MySQL 데이터베이스 최적화

김병준 bjkim@itbridge.co..kr

MySQL이 오픈소스이기 때문일까? Oracle이나 MS SQL의 경우 적절한 하드웨어에 온갖 튜닝이 다 된 상태로 사용하면서 MySQL은 그저 설치만한 상태 그대로 이용하는 경우가 많다. MySQL에 문제가 있다고 해서 기술지원을 가보면 기본적인 설정에도 문제가 있는 경우가 매우 많았던 것이 사실이다. 적을 알고 나를 알면 백전 백승이라고 했다. 먼저 현재 시스템에 대한 모니터링을 통해 현재 MySQL이 적절히 작동하고 있는지 문제가 무엇인지부터 살펴보자.

MySQL 데이터베이스 모니터링

튜닝의 시작은 먼저 현재 시스템의 상태와 문제점을 파악하는 것이다. 이를 위해선 여러 가지 방법을 통해 시스템을 모니터링 하는 것이다. 현재 MySQL을 모니터링 하는 방법에는 3가지가 있다. 첫째로 커맨드라인 명령어들을 이용하여 모니터링 하는 것이며 두 번째는 GUI기반의 관리 툴인 MySQL Administrator를 통한 모니터링 세 번째로 MySQL이 남기는 각종 로그를 통한 모니터링이 있다. 먼저 가장 기본적인 모니터링 방법인 커맨드라인 명령어들을 통한 모니터링에 대해 알아보자.

커맨드라인 명령어들을 통한 모니터링

커맨드라인 명령어들을 통한 모니터링의 가장 큰 장점은 어떠한 환경에서도 수행이 가능하며 가장 빠르고 정확하게 자신이 원하는 바를 알아낼 수 있다는 것이다. MySQL의 커맨드라인 프로그램과 각종 SHOW 명령어들에 대해 자세히 알아보자.

mysqladmin

mysqladmin은 MySQL 데이터베이스의 커맨드라인 기반 관리자 프로그램이다. Mysqladmin을 통해 시스템의 현재 설정상황과 동작상황을 모니터링 할 수 있다. 아래는 mysqladmin을 통해 수행할 수 있는 성능관련 커맨드들이다.

extended-status	MySQL 데이터베이스의 현재 상황을 보여준다.
flush-hosts	MySQL에 캐시 된 모든 호스트를 초기화한다.
flush-logs	MySQL의 log파일을 새로 작성하며 초기화한다.
flush-status	MySQL의 상태정보를 초기화한다.
flush-tables	MySQL에 캐싱된 테이블 정보를 초기화한다.

flush-thread	쓰레드 캐시에 저장된 쓰레드를 초기화한다.
flush-privileges	권한정보 테이블을 다시 읽는다.
kill id	특정 MySQL 프로세스를 죽인다.
Processlist	현재 MySQL 프로세스 목록은 본다.
Refresh	현재 캐시 되어 있는 모든 테이블을 초기화 하고 log파일을 새로 만든다.
Variables	설정 가능한 모든 변수를 보여줍니다.

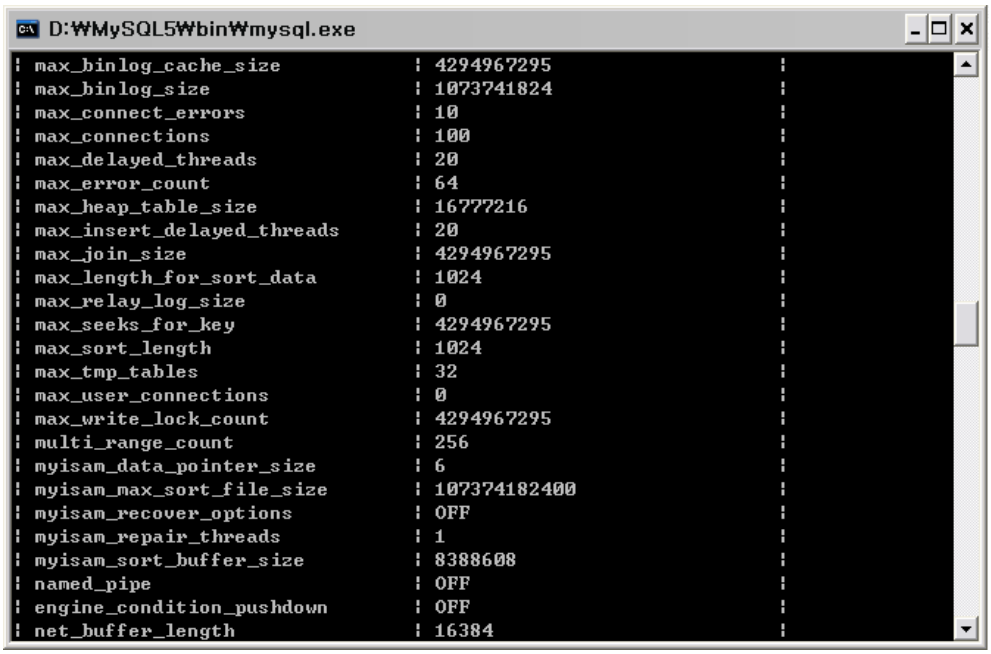
SHOW ENGINES

MySQL의 가장 큰 특징 중 하나는 여러 가지 스토리지 엔진을 가지고 있다는 것이다.
이 명령은 현재 MySQL의 시스템이 어떠한 스토리지 엔진을 사용할 수 있는지 보여준다.

```
mysql >SHOW ENGINES
+------------+---------+------------------------------------------------------------+
| Engine     | Support | Comment                                                    |
+------------+---------+------------------------------------------------------------+
| MyISAM     | DEFAULT | Default engine as of MySQL 3.23 with great performance     |
| HEAP       | YES     | Alias for MEMORY                                           |
| MEMORY     | YES     | Hash based, stored in memory, useful for temporary tables  |
| MERGE      | YES     | Collection of identical MyISAM tables                      |
| MRG_MYISAM | YES     | Alias for MERGE                                            |
| ISAM       | NO      | Obsolete storage engine, now replaced by MyISAM            |
| MRG_ISAM   | NO      | Obsolete storage engine, now replaced by MERGE             |
| InnoDB     | YES     | Supports transactions, row-level locking, and foreign keys |
| INNOBASE   | YES     | Alias for INNODB                                           |
| BDB        | NO      | Supports transactions and page-level locking               |
| BERKELEYDB | NO      | Alias for BDB                                              |
| NDBCLUSTER | NO      | Clustered, fault-tolerant, memory-based tables             |
| NDB        | NO      | Alias for NDBCLUSTER                                       |
| EXAMPLE    | NO      | Example storage engine                                     |
| ARCHIVE    | YES     | Archive storage engine                                     |
| CSV        | NO      | CSV storage engine                                         |
| BLACKHOLE  | NO      | Storage engine designed to act as null storage             |
+------------+---------+------------------------------------------------------------+
```

SHOW VARIABLES

MySQL은 엄청나게 많은 설정 가능한 값들을 가지고 있으며 SHOW VARIABLE 명령을 통해 현재 설정되어 있는 모든 값을 볼 수 있다.

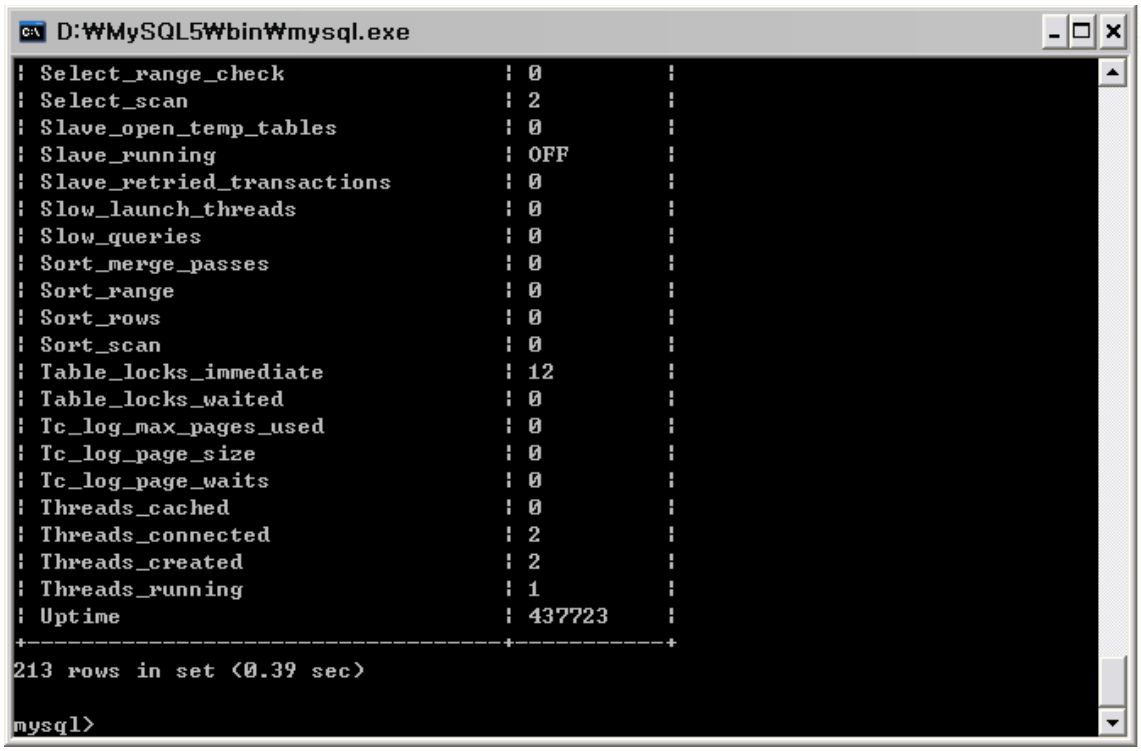


<화면1> SHOW VARIABLES로 통해 본 설정

SHOW VARIABLES로 볼 경우 총 207개 정도의 변수가 표시된다. 오히려 너무 많아서 원하는 값을 찾기가 힘들다. 그래서 SHOW VARIABLES 명령은 뒤에 LIKE % 를 사용하여 원하는 값만을 볼 수 있다.

