목차

Linux Kernel Parameter - Dirty ; Page cache control

원본 링크 : http://kakaodbe.blogspot.kr/2014/03/linux-kernel-parameters-and-mysql_2226.html

Overview

리눅스 커널 파라미터들 중 데이터 기록 빈도와 관련된 파라미터들에 대해 조사해보았고, 이에 따라 MySQL 서버에 어떤 영향을 미칠지 정리해보았습니다.

Definition

Linux의 /proc/sys/vm 디렉토리에는 서버의 가상메모리와 관련된 커널 파라미터들이 파일로 존재합니다. 이 중 데이터의 기록, 즉 Page Cache에 있는 dirty data를 disk에 flush하는 것과 관련한 6개의 파라미터들에 대해 설명드리겠습니다.

Kernel Parameter	description	Default
dirty_background_ratio	전체 시스템 메모리에서 dirty page 비율, pdflush 데몬이 dirty page를 쓰기 시작하는 기준값임.	10(%)
dirty_ratio	전체 시스템 메모리에서 dirty page 비율, 그러나 이 값은 process가 dirty page를 쓰기 시작하는 기준값임.	40(%)
dirty_background_bytes	위의 dirty_background_ratio와 같은 의미지만 이 변수에는 바이트 값이 들어감. (비율과 달리 이 값은 절대적인 값임.) 일반적인 경우 dirty_background_ratio보다 이 변수값이 더 우선적이나, 값이 0으로 설정된 경우에는 dirty_background_ratio의 설정이 우선권을 가짐	0
dirty_bytes	위의 dirty_ratio와 같은 의미지만 이 변수에는 바이트 값이 들어감. (비율과 달리 이 값은 절대적인 값임.) 일반적인 경우 dirty_ratio 보다 이 변수값이 더 우선적이나, 이 값이 0으로 설정된 경우에는 dirty_ratio의 설정이 우선권을 가짐	0
dirty_writeback_centisecs	이 값에 해당하는 시간을 주기로 캐시를 체크해서 dirty page들을 디스크에 flush함	500(1/100sec)
dirty_expire_centisecs	page들이 수정된 상태로 있을 수 있는 시간을 나타냄, 즉 dirty page가 캐시에 존재할 수 있는 시간을 말함	3000(1/100sec)

Process

커널 스레드 pdflush는 'drity_writeback_centisecs' 마다 깨어나서 writeback 여부를 확인합니다. 즉, 전체 모든 파일들의 page cache를 체크하여 전체 메모리에 대한 dirty page의 비율이 'dirty_background_ratio'를 넘는지 검사하고 이 값을 넘는 경우 'dirty_expire_centisecs' 이상 존재한 dirty page들을 disk에 flush합니다.

Opinion

dirty_background_ratio 값을 크게 설정할 경우 변경해야될 페이지들이 그만큼 캐시에 많이 존재할 수 있게 됩니다. 그렇게 되면 pdflush 데몬이 disk에 flush할 dirty page들이 많아지는 것이며, 이 때문에 I/O 시간이 길어집니다. 보통 일반 시스템에서 디스트I/O는 병목지점이기 때문에 I/O 시간이 길어지면 그만큼 시스템이 느려질 것이라 생각됩니다. 또한 이렇게 변경될 데이터들이 캐시에 많이 있는데 서버 장애가 나면 그 데이터는 손실되기 때문에 dirty_background_ratio 값은 줄여서 사용하는 것이 효율적이라 생각합니다. (특히 DB에서 insert나 update 작업이 많아서 데이터 파일에 쓰기 작업이 많은 경우!)

dirty_writeback_centisecs 값도 클 경우 그만큼 체크하는 주기가 길어지는 것이기 때문에 한번에 처리할 page 수가 많아질 것입니다. 따라서 이 값도 어느정도 줄여서 한번에 비워내야할 page수를 줄이는 것이 효율적이라고 생각합니다.

dirty_expire_centisecs은 디폴트값이 30초로, 이 시간이 지나기 전까지는 실질적으로 disk에 쓰지 않는다는 것입니다. 그렇다면 30초동안 dirty data가 계속 캐시에 머무른다는 것이므로 이 값도 줄여서 사용하는 것이 옳다고 생각합니다. 하지만 그렇다고 해서 너무 줄이면 너무 빈번하게 disk에 쓰기를 시도하기 때문에 너무 낮추지는 말고 적절히 줄여서 사용하는 것이 바람직하다고 생각하다고 생각합니다. (하지만 InnoDB 스토리지 엔진을 사용하는 경우 innodb_flush_method 옵션에 따라 위 커널 파라미터들에 대해 영향을 받지 않을 수도 있을 것 같습니다. 이런 경우 innodb_max_dirty_pages_pct 값을 조정해줄 수 있습니다.) ** innodb_max_dirty_pages_pct : innodb buffer fool 내에 허용된 dirty page 비율

