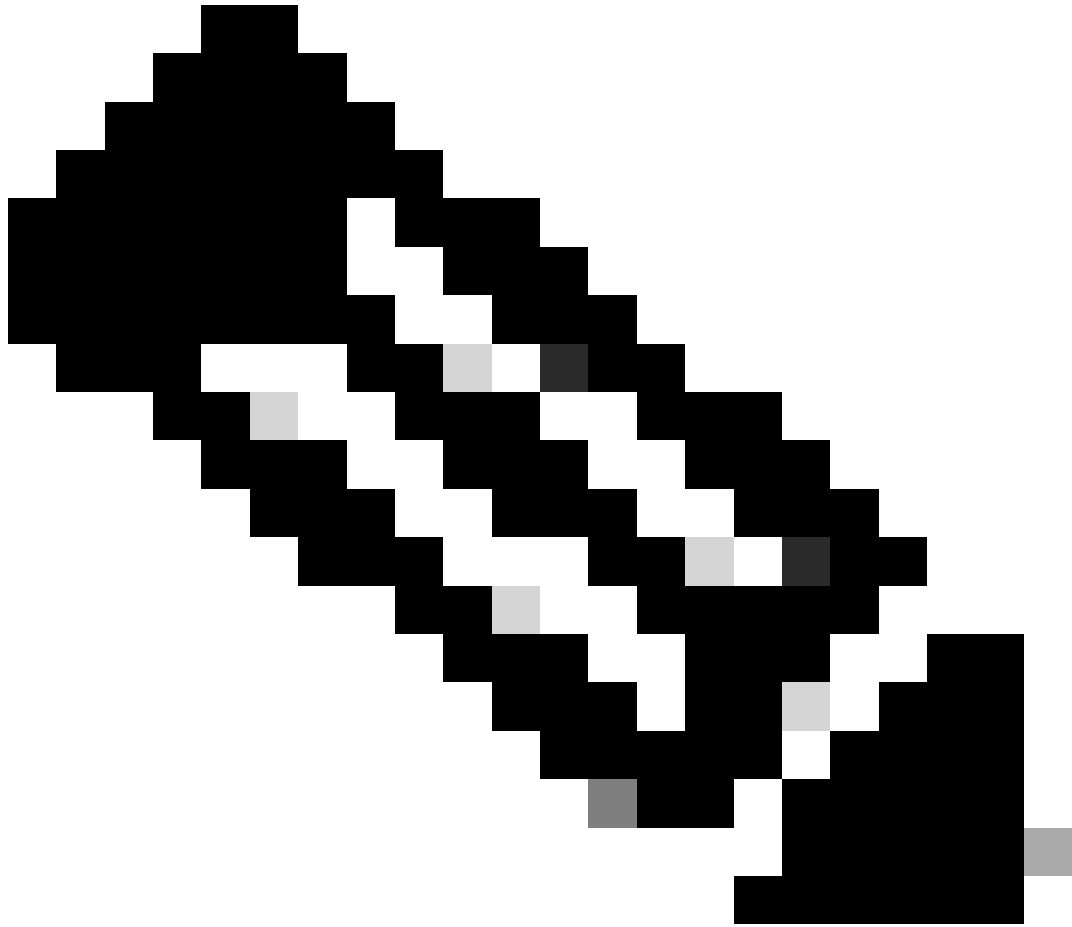
1. 높은 메모리 사용량이 확인된 VM에 로그인합니다.

2. 디렉토리로 /var/log/mongodb-<xxxx>.log 이동하여 메모리 소비와 writeConcernMajorityJournalDefault 매개변수와 관련된 경고/메시지가 있는지 파일을 체크 인합니다.

```
2022-12-13T00:30:39.012+0200 I REPL [replexec-0] ** WARNING: This replica set node is running without journaling enabled but the
2022-12-13T00:30:39.012+0200 I REPL [replexec-0] ** writeConcernMajorityJournalDefault option to the replica set config
2022-12-13T00:30:39.012+0200 I REPL [replexec-0] ** is set to true. The writeConcernMajorityJournalDefault
2022-12-13T00:30:39.012+0200 I REPL [replexec-0] ** option to the replica set config must be set to false
2022-12-13T00:30:39.012+0200 I REPL [replexec-0] ** or w:majority write concerns will never complete.
2022-12-13T00:30:39.012+0200 I REPL [replexec-0] ** In addition, this node's memory consumption may increase until all
2022-12-13T00:30:39.012+0200 I REPL [replexec-0] ** available free RAM is exhausted.
```

3. 각 mongoShell에 로그인하여 mongo protocolVersion 및 writeConcernMajorityJournalDefault 의 현재 값을 확인합니다.

```
set04:PRIMARY> rs.status().optimes.lastCommittedOpTime.t
NumberLong(0)
set04:PRIMARY>
```



참고: mongo 프로토콜 버전 0의 NumberLong o/p에서는 항상 음수 값입니다.

```
set04:PRIMARY> rs.conf().writeConcernMajorityJournalDefault  
set04:PRIMARY>
```

주: 출력에서 none을 반환하는 경우 이 값은 기본적으로 true로 writeConcernMajorityJournalDefault 설정된 것으로 간주해야 합니다.

4. protocolVersion1 writeConcernMajorityJournalDefault 경우 true 각 mongoShell에서 이 명령을 실행하여 값을 다음으로 수정합니다 writeConcernMajorityJournalDefault. false.

```
#cfg=rs.conf()
#cfg.writeConcernMajorityJournalDefault=false
#rs.reconfig(cfg)
```


5. 값이 writeConcernMajorityJournalDefault 로 변경되었는지 확인합니다false.

```
set03:PRIMARY> rs.conf().writeConcernMajorityJournalDefault  
false  
set03:PRIMARY>
```

6. 명령에 의한 메모리 사용량 감소를 free-m 확인합니다.

이 번역에 관하여

Cisco는 전 세계 사용자에게 다양한 언어로 지원 콘텐츠를 제공하기 위해 기계 번역 기술과 수작업 번역을 병행하여 이 문서를 번역했습니다. 아무리 품질이 높은 기계 번역이라도 전문 번역가의 번역 결과물만큼 정확하지는 않습니다. Cisco Systems, Inc.는 이 같은 번역에 대해 어떠한 책임도 지지 않으며 항상 원본 영문 문서(링크 제공됨)를 참조할 것을 권장합니다.