SHOW STATUS

MySQL은 내부적으로 동작상황에 대한 실시간 통계정보를 가지고 있다. SHOW STATUS는 이러한 통계정보를 보기 위한 명령이다. 모니터링을 할 때 가장 기본이 되는 것이 바로 위에 설명한 SHOW VARIABLES의 정보와 SHOW STATUS의 정보로써 웹 기반의 모니터링 툴을 비롯한 각종 모니터링 툴들이 바로 이 두 명령어를 통해 나온 정보를 조합하여 사용하는 것이다. SHOW STATUS도 SHOW VARIABLES와 마찬가지로 LIKE를 사용하여 원하는 값만을 볼 수 있다.



<화면2> SHOW STATUS를 통해 본 서버의 사용 통계

SHOW PROCESSLIST

현재 동작하고 있는 MySQL데이터베이스 서버의 모든 동작중인 쓰레드와 유저 커넥션 정보를 보기 위한 명령어이다. 이를 통해 얻어진 정보로 시스템 자원을 지나치게 많이 사용하거나 잘못된 수행을 하고 있는 프로세스를 죽일 수 있다.

SHOW TABLE/TABLE STATUS/INDEX/INNODB STATUS

SHOW TABLE 명령은 현재 데이터베이스에 존재하는 테이블에 대한 기본적인 정보를 보여주며 SHOW TABLE STATUS는 각 테이블의 생성일자, 테이블 크기와 인덱스의 크기 등 구체적인 정보를 보여준다. 하지만 주의해야 할 점이 하나 있는데 SHOW TABLE STATUS의 경우 테이블의 스토리지 엔진이 MyISAM인 경우만 정확한 정보를 표시하며 InnoDB의 경우 부정확한 정보를 보여준다는 것이다. InnoDB 스토리지 엔진으로 되어 있는 테이블은 SHOW INNODB STATUS로 구체적인 정보를 확인할 수 있으며 SHOW INDEX를 통해 테이블의 인덱스에 대한 각종 정보를 볼 수 있다.

GUI 기반의 모니터링

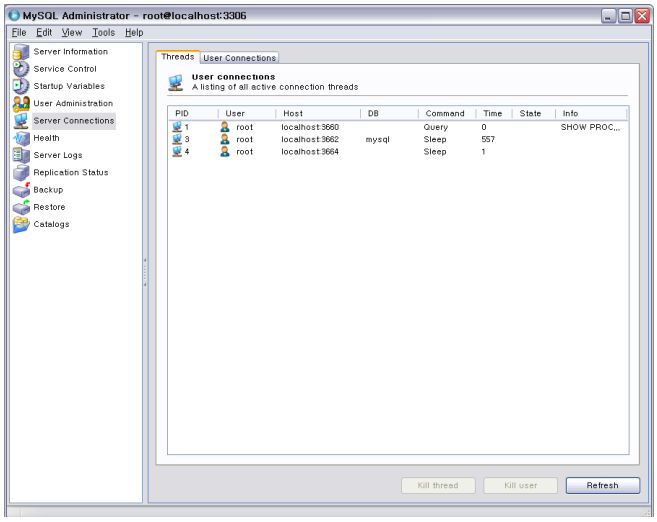
사실 MySQL의 경우 GUI기반의 여러 관리 툴들에 대한 지원이 매우 미약했던 것이 사실이다. 하지만 올해 초 4.1 발표 이후 연속적으로 GUI기반의 관리 툴을 발표했으며 그 완성도 또한 이전의 여러 GUI 프로그램들에 비해 비약적인 향상을 가져왔다. 빠르고 정확한 정보의 확인을 위해서는 커맨드라인 관리 툴이 유용하지만 사실 일반적인 모니터링에는 GUI기반의 모니터링 툴의 사용이 훨씬 편리하다. MySQL이 새롭게 내놓은 GUI기반 툴 중 모니터링을 위하여 이용할

수 있는 툴은 MySQL Administrator와 MySQL Query Browser이다.

MySQL Administrator

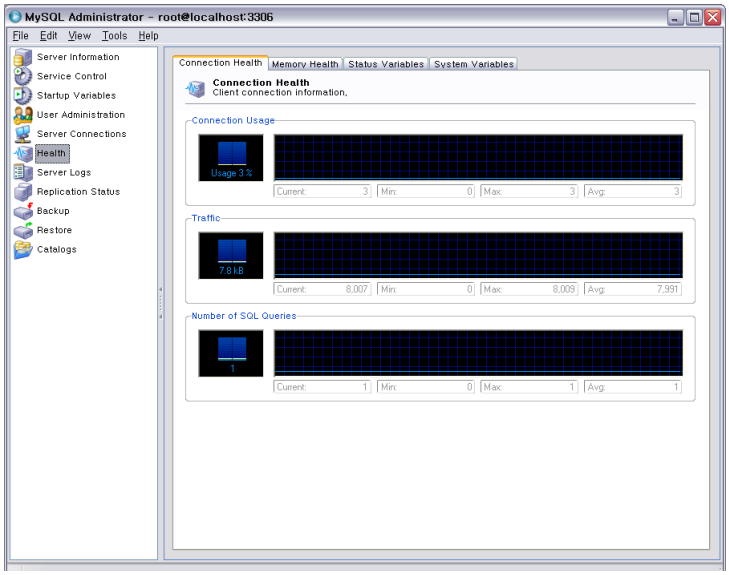
MySQL의 GUI기반 관리 툴인 MySQL Administrator는 기존의 GUI 관리 툴과는 달리 매우 다양한 관리 업무와 모니터링 작업을 편리하게 지원한다. 이 중 가장 돋보이는 기능이 모니터링 기능인데 이 툴로 인해 MySQL 튜닝 작업이 두 배는 편리해 졌다고 말할 수 있을 정도이다. MySQL Administrator의 모니터링 관련 메뉴는 Server Connections, Health, Server Logs 이렇게 세가지가 있다.

Server Connection은 커맨드라인 명령 중 SHOW PROCESSLIST와 같은 역할과 함께 각 유저별 접속 현황을 알 수 있다.



<화면3> Server Connection에서 thread별 정보를 보는 화면

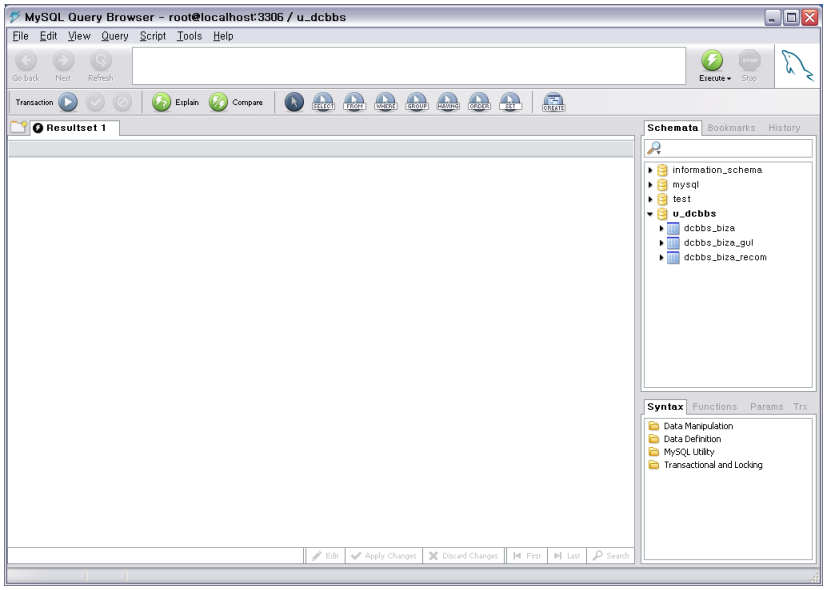
MySQL Administrator의 모니터링 기능의 백미는 바로 Health 메뉴이다. Health메뉴에서는 기본적으로 Connection Health, Memory Health, Status Variables, System Variables 네 가지 탭을 가지고 있으며 이전 커맨드라인 모니터링에서 했던 대부분의 모니터링 작업을 여기서 수행할 수 있다. 그리고 가장 큰 특징이라면 기본적으로 보여주는 주요 사항에 대한 모니터링 외에도 SHOW STATUS를 통해 볼 수 있는 모든 항목에 대한 모니터링 그래프를 추가 할 수 있다는 것이다. 이를 통해 직관적인 화면상의 변화를 보며 유저수의 변화나 시간대별 변화에 대한 쉽고 편한 모니터링이 가능해졌다.