Latest Issue and Kernel Parameters

최근 일부 서버에서 디스크 i/o의 급증으로 인해 서버의 부하가 심한 이슈가 있었습니다. 이슈가 발생한 원인은 해당 서버의 Filesystem Cache 대부분이 dirty data들로 꽉 차있는 경우, swap 방지를 위해 설정해놓은 cache unmap 스크립트가 디스크에 dirty page를 쓰면서 1기가정도까지 Cache를 줄이지만 Cache에 남은 그 data들이 모두 dirty data이기 때문에 disk i/o가 다시 발생하여 서버에 부하가 온 것입니다.

그래서 현재는 해당 서버의 커널 파라미터를 아래와 같이 조정해준 상태입니다.

```
/proc/sys/vm/dirty_background_ratio : 10 ===> 1
/proc/sys/vm/dirty_expire_centisecs : 3000 ===> 1000
```

dirty_background_ratio 값을 1로 낮춤으로써 전체 cache의 1%만큼만 dirty page가 존재할 수 있게 하였고, dirty_expire_centisecs도 10초로 줄임으로써 너무 오랫동안 dirty page가 cache에 남아있지 않도록 조치를 취하였습니다.

Details of Mysql_Cache_Unmap

이전에 Kakao DB Team 블로그에서 Filesystem Cache의 swap 문제와 함께 cache_unmap 스크립트에 대해 언급했었습니다. (참고 url : http://kakaodbe.blogspot.kr/2013/09/mysql-linux-filesystem-cache-2.html) 이 챕터에서는 cache unmap에 대해 실제로 서버에 어떻게 적용하고 있는지에 대해 살펴보겠습니다.

지금 db서버에 적용된 mysql_cache_unmap 스크립트는 Matt이 작성한 것으로 요시노리 개발자가 작성한 스크립트를 참고한 것입니다. (소스 코드 파일 다운 : mysql_cache_unmap.c)

주기적으로 cache unmapping을 수행하기 위해 Crontab에는 아래와 같이 설정되어 있습니다.

```
# Crontab Setting
*/10 * * * * root LD_LIBRARY_PATH=/otp/mysql/lib:
/otp/mysql/admin/mysql_cache_unmap --defaults-file=/etc/my.cnf --
binary_os_cache_size=1024M > /otp/mysql/admin/mysql_cache_unmap.log2>&1
```

–defaults-file 옵션을 통해 mysql server의 configuration files 경로를 지정할 수 있습니다. cache unmap 유틸리티에서는 이 설정 파일을 읽어서 binary log와 data file 그리고 innodb redo log 파일의 경로를 얻게 됩니다. –binary_os_cache_size 옵션은 Linux OS Cache에서 제거하지 않고 남겨둘 용량을 의미하는데, cache_unmap 유틸리티는 최근에 발생한 바이너리 로그부터 지정된 용량(위 경우 1024M)을 Linux OS Cache에 남겨두게 됩니다.

다음은 mysql_cache_unmap 스크립트 실행로그 입니다.

```
root@host:~ 14:10:09> cat /opt/mysql/admin/mysql_cache_unmap.log
Read configuration
    innodb_data_dir : /opt/mysql/data
    innodb_log_dir : /opt/mysql/data
    binary_log_dir : /opt/mysql/data/mysql-binary
    binary_os_cache_size : 1073741824
    relay_log_dir : /opt/mysql/data/mysql-relay
> unmap_file_all : datafile : /opt/mysql/data/dbname1/tablename1.ibd
> unmap_file_all : datafile : /opt/mysql/data/dbname1/tablename2.ibd
> unmap_file_all : logfile : /opt/mysql/data/ib_logfile0
> unmap_file_all : logfile : /opt/mysql/data/ib_logfile1
> skip unmap file: binary : /opt/mysql/data/mysql-binary.041728 : 0 ~
60083119
> skip unmap file: binary : /opt/mysql/data/mysql-binary.041727 : 0 ~
104857974
> skip unmap file: binary : /opt/mysql/data/mysql-binary.041726 : 0 ~
104857975
> skip unmap file: binary : /opt/mysql/data/mysql-binary.041725 : 0 ~
104857883
> skip unmap file: binary : /opt/mysql/data/mysql-binary.041724 : 0 ~
104857852
> skip unmap file: binary : /opt/mysql/data/mysql-binary.041723 : 0 ~
104857920
> skip unmap file: binary : /opt/mysql/data/mysql-binary.041722 : 0 ~
104857989
> skip unmap file: binary : /opt/mysql/data/mysql-binary.041721 : 0 ~
104857781
> skip unmap file: binary : /opt/mysql/data/mysql-binary.041720 : 0 ~
104857784
> skip unmap file: binary : /opt/mysql/data/mysql-binary.041719 : 0 ~
104857666
```

```
> unmap_file_segment : binary : /opt/mysql/data/mysql-binary.041718 : 0 ~
34920063 of 104857944
> unmap_file_all : binary : /opt/mysql/data/mysql-binary.041717 : 0 ~
104857764
> unmap_file_all : binary : /opt/mysql/data/mysql-binary.041716 : 0 ~
104857852
> unmap_file_all : binary : /opt/mysql/data/mysql-binary.041715 : 0 ~
104857772
> unmap_file_all : binary : /opt/mysql/data/mysql-binary.041714 : 0 ~
104857847
```