<화면4> MySQL Administrator를 통해 커넥션 관련 정보를 실시간 모니터링 한다.

MySQL Query Browser

MySQL Query Browser는 기존의 MySQL관리자 또는 프로그래머들이 많이 이용하던 SQLGate와 비슷한 역할을 수행하는 프로그램이다. GUI상에서 MySQL쿼리들을 수행할 수 있으며 여러 탭을 이용하여 빠른 작업이 가능하다. 또한 내장 도움말과 명령어들에 대한 하이라이팅을 지원함으로써 편리하고 정확하게 작업할 수 있다. 위의 커맨드라인 명령어들을 여기에서 모두 실행해 볼 수 있다. 사실 초기버전에서는 한글을 입력하면 다운되는 치명적인 버그가 있었으나 지금은 수정이 되어 우리나라의 유저들도 자유로운 사용이 가능해졌다.



<화면5> MySQL Query Browser

로그를 통한 모니터링

적절한 수준의 로그를 남기는 것은 빠르고 건강한 MySQL을 유지하는 비결이다. 일반적으로 운영되는 서버라면 에러로그와 슬로우 쿼리 로그를 남기는 정도로 충분하지만 서비스를 위한 테스팅 기간이거나 문제를 찾아야 하는 시점이라면 General Query Log를 남겨 어떠한 쿼리가 가장 많이 사용되는지 파악하고 그 쿼리를 더 빠르게 할 수 있는 방법이 없는지를 찾는 것도 좋은 데이터베이스 최적화 방법 중 하나이다. 일반적으로 MySQL을 사용하는 유저들의 경우 기본적으로 지원하는 에러로그만을 남기고 슬로우 쿼리 로그를 남기지 않는 경우가 많은데 슬로우 쿼리는 MySQL의 성능을 떨어뜨리는 주범이다. 반드시 슬로우 쿼리로그를 남기고 확인하여 개선점을 찾도록 하자.

MySQL 서버 튜닝

MySQL의 튜닝은 MySQL의 데이터베이스 시스템관련 파라메터들에 대한 튜닝과 각각의 스토리지 엔진 관련 튜닝으로 나뉘어진다. 이번 회에서는 MySQL의 데이터베이스 시스템 즉 MySQL 전체 성능에 영향을 미치는 튜닝에 대해 알아보고 각각의 스토리지 엔진에 대한 튜닝 및 최적화는 다음 회에 알아보도록 한다.

MySQL의 시스템 관련 튜닝은 MySQL의 설정 파일인 my.cnf(윈도우의 경우 my.ini)파일을 수정하게 되며 MySQL 커넥션에 관한 부분과 메모리에 관한 부분으로 나눌 수 있다. 먼저 커넥션에 관한 부분부터 살펴보도록 하자.

MySQL 커넥션 튜닝

실질적으로 MySQL이 가장 많이 사용되는 분야를 꼽는다면 역시 인터넷 분야라고 할 수 있다. 포탈사이트나 게임사이트 등 매우 많은 부하가 발생하는 사이트에서 가장 문제가 되는 것은 MySQL의 커넥션에 관련된 문제이다. 커넥션에 관련된 모니터링은 SHOW STATUS LIKE '%CONNECT%'로 알아 볼 수 있다.

```
mysql> SHOW STATUS LIKE '%CONNECT%';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| Aborted_connects     | 12    |
| Connections          | 212   |
| Max_used_connections | 112176|
| Threads_connected    | 168   |
+----------------------+-------+
4 rows in set (0.00 sec)

mysql> SHOW STATUS LIKE '%CLIENT%';
```

```
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| Aborted_clients | 2     |
+-----------------+-------+
1 row in set (0.00 sec)
```

connect_timeout/interactive_timeout/wait_timeout

connect_timeout은 MySQL이 클라이언트로부터 접속 요청을 받는 경우 몇 초까지 기다릴지를 설정하는 변수이다. 기본값은 5초이며 일반적으로 수정해야 할 필요는 없다. Interactive_timeout은 “mysql>”과 같은 콘솔이나 터미널 상에서의 클라이언트 접속을 말한다. 기본값으로 8시간이 잡혀 있으나 1시간 정도로 낮추는 것이 좋다. 이러한 접속은 그다지 빈번하지 않으며 작업을 위해 접속하는 경우가 많으므로 따로 설정하지 않아도 크게 영향은 없다. 가장 중요한 것은 wait_timeout으로 wait_timeout은 접속을 맺은 후 쿼리가 들어올 때까지 기다리는 시간이다. 접속이 많은 데이터베이스 시스템에서는 이 값을 낮춰 sleep상태로 커넥션만 유지하고 있는 클라이언트들의 접속을 빠르게 끊어주는 동시접속을 낮추는 것으로써 전체 성능을 크게 향상시킬 수 있다. 하지만 주의해야 할 점은 너무 낮추게 되면 실제로 서비스를 하기도 전에 끊긴다던지 지나치게 잦은 커넥션이 발생한다는 것이다. 일반적으로 15~20 사이의 값이 적당하며 SHOW STATUS를 통해 aborted_client가 가장 적게 발생하도록 값을 맞춰야 한다. Aborted client는 2% 아래인 것이 바람직하며 물론 없는 것이 가장 좋은 상태인 것은 두말할 필요도 없다.

net_buffer_length/max_allowed_packet

MySQL의 커넥션은 쓰레드 단위로 일어나게 되는데 각 쓰레드가 생성되면서 메시지 전송을 위한 버퍼를 생성하게 된다. 일반적으로 max_allowed_packet만을 정해 놓는 경우가 많은데 net_buffer_length를 설정해 두면 그 용량을 넘는 메시지를 전달해야 할 경우 자동으로 이 값을 늘리게 된다. 그러므로 가장 효율을 높이기 위해서는 net_buffer_length를 일반적인 쿼리에서 전송되는 바이트 값의 평균 정도를 생각하여 충분히 낮은 값을 설정하여 두고 max_allowed_packet은 최대로 전송될 수 있는 높은 값을 설정하여 두는 것이 좋다. Max_allowed_packet은 1GB까지 설정 할 수 있다.

max_connections/back_log

max_connections는 서버가 허용하는 최대한의 커넥션 수이다. MySQL데이터베이스를 운영하고 있는 서버의 사양에 따라 달라질 수 있으며 일반적으로 120에서 250개 사이로 설정해 놓는 것이 보통이다. 하지만 접속이 많고 고용량의 서버의 경우 1000개 정도의 높은 값을 설정하는 것도 가능하다. Too many connection에러가 발생하지 않도록 적절한 값을 설정하는 것이 중요하다. Back_log의 경우 max_connection이상의 접속이 발생할 때 얼마만큼의 커넥션을 큐에 보관할지에 대한 설정 값이다. 기본값은 50으로 되어 있으며 접속이 많은 서버의 경우 이 값을

늘려줄 필요가 있다.

skip-name-resolve

외부로부터 접속요청을 받을 경우 인증을 위해 아이피를 호스트네임으로 바꾸는 과정이 수행된다. 말하자면 hostname lookup 과정이 수행되는데 접속이 많은 서버에서는 이 과정에서 상당히 많은 과부하가 발생한다. 그러므로 인증 부분을 호스트 기반이 아닌 아이피 기반으로 변경하고 위와 같은 옵션을 통해 hostname lookup 과정을 생략하도록 하면 눈에 띄는 성능 향상을 경험할 수 있다.

MySQL 메모리 튜닝

사실 데이터베이스 시스템의 튜닝은 메모리 관련 파라메터를 조정하는 것이 90% 정도를 차지한다고 할 수 있을 정도로 데이터베이스 시스템의 성능은 메모리관련 설정들에 크게 영향을 받는다. MySQL의 메모리 부분 튜닝은 사실 대부분 스토리지 엔진에 특화된 부분이다. 하지만 시스템 전체에 영향을 미치는 메모리 셋팅이 있는데 쓰레드 관련 메모리 셋팅과 쿼리 캐시관련 메모리 셋팅이 그러하다. 먼저 쓰레드 관련된 메모리 셋팅부터 살펴보자

쓰레드 관련 메모리 튜닝

MySQL은 커넥션 마다 하나의 쓰레드를 생성시켜서 요청을 처리하게 된다. 그러므로 쓰레드가 생성되는 시점에서 쓰레드에 메모리가 할당되며 많은 쓰레드가 생성되고 사라지면서 과부하가 발생하게 된다. 일반적인 시스템에서는 쓰래든 관련 파라메터들의 조정할 필요가 없지만 부하가 심한 서버에서는 모니터링 결과에 따라 이 셋팅을 바꾸어서 성능 향상을 이룰 수 있다. 먼저 현재 쓰레드와 관련된 상태를 알아보자.

```
mysql> SHOW STATUS LIKE '%THREAD%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Delayed_insert_threads  | 0     |
| Slow_launch_threads     | 0     |
| Threads_cached          | 0     |
| Threads_connected       | 1     |
| Threads_created         | 2     |
| Threads_running         | 1     |
+-------------------------+-------+
6 rows in set (0.00 sec)
```

위에서 볼 수 있는 항목 중 Threads_connected가 Threads_cached에 비해 매우 높다면 thread_cache_size를 높여줄 필요가 있다. Thread_cache_size는 지나치게 높여둘 필요는 없

으며 일반적으로 threads_connected의 피크 치 보다 약간 낮은 수치 정도를 설정해 두는 것이 좋다. 이를 통해 thread가 생성되고 소멸되면서 겪게 되는 메모리 및 각종 자원과 시간의 낭비를 줄일 수 있다. Thread와 관련하여 또 하나 셋팅할 수 있는 옵션은 thread_concurrency 옵션인데 이 옵션은 솔라리스 외의 시스템에서는 신경 쓸 필요 없으며 솔라리스에서는 CPU갯수 * 2의 값을 넣어주면 된다.

캐싱 관련 메모리 튜닝

MySQL 데이터베이스 시스템에서 메모리 관련 주요 셋팅들은 대부분 캐싱과 관련된 파라메터들이다. Buffer pool 또는 Key Cache 사이즈 그리고 Query Cache사이즈가 있는데 이 중 앞의 두 개는 InnoDB와 MyISAM의 핵심 파라메터 이므로 다음 회에 설명하게 되며 지금 살펴볼 항목은 바로 Query Cache에 관한 부분이다.

Query Cache란?

쿼리 캐시란 빈번하게 수행되는 Select관련 쿼리와 쿼리의 결과를 임시 저장하는 캐시 메모리이다. 데이터베이스 시스템에서 가장 시간이 많이 걸리는 작업은 바로 디스크를 액세스 하는 작업이다. 그러므로 디스크를 액세스 하는 작업을 줄이는 것이 가장 크게 성능을 올리는 일이 되는 것이다. 쿼리 캐시는 Select 쿼리에만 해당이 되며 쿼리 캐시를 사용하지 않게 되거나 쿼리 캐시에 저장된 내용을 초기화 하게 되는 경우는 아래와 같다.

- 데이터나 테이블 구조가 변경되었을 때
- 쿼리 캐시에 저장된 것과 다른 쿼리가 접수되었을 때
- 하나의 트랜잭션이 commit과 함께 마무리 되었을 때
- 쿼리가 내부적으로 임시 테이블을 생성해야 할 때

현실적으로 어려운 이야기지만 위와 같은 경우를 줄이면 줄일수록 쿼리 캐시의 사용률 및 효율을 높여 더 빠른 성능을 기대 할 수 있다.

Query Cache의 사용

먼저 현재 사용하고 있는 MySQL이 Query Cache를 지원하는 버전인지 확인하자.

```
mysql> SHOW VARIABLES LIKE 'HAVE_QUERY_CACHE';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| have_query_cache | YES   |
+------------------+-------+
1 row in set (0.00 sec)
```

만약 쿼리 캐시가 없는 MySQL버전을 사용하고 있다면 가능한 한 업그레이드를 하도록 한다. 가장 쉽고 가장 확실한 성능 향상 법은 최신 버전의 소프트웨어를 사용하는 것이라는 것을 잊지 말자. MySQL의 경우 특히 4.1 버전 이후로 많은 부분에 있어 성능 및 기능이 향상되었다. 아직도 3.x 버전을 사용하고 있다면 이번 기회에 업그레이드를 고려해 보는 것이 좋다.

쿼리 캐시를 지원하는 버전일 경우 query_cache_size=64M 와 같은 방식으로 정확한 쿼리 캐시 사이즈를 정해 주는 것만으로 쿼리 캐시를 사용하게 된다. 그리고 쿼리 캐시의 동작 방식을 정해주는 옵션으로 query_cache_type이라는 옵션이 있는데 0은 쿼리 캐시를 비 활성화 시키게 되고 1은 사용 가능한 모든 쿼리가 쿼리 캐시를 이용하게 되며 2는 쿼리 캐시를 이용하라고 정해주는 쿼리만 쿼리 캐시를 이용하게 된다. 2의 경우 쿼리 문 뒤에 SQL_CACHE라고 덧붙여 주면 된다.

Query Cache 최적화

데이터베이스 관련 모든 메모리 셋팅은 높다고 다 좋은 것이 아니다. 중요한 것은 균형 있는 값을 찾아 내는 것이다. 왜냐하면 쿼리 캐시와 MyISAM의 key cache, InnoDB의 buffer pool은 소중한 메모리 공간을 높고 서로 경쟁하는 관계이기 때문이다.

먼저 쿼리 캐시 사이즈를 결정해야 한다. 일반적으로 시스템 전체 메모리의 5%에서 10%사이를 사용하는 것이 보통이다. 일단 이 사이의 값으로 설정한 후 모니터링을 통해 쿼리 캐시 사용률이 100%에 가깝도록 하는 것이 좋다. 이를 모니터링 하는 가장 좋은 방법은 MySQL Administrator를 사용하는 것으로써 MySQL Administrator의 Health부분에서 쿼리 캐시의 효율을 지속적으로 모니터링 할 수 있기 때문이다.

다음으로 쿼리 캐시에서 받아들일 쿼리의 최대 크기를 설정하는 것이 필요하다. Query_cache_limit 옵션으로써 기본값으로 1MB가 설정되어 있으나 이는 너무 큰 값일 경우가 많다. 빈번하게 사용되는 쿼리의 용량이 어느 정도인지 살펴본 후 이보다 10%정도 높은 값을 설정해 주도록 하자.

MyISAM 스토리지 엔진 튜닝

데이터베이스의 최적화를 간단히 정의하자면 어떻게 하면 디스크의 사용을 더 줄이고 메모리를 효율적으로 사용하게 할 수 있는가 라고 할 수 있다. MyISAM 스토리지 엔진도 디스크의 사용을 줄이고 최대한 많은 정보를 적절한 메모리에 올려 사용하기 위한 여러가지 파러메터들을 가지고 있다. 그 중 가장 중요한 파라메터인 키 캐쉬(Key Cache)는 바로 이러한 목적을 위해 마련되었으며 그 중 인덱싱된 정보의 처리에 중점을 두고 있다. MyISAM 스토리지 엔진에서 키 캐쉬 튜닝은 전체 튜닝의 80% 이상을 차지한다고 할 수 있다. 먼저 키 캐쉬의 동작 방식 및 튜닝에 대해 자세히 알아보자.

MyISAM 키 캐쉬의 동작 방식

키 캐쉬란 하나 또는 그 이상의 테이블의 인덱스 정보를 저장할 수 있는 관리자가 정의하고 설정할 수 있는 메모리 블록을 말한다. 특정 테이블로부터 인덱스 정보를 생성, 수정 또는 가져오려고 할 때 MySQL은 먼저 관련된 정보를 메모리로부터 읽어올 수 있는지에 대해 캐쉬를 살펴보게 된다. 만약 캐쉬로부터 정보를 읽어올 수 있다면 키 캐쉬를 통해 읽기 또는 쓰기 작업을 매우 빨리 처리할 수 있게 되는것이다. 또는 만약 캐쉬로부터 정보를 얻을 수 없는 경우, 예를 들어 데이터 또는 인덱스 정보가 변경되었을 때 MySQL은 새로운 값을 디스크에 쓴 후 키 캐쉬에서 교체할 항목들을 정의한 후 새로운 정보로 대체하게 된다. 아래에서 키 캐쉬에 대한 몇가지 중요한 사항들을 정리해 보자.

- **모든 인덱스 블록은 명시적인 타임 스템프를 가진다** – MySQL은 어떠한 블록이 가장 오래되었는지 파악하기 위해 블록을 큐 방식을 사용해 저장한다. 이것은 키 캐쉬가 한정된 공간을 가지고 있고 새로운 블록을 저장하기 위해 현재 존재하는 블록을 밀어내야 하기 때문에 매우 중요하다.
- **블록은 캐쉬에 있는 동안 변화될 수 있다** – 만약 특정인의 성을 김에서 금으로 고치고 해당 컬럼이 인덱스 되어 있는 경우 키 캐쉬의 인덱스 블록 또한 수정된다. 그리고 언제든지 키 캐쉬의 블록은 제거될 수도 있기 때문에 수정된 정보를 바로 디스크에 저장하게 된다.
- **충분한 메모리를 가지고 있다면 MySQL이 기본으로 제공하는 키 캐쉬외에 여러 개의 추가적인 키 캐쉬를 가질 수 있다** – 이 기능은 MySQL 4.1 버전에 새로이 도입된 기능이다. 예를 들어 하나의 키 캐쉬는 매우 변화가 많은 트랜잭션이 주로 이루어지는 데이블들을 위한 용도로 사용하고 다른 키 캐쉬는 의사 결정을 위해 사용되는, 데이터의 변화가 별로 없는 테이블들을 위한 용도로 사용하는 것이다.
- **동시접속률을 증가시키기 위해 여러 개의 쓰래드가 하나의 캐쉬를 동시에 사용할 수 있다** – 물론 하나의 쓰레드가 캐쉬를 수정하는 경우 다른 쓰래드들은 수정이 완료될 때까지 아주 잠시 기다리게 된다.
- **키 캐쉬 블록의 교체를 위해 기본적인 큐 방식외에도 추가적인 매우 복잡한 알고리즘을 제공한다** – 이른바 '중간 삽입 전략(Midpoint Insertion Strategy)'이라고 하는 방식을 제공하는데 이는 사용률에 따라 구분하는 방식을 말한다.
- **위의 방식이 적용될 경우 키 캐쉬 블록의 교체를 위한 후보를 구분할 때 사용되는 빈도에 따라 핫리스트(Hot List)와 웜리스트(Warm List)로 구분한다.**

MyISAM 키 캐쉬의 설정

이제 키 캐쉬가 어떻게 동작하는지 알아보았으니 키 캐쉬를 어떻게 설정하고 동작방식을 컨트롤하는지 알아보자.