mysql_cache_unmap은 위 로그에서 볼 수 있듯이 filesystem cache에서 datafile 및 redo_log_file, binary_log_file의 caching 영역을 unmap합니다. mysql_cache_unmap을 사용하는 이유는 centOS 5.x에서 Filesystem Cache를 가장 우선시함에 따라 실제 free memory가 없는 경우 memory를 가장 많이 사용하고 있는 application의 일정량을 swap으로 내려버리는 것에 있습니다. 만약 memory를 가장 많이 사용하고 있는 것이 MySQL이라면 MySQL에서 캐싱된 데이터들이 swap되어 내려갈 것이고, 쿼리 수행 시 다시 데이터를 디스크에서 퍼올려야하는 문제가 발생하게 됩니다. 따라서 이러한 문제가 생기지 않도록 하기 위해 mysql_cache_unmap을 주기적으로 실행해주며 cache size가 너무 많이 증가하지 않도록 하고 있습니다.

Attention

cache_unmap은 redo log와 innodb system data file의 내용은 무조건 하나도 남기지 않고 Cache에서 purge합니다. 만약 redo log나 innodb data file의 쓰기 방식 (innodb_flush_method 옵션)이 innodb_flush_method =O_DIRECT인 경우에는 innodb data file은 direct io를 수행하지만 innodb redo log는 direct_io가 아니라 cached io를 수행하게 됩니다. 따라서 이 경우에는 redo log 파일에 대한 cache unmap은 피하는 것이 좋으며, innodb_flush_method가 O_DIRECT나 ALL_O_DIRECT (MariaDB & PerconaServer only)를 사용하지 않는다면 redo log와 Innodb data file의 cache unmap은 사용하지 않는 것이 좋습니다. 이는 mysql_cache_unmap 프로그램을 변경해서 적용하는 것을 추천합니다.

mysql_cache_unmap 스크립트에서는 'posix_fadvise'라는 커널 함수를 이용하여 caching된 영역을 unmap합니다.

```
## posix_fadvise SYSNOPSIS
#define _XOPEN_SOURCE 600
#include <fcntl.h>
int posix_fadvise(int fd, off_t offset, off_t len, int advice);
```

fd가 참조하는 파일내에서 offset을 시작으로 len bytes까지의 영역에 대해 advice값에 해당하는 설정을 적용합니다. 이때 len의 값이 '0'일 경우, 파일의 끝이라 간주하고 파일 전체가 설정 적용 영역에 해당됩니다. 함수가 제대로 수행이되면 '0'값을 반환하고, 에러가 발생한 경우에는 해당 에러에 대한 넘버를 반환합니다.

아래는 mysql_cache_unmap 소스에서 발췌한 코드로 posix_fadvise 함수를 통해 실질적으로 cache unmap이 이루어지는 부분입니다.

```
#define _XOPEN_SOURCE 600
```

```
#include <fcntl.h>
int unmap_file_segment(const char *fpath, size_t start, size_t len){
  int fd = open(fpath, O_RDONLY);
  if (fd < 0){
    fprintf(stderr, "ERROR : Failed to open %s\n", fpath);
    return 1;
  }
  int r = posix_fadvise(fd, start, len, POSIX_FADV_DONTNEED);
  if (r != 0){
    fprintf(stderr, "ERROR : posix_fadvice failed for %s\n", fpath);
  }
  close(fd);
  /* if posix_fadvise is succeeded, then sleep 25 milli seconds */
  usleep(25 * 1000);
  return 0;
}
```

코드에서 advice 파라미터의 값이 'POSIX_FADV_DONTNEED'로 설정되어 있는 것을 볼 수 있습니다. advice 타입에는 여러 종류가 있으나, 'POSIX_FADV_DONTNEED'로 값을 설정한 경우 커널이 page cache(filesystem cache)에서 해당 file이 차지하는 영역 중 offset을 기준으로 len길이 까지 만큼의 영역에 대해 unmapping을 진행하게 됩니다.

~~DISCUSSION~~