1. 다중 키 캐쉬를 구성할 것인지 만약 구성한다면 몇 개 정도를 구성할지 결정한다. 몇 개를 설정할지가 중요한 것이 아니라 각 키 캐쉬가 성능에 어떻게 영향을 미칠지 이해하는 것이 각 키 캐쉬를 설정하고 모니터링하며 튜닝하는 것 만큼 중요하다.

2. 각 키 캐쉬를 위해 적절한 값을 설정한다. 키 캐쉬를 만들 때 마다 아래와 같은 몇가지 항목에 대해 결정해야 한다.

- **키 캐쉬 버퍼의 메모리 사이즈를 정한다** – 'key_cache_block_size' 파라메터를 통해 각 키 캐쉬 버퍼가 얼마나 많은 매모리를 사용할지 설정한다. 1,024 바이트가 기본값이며 이 값은 대부분의 어플리케이션에 적합한 값이다. 현재 MySQL에서는 이 값의 변화가 큰 의미를 가지지 않으나 앞으로 이 파라메터가 좀더 중요한 역할을 하도록 바뀔 예정이다.
- **키 캐쉬에 메모리를 할당한다** – 가장 중요한 항목이다. 'key_buffer_size' 변수를 너무 작게 설정하는 것은 키 캐쉬가 주는 이득을 제대로 활용하지 못하는 것이며 너무 크게 주는 것은 소중한 메모리 공간을 낭비하는 것이된다. 각자의 환경이 모두 다르기 때문에 이것이 가장 좋다 라고 추천할 수 있는 값은 존재하지 않지만 일단 모든 키 캐쉬의 메모리 사용량이 전체 메모리의 5%~10%가 되도록 설정하는 것이 튜닝을 위한 바람직한 시작이 된다. 먼저 이 정도 값을 설정 한 후 모니터링 결과에 따라 값을 올리도록 하자.
- **중간 삽입 전략을 사용할지 결정하자** – 이 방식을 사용하지 않으려면 'key_cache_division_limit'을 100으로 설정하면 된다. 이 파라메터를 30으로 설정하면 웜리스트를 위해 30% 이상의 공간을 배정하지 않게 된다.

3. 인덱스와 원하는 캐쉬와 연결시켜 줍니다. CACHE INDEX 문을 통해서 키 캐쉬와 그 키 캐쉬를 사용할 인덱스를 지정해 줄 수 있습니다.

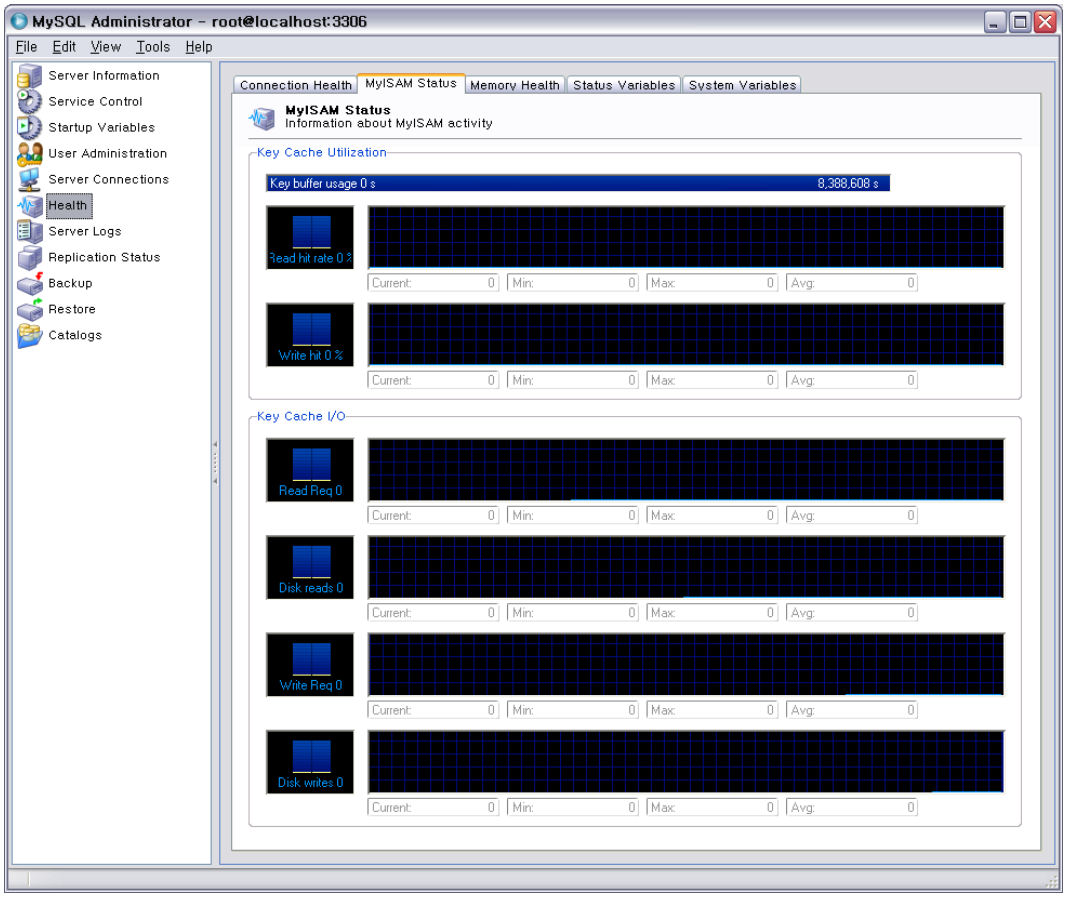
4. 키 캐쉬를 미리 로딩할지 결정합니다. MySQL은 키 캐쉬가 해당 레코드가 요청될 때 바로 로딩되거나 미리 로딩할지를 선택할 수 있습니다. 만일 미리 로딩시키기를 원한다면 LOAD INDEX문을 통하여 로딩 작업을 수행할 수 있습니다. LOAD INDEX문을 통해서 로딩할 메모리의 양을 'preload_buffer_size' 파라메터를 통해 설정해 줄 수 있습니다.

5. 키 캐쉬를 모니터링합니다. 키 캐쉬의 성능을 모니터링할 수 있는 방법은 여러가지가 있습니다만 MySQL Administrator를 사용하는 것이 가장 효율적입니다.

6. 키 캐쉬의 정보를 클리어 하는 방법으로는 MySQL 서버를 재시작하거나 key_buffer_size를 변경하는 것이 있습니다.

MyISAM 키 캐쉬의 모니터링과 튜닝

키 캐쉬의 설정이 끝났다면 이제 키 캐쉬의 모니터링을 위해 MySQL Administrator에 몇 가지 그래프를 추가해 보자. 아래 <화면1>은 키 캐쉬 모니터링을 위해 MyISAM Activity라는 페이지를 만들어 필요한 그래프 들을 추가해 놓은 모습이다.



<화면1> MyISAM 모니터링을 위해 구성해 놓은 MySQL Administrator의 모습

- **키 캐쉬 사용량** - 현재 얼마나 많은양의 키 캐쉬가 사용되고 있는지 그래프로서 표현하려면 계속적으로 변하는 값인 key_block_used와 각 block의 크기인 key_block_size를 곱한 값을 최대 키 캐쉬 사이즈인 key_buffer_size와 비교하여 알아낼 수 있다. Add graph한 후 value fomula에 [Key_blocks_used]*[key_cache_block_size]를 사용하고 Max fomular에 key_buffer_size를 입력한 후 bar graph로 설정하면 쉽게 사용량을 확인할 수 있다. 이 그래프를 모니터링 함으로써 기본적으로 키 캐쉬 사이즈가 적절히 설정되었는지를 확인할 수 있다. 만약 순식간에 그래프가 최고치에 도달한다면 키 캐쉬에 좀 더 많은 메모리를 할당할 필요가 있으며 반대로 지속적으로 저조한 사용량을 보인다면 값을 줄여줄 필요가 있다. 10분정도의 모니터링 결과로 값을 결정하는 것은 바람직하지 않으며 최소한 피크 타임에 1시간, 일반적인 사용시에 1시간 정도를 모니터링 한 후 적절한 값을 설정하는 것이 좋다.
- **키 캐쉬 읽기 적중율** - MySQL이 디스크가 아닌 키 캐쉬에서 얼마나 많이 읽어오는지를 파악하려면 아래의 공식을 통해 그래프를 만들 수 있다.

계산식 = 100-(^[key_read]/^[key_read_request])*100

그렇다면 이 그래프를 어떻게 해석할 수 있을까? 만약 지속적으로 90%이상의 적중율을 보인다면 키 캐쉬가 효율적으로 설정된 것이며 적중율이 지나치게 낮다면 키 캐쉬 메모리를 증가시켜 줄 필요가 있다. 하지만 데이터베이스가 엄청나게 크거나 여러 데이터를 골고루 많이 읽는 데이터베이스라면 아무리 많은 량의 키 캐쉬를 설정하여도 위와 같은 결과를 얻을 수 밖에 없다. 그리고 적중률이 99~100%를 기록한다면 그 역시 너무 많은 메모리가 키 캐쉬에 할당된 것이다. 이러한 경우에는 값이 떨어지기 시작하는 시점까지 지속적으로 키 캐쉬 메모리 할당량을 줄여야 한다.

- **키 캐시 쓰기 적중률** - 이 그래프는 키 캐쉬 쓰기 요청과 실제로 디스크에 쓰여지는 키 블록간의 상관관계를 볼 수 있다. 계산식은 아래와 같다.

  계산식 = 100-(^[key_writes]/^[key_write_requests[]*100

  일반적으로 키 캐시 쓰기 적중률은 읽기 적중률보다 상당히 낮은 값을 나타내는 것이 정상이다. 하지만 대용량 데이터 입력이나 큰 인덱스를 생성하는 경우 순간적으로 값이 높아질 수 있으니 주의해야 한다.
- **키 캐시 읽기 I/O** - 아래 두 그래프는 key_read_request 와 key_read의 횟수를 보여주는 그래프이다. 두가지 그래프를 비교함으로써 MySQL이 얼마나 많은 정보를 디스크가 아닌 키 캐쉬에서 읽어 오는지 알 수 있다.
- **키 캐쉬 쓰기 I/O** - 아래 두 그래프를 통해 키 캐쉬가 얼마나 많이 쓰이고 있는지 알 수 있다. 성공적인 키 캐쉬 쓰기 요청(key_write_request)와 디스크에 쓰여진 횟수(key_writes)를 비교하게 된다.

MyISAM 기타 메모리 관련 파라메터 정리

MyISAM에는 키 캐쉬 외에도 각각의 작업별로 영향을 미치는 여러가지 메모리 관련 파라메터들이 있다. 각종 파라메터들을 전체 스토리지 엔진에 영향을 미치는 파라메터와 각 쓰래드에만 영향을 미치는 파라메터로 나누어 표로 정리하였다.

범위	파라메터	설명
서버 전체가 공유함	Key_buffer_size	인덱스를 메모리에 저장하는 버퍼의 사이즈
	Table_cache	전체 쓰래드가 사용할 오픈 가능한 테이블의 수
	Thread_cache_size	재사용을 위해 캐싱될 쓰래드의 수
각 쓰래드별로 사용됨	myisam_sort_buffer_size	테이블 repair, Alter table, load data에 사용되는 버퍼 메모리의 사이즈
	join_buffer_size	조인을 위한 메모리 버퍼 사이즈
	record_buffer	순차적인 검색을 위해 사용되는 메모리 버퍼 사이즈
	record_rnd_buffer	Order by절을 사용하는 경우 디스크 사용을 피하기 위하여 사용하는 메모리 버퍼 사이즈
	sort_buffer	Order by와 group by에 사용되는 메모리 버퍼 사이즈
	tmp_table_size	Group by시 디스크를 사용하지 않고 임시 테이블을 만들기 위해 사용되는 메모리 사이즈

<표1> MyISAM의 각종 메모리 관련 파라메터

InnoDB 스토리지 엔진 튜닝

MySQL 3.x 버전에서 트랜잭션을 지원하기 위해 도입된 InnoDB 스토리지 엔진은 처음에는 MyISAM에 비하여 지나치게 느린 성능 등을 이유로 많이 사용되지 않았지만 지속적인 성능향상이 이루어지고 트랜잭션 지원에 대한 사용자들의 요구사항이 많아지면서 현재는 MyISAM과 거의 대등한 위치에 올라선 스토리지 엔진이다. InnoDB 스토리지 엔진의 특징이라면 무엇보다도 ACID를 완벽히 만족하는 트랜잭션의 지원과 로레벨 락킹(row level locking)이다. MySQL이 점차 다양한 기업환경에 사용되고 주요업무에도 도입되면서 이러한 특징에 따라 InnoDB는 점차 MySQL의 주요 스토리지 엔진으로 자리 잡을것으로 예상되며 MySQL AB에서도 InnoDB에 성능개선에 많은 노력을 기울여 왔다. 그 결과가 이번에 발표되는 MySQL 5.0 버전으로써 MySQL 5.0.7 베타버전을 기준으로 행해진 내부 벤치마크 테스트에서는 Select에서도 MyISAM보다 30% 이상 뛰어난 성능을 보여 MySQL AB의 내부에서도 큰 반향을 일으켰다. 그럼 앞으로 MySQL을 대표하는 스토리지 엔진이 될 InnoDB의 튜닝에 대하여 알아보자

InnoDB 버퍼 풀(Buffer Pool)의 설정

MyISAM의 튜닝에서 가장 큰 부분을 차지하는 것이 키 캐쉬라면 InnoDB에서 가장 큰 부분을 차지하는 것이 바로 버퍼 풀이다. 버퍼 풀의 사이즈를 조절하는 파라메터는 innodb_buffer_pool_size오 일반적으로 전체 시스템 메모리의 50%~80%정도를 설정하게 된다. 버퍼 풀의 사이즈를 결정하기 전에 먼저 중요한 몇가지 고려사항들을 짚어보자.

- **서버의 용도** - 만일 서버를 MySQL 전용서버로 사용한다면 좀더 마음놓고 버퍼 풀 사이즈를 올릴 수 있다. 하지만 같은 서버에서 웹서버 또는 어플리케이션 서버를 함께 운영한다면 꼭 필요한 만큼만 설정할 수 있도록 주의를 기울여야 한다.
- **사용가능한 시스템 메모리** - MySQL 전용서버라 할지라도 전체 물리적 메모리의 80% 또는 그 이상을 설정하는 것은 전반적인 시스템 성능에 큰 무리를 주게된다.
- **데이터베이스의 사용 유형** - 쓰기 중심의 데이터베이스 서버와 읽기 중심의 데이터베이스 서버는 버퍼 풀에 대한 요구사항이 크게 달라지게 된다. 이것이 바로 버퍼풀의 사용현황을 꾸준히 모니터링하는 것이 중요한 이유이다.
- **다중 스토리지 엔진의 사용** - MyISAM의 키 캐쉬와 InnoDB 버퍼 풀은 서로에게는 전혀 도움이 되지 않고 오히려 한정된 메모리를 가지고 경쟁하는 관계이다. 그러므로 각 스토리지 엔진 별로 사용빈도나 용도를 잘 따져서 설정해야 한다.

위의 사항들을 고려하여 버퍼 풀 사이드를 결정하였다면 이제 버퍼 풀과 관련된 몇가지 파라메터들을 조정하여야 한다. 버퍼 풀은 InnoDB의 인덱스와 데이터 정보를 캐싱하는 메인 메모리 캐쉬이기 때문에 아래 두 가지 사항은 매우 중요한 문제가 된다.

- 변화된 캐쉬 페이지는 정기적으로 디스크에 내려 써야 한다. 그렇지 않으면 예상치 못

한 서버 셧다운이 발생하였을 때 캐싱된 데이터는 유실되게 된다.

- 버퍼 풀에는 새로운 데이터 또는 인덱스 그리고 유저가 직접 데이터베이스를 조작하는 경우를 위한 공간을 반드시 남겨 두어야 한다.

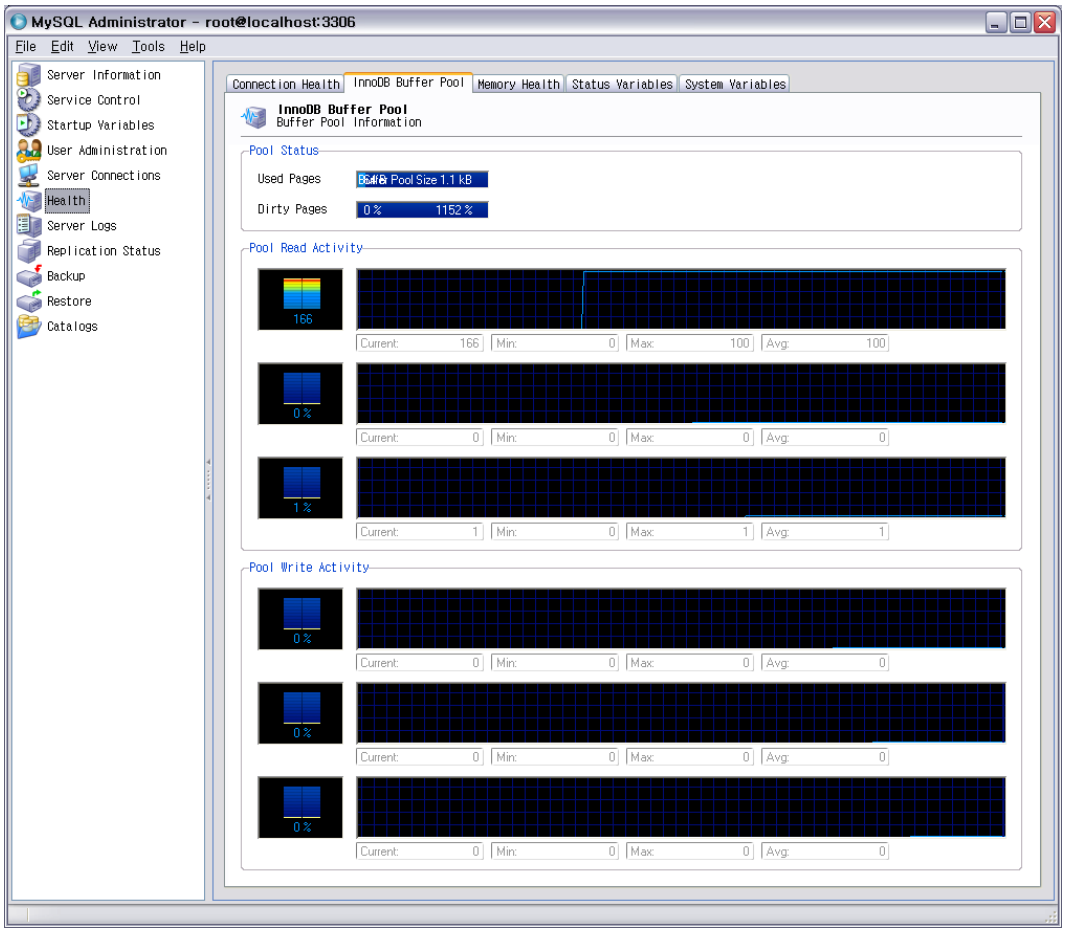
Innodb_max_dirty_page_pct는 위 두가지 요구사항을 충족시키는 것을 도와주는 파라메터이다. 0부터 100까지 설정할 수 있으며 설정한 수치에 따라 InnoDB 쓰래드가 디스크와 싱크하도록 지시하게 된다. 예를 들어 이 파라메터를 80이라고 설정하면 InnoDB는 전체 버퍼 풀의 80% 이상이 수정되었거나 필요없는 캐쉬 패이지 즉 더티 페이지를 가질 수 없도록 한다.

윈도우에서의 버퍼 풀 사이즈 한계를 뛰어 넘기 위한 파라메터 역시 제공하는데 이는 마이크로소프트에서 근년에 발표한 AWE(Address Windowing Extensions)라는 메모리 확장 기술을 사용하게 하여 준다. 이는 기존 윈도우 시스템의 메모리 한계인 4GB이상을 관리자가 사용할 수 있도록 해 주는 기술로서 innodb_buffer_pool_awe_mem_mb라는 파라메터이다. 이 파라메터를 통해 64GB까지의 메모리를 버퍼 풀로 사용할 수 있다.

그리고 버퍼 풀 이외에 메모리가 필요한 작업들을 하기 위해 설정하는 파라메터로서 innodb_additinoal_mem_pool_size라는 파라메터가 있는데 대부분의 경우 이는 기본값을 두어도 무방하다.

InnoDB 버퍼 풀의 모니터링 및 튜닝

앞선 MyISAM 스토리지 엔진에서 한것과 같이 InnoDB 버퍼 풀을 위해 MySQL Administrator에 커스텀 그래프들을 생성하여 모니터링 하고 이를 바탕으로 튜닝을 하도록 하자. 아래 <화면 2>는 버퍼 풀을 모니터링 하기 위해 새로운 페이지를 구성한 모습이다.



<화면2> InnoDB 모니터링을 위해 구성해 놓은 MySQL Administrator의 모습

- **버퍼 풀 사용량** - 전체 버퍼 풀 사이즈인 innodb_buffer_pool_pages_total 중에 현재 사용중인 버퍼 풀 사이즈인 innodb_buffer_pool_pages_data로 전체 버퍼 풀 사용량을 알아볼 수 있다. 시작하자마자 차지하고 있는 용량은 InnoDB가 내부적인 용도로 사용중인 것이다.
- **버퍼 풀 내의 더티 페이지 사용량** - 전체 버퍼 풀 사이즈인 innodb_buffer_pool_pages_total 중에 현재사용중인 더티 페이지 사이즈인 innodb_ _pool_pages_dirty로 전체 버퍼 풀 중 더티 페이지가 차지하는 용량을 알아볼 수 있다.
- **버퍼 풀 적중률** - 버퍼 풀이 용도에 맞게 효율적으로 설정되었는지 판단할 수 있는이중 가장 중요한 그래프이다. 그래프의 계산 공식은 아래와 같다.

  계산공식 = 100-(100*(^[innodb_pages_read]/^[innodb_buffer_pool_read_requests]))

  위 계산공식은 전체 버퍼 풀 읽기 요청 중 실제로 디스크에서 읽지 않고 버퍼 풀에서 읽은 횟수를 계산하는 것이다. 이 수치가 높다는 것은 버퍼 풀이 적절히 구성되어 동작중이라는 것이고 수치가 낮은 것은 버퍼 풀에서 실제로 자주 필요로 하는 정보를 찾을 수 없다는 것이다.
- **버퍼 풀 읽기 요청** - 이 그래프는 시시각각 변하는 버퍼 풀 읽기 요청을 모니터링 하기

위해 쓰인다. 언제 어떤 작업을 수행하기 위해 갑자기 버퍼 풀 읽기 요청이 증가하는지를 파악하는데 도움을 준다.

- **버퍼 풀의 연속적인 데이터 미리 읽기 활동 측정** – InnoDB는 복잡한 알고리즘으로 구현되어 있으며 어떠한 프로그램이 많은량의 연속적인 데이터 읽기 작업. 보통 전체 테이블 스캔이 되는 경우, 가 발생할지 미리 판단하게 된다. Innodb_buffer_pool_read_ahead_seq 의 상태를 지속적으로 모니터링 함으로써 파악할 수 있으며 계속해서 상승하는 것은 InnoDB가 더 많은 테이블 스캔을 하고 있다는 의미이다.
- **버퍼 풀의 랜덤한 미리 읽기 활동 측정** – InnoDB의 미리 읽기 알고리즘(read-ahead algorithm)은 연속적인 읽기 뿐만 아니라 비연속적인 읽기가 대량으로 발생하는것도 미리 예측하여 작업을 하게 된다. 이는 innodb_buffer_pool_read_ahead_rnd 의 상태를 모니터링 함으로써 파악 할 수 있다.
- **버퍼 풀에 대한 쓰기 요청** – 버퍼 풀이 얼마나 자주 변경 되는지 파악하려면 계속적으로 변하는 값인 innodb_buffer_pool_write_request 의 상태를 추적하면 된다.
- **플러시된 버퍼 풀 페이지** – 앞서 이야기한 바와 같이 MySQL은 정기적으로 버퍼 풀에 있는 페이지를 디스크로 씽크하는 작업을 한다. 이는 데이터의 순간적인 손실을 막기 위한 작업이다. Innodb_buffer_pool_pages_flushed 의 상태를 추적함으로서 모니터링이 가능하다.
- **버퍼 풀에 들어가기 위해 대기하고 있는 큐의 수** – 버퍼 풀의 용량이 충분하지 못하면 위의 플러시 이벤트가 발생할 때 까지 기다리게 되며 이러한 상황이 얼마나 발생하는지 카운트 하는 것은 매우 중요하다. 이러한 일이 자주 발생한다는 것은 현재 버퍼 풀의 사이즈가 요구량에 비해 작게 설정되어 있다는 뜻이기 때문이다. 이는 innodb_buffer_pool_wait_free를 추적함으로써 모니터링이 가능하다.

InnoDB 로그 파일(Log file)의 설정

InnoDB는 바이너리 로그 파일을 반드시 생성해야 하며 로그 파일과 관련된 설정 역시 성능에 큰 영향을 미치게 됩니다. MySQL 데이터베이스 관리자로서 InnoDB를 관리하려면 로그 파일과 관련하여 몇가지 결정해야 할 문제가 있습니다.

1. 첫번째로 몇 개의 로그 파일을 만들지 결정해야 합니다. 기본으로 설정되어 있으며 가장 최소의 값은 2입니다. 더 많은 로그파일을 만들려면 innodb_log_files_in_group 셋팅을 수정하면 됩니다.
2. 몇 개의 로그파일을 생성할지 결정되었다면 로그 파일의 용량을 결정해야 합니다. Innodb_log_file_size로 설정할 수 있고 기본값은 5MB로 되어 있으며 이는 작은 데이터베이스를 위해서는 충분한 값입니다. 하지만 기업에서 사용하기에는 훨씬 큰 용량이 필요합니다. 로그파일이 작게 설정되어 있을 경우 자주 메모리 기반의 버퍼 풀과 디스크 기반의 로그파일간의 체크포인트 생성 작업이 이루어 지게 됩니다. 이는 InnoDB성능을 매우 떨어뜨리는 주범이 되게 됩니다. 일반적으로 적당한 로그파일의 크기는 전체

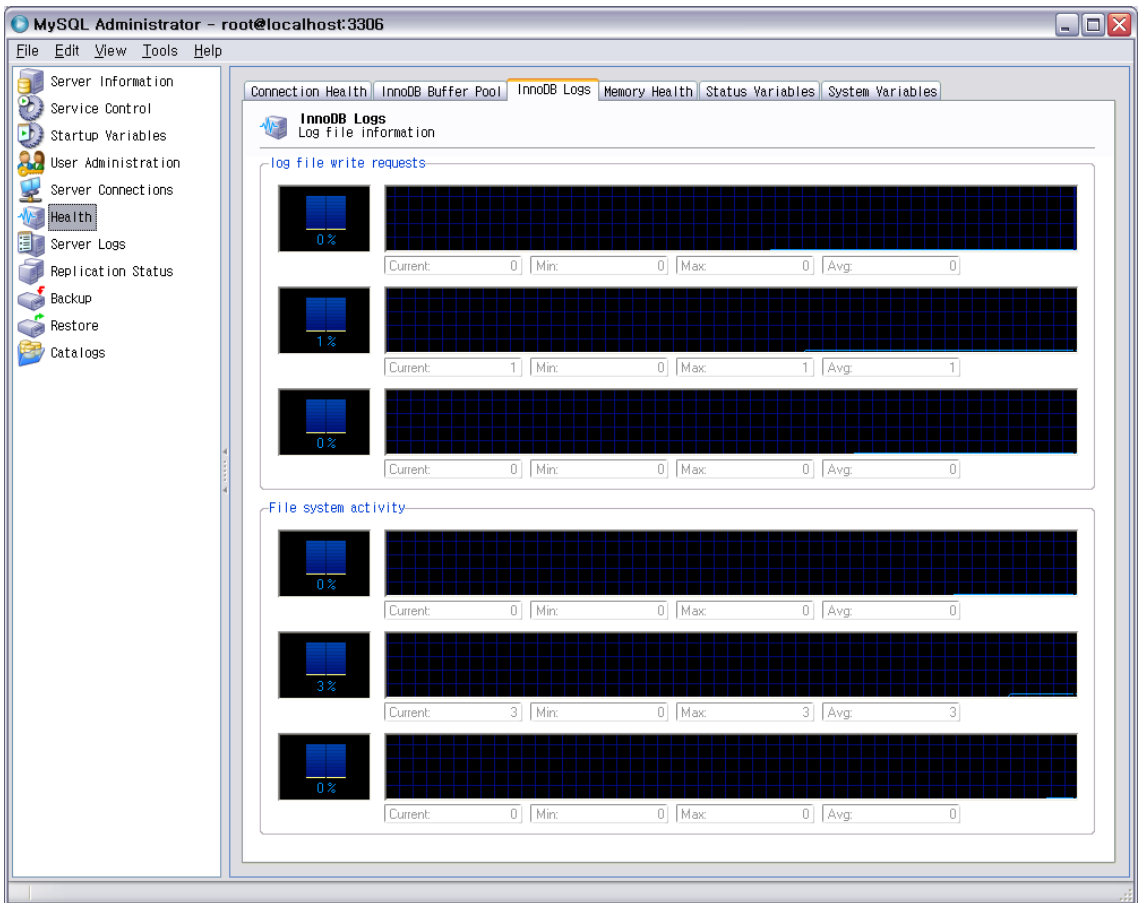
버퍼 풀 사이즈를 앞서 설정한 로그 파일의 개수로 나눈 값으로 합니다. 예를 들어 버퍼 풀 사이즈가 180MB이고 innodb_log_files_in_group을 3으로 설정하였다면 적당한 innodb_log_file_size는 60MB가 됩니다.

3. 로그파일의 개수를 몇 개로 하고 각각의 용량을 얼마로 할지를 정했다면 이제 innodb_log_buffer_size를 설정해야 합니다. 로그 버퍼는 디스크로 쓰기 전 메모리에 트랜잭션 정보를 담아두기 위한 버퍼 메모리의 크기를 말합니다. 1MB부터 8MB까지 설정할 수 있으며 이 용량이 클수록 디스크의 사용이 줄어들어 성능이 향상되지만 그만큼 갑작스러운 시스템 다운이 일어났을 때 손실되는 트랜잭션의 양도 늘어나게 됩니다. 충분히 많은 매모리와 특별한 사고 위험이 없다면 실제로는 크게 잡으면 잡을수록 좋습니다. 일반적으로 8MB를 추천합니다.
4. 마지막으로 COMMIT이 수행되었을 때 로그 버퍼와 파일이 어떤 작업을 하게 될지를 결정해야 합니다. Innodb_flush_log_at_trx_commit 셋팅은 아래와 같은 몇가지 옵션을 제공합니다.

- **바로 디스크에 쓰기** – 가장 안전한 방법이지만 가장 느린 방법입니다. 위의 셋팅을 1로 하게 되면 바로 디스크에 쓰게 됩니다.
- **조금 기다린 후 디스크에 쓰기** – 0또는 2를 설정할 수 있으며 0으로 설정할 경우 commit의 수행여부와 상관없이 매초 로그 버퍼가 디스크에 기록하게 되며 2로 설정하게 되면 commit이 수행되면 강제로 로그 버퍼를 로그 파일에 쓰게 만들지만 1초가 지나기 전엔 디스크에 저장되지 않게 하는 것입니다. 0으로 설정할 경우 성능은 많이 향상되지만 1초간의 트랜잭션 정보는 잃을 위험을 동반합니다.

InnoDB 로그 파일(Log file)의 모니터링 및 튜닝

MySQL은 5.0.2버전 이후로 서버에서 볼 수 있는 상태 정보를 엄청나게 많이 늘렸습니다. 이를 통해 사용자는 데이터베이스 내부의 상황을 더욱 자세히 파악할 수 있게 되었고 MySQL Administrator와 함께 세밀한 튜닝이 가능해 졌습니다. 새로 추가된 상태 정보들을 통해 로그 파일을 모니터링 하고 이를 통해 튜닝 포인트들을 점검해 봅니다.



<화면3> 로그 파일 모니터링을 위해 구성해 놓은 MySQL Administrator의 모습

- **로그 파일 쓰기 요청** – 계속해서 변하는 값인 innodb_log_write_requests를 추적함으로써 파악할 수 있습니다. 이를 모니터링 함으로써 어플리케이션이 얼마나 자주 로그 파일의 쓰기를 요청하는지 파악할 수 있습니다
- **로그 파일 쓰기 횟수** – 첫번째 그래프는 로그 파일 쓰기 요청을 추적하고 이 그래프에서 innodb_log_writes를 추적함으로써 얼마나 많은 실제적인 쓰기가 이루어 지는지 파악할 수 있습니다. 첫번째와 두번째 그래프 사이에 눈에 띄게 많은 랙이 존재한다면 이를 통해 로깅이 바틀넷 현상을 일으키고 있음을 알 수 있습니다.
- **로그 버퍼 웨이팅** – InnoDB는 로그 버퍼에 쓰기 위해 기다리는 경우가 발생할 때 마다 계속해서 innodb_log_waits의 값을 올리게 됩니다. 이러한 증상이 자주 발생하는 주원인은 실제의 요청량에 비해 로그 버퍼가 지나치게 작게 설정되어 있기 때문입니다. 이러한 증상은 특히 대용량의 정보를 로딩하거나 갑자기 트랜잭션이 증가할 때 발생합니다. 이러한 증상이 자주 발생한다면 innodb_log_buffer 셋팅을 증가시켜주는 것이 좋습니다.
- **로그파일과 운영체제간의 상호작용** – 변경된 정보는 결국 로그파일로 디스크에 저장되게 되고 이는 결국 운영체제의 파일시스템과 상호작용을 하게 됩니다. 그러므로 InnoDB와 파일시스템간에 벌어지는 일들을 모니터링 하는 작업도 매우 중요합니다. 두번째 그룹의 첫번째 그래프는 파일 시스템의 병목으로 로그파일에 쓰는 것이 늦춰지

는 것을 파악합니다. 이는 innodb_os_log_pending_writes를 통해 추적해 볼 수 있습니다. 그리고 다음 두가지의 그래프는 실제적인 디스크 쓰기를 담당하는 Fsync() 함수를 innodb_os_log_fsyncs를 통해 추적함으로써 실제적인 쓰기 요청이 얼마나 일어나는지 그리고 fsync()의 실행이 늦춰지는 경우는 얼마나 있는지를 innodb_os_log_fsyncs_pending을 통해 파악함으로써 현재 innodb 시스템이 얼마나 I/O의 병목현상 없이 잘 실행되고 있는지를 파악할 수 있습니다.

낚시하는 방법을 배우자

공개된 여러 튜닝 정보들을 보면 각각의 튜닝이 어떤 내부 구동원리를 통해 이뤄지는가를 설명하기 보다는 각 항목에 대한 구체적인 추천 수치만을 제시함으로써 일회성의 수동적인 튜닝에 그치는 경우가 많다. 그래서 이 글에서는 구체적인 수치를 제시하기 보다는 각 항목들이 데이터베이스 내부에서 어떤 역할을 하며 수치를 변경하는 것이 어떤 결과를 낳게 되는지 그리고 모니터링을 통해 이를 실제로 확인할 수 있게 하는 것을 주로 살펴보았다. 설명이 미흡하거나 따라하기 힘든 부분도 많아 아쉬움이 남지만 MySQL의 활용에 작은 도움이라도 되었으면 한다.

참고자료

MySQL 홈페이지: www.mysql.com
MySQL Design and Tuning (MySQL Press)