대량 트래픽 처리를 위한 RDBMS 모델링에 대한 연구

유기정, 김응모
성균관대학교 정보통신대학원
e-mail:kijung3@gmail.com, ukim@skku.edu

A Study of RDBMS Modeling for Massive Traffic Handling

Ki-Jung Yoo, Ung-Mo Kim
Graduate School of Information and Communications, Sungkyunkwan University

요 약

최근 소셜 네트워크 서비스가 확산되면서 대량 트랜잭션 환경에서의 RDBMS 성능에 대한 관심이 높아지고 있다. 본 논문에서는 대량 트랜잭션 환경에서 DBMS가 SQL문을 처리하면서 발생시키는 I/O의 특징을 고려하여 데이터의 쓰기 블록 수와 트랜잭션 간에 발생하는 배타적 Lock의 빈도를 최소화시키기 위한 모델링을 제안하고 일반적인 모델링과 성능 비교 실험을 하였다. 실험 분석 결과 DBMS의 트랜잭션 처리량이 많고 트랜잭션 간의 교착 빈도가 높게 발생할수록 일반적인 모델링보다 제안하는 모델링에서의 SQL문 처리 성능이 우수하였다.

1. 서론

최근 소셜 네트워크 서비스를 제공하는 기업들은 기존과는 다른 대량의 트래픽에 의한 시스템 부하를 경험하게 되면서 이를 해결하기 위한 노력을 지속적으로 하고 있다.

부하를 해소시키기 위한 방법으로는 시스템을 구성하는 서버들을 최적화시키고 역할을 분리하거나 서버의 수량을 늘려 트래픽을 분산시키는 방법이 있다. 하지만 데이터베이스 서버는 시스템을 구성하는 다른 서버들에 비해 서버의 역할을 분리하여 데이터를 분산시키기가 어렵고 하드웨어를 구매하거나 증설시키는 비용이 많이 들기 때문에 데이터베이스의 성능에 대한 관심은 높아지고 있다.

대부분 시스템에서는 조회의 성능을 중심적으로 튜닝을 하여 성능을 개선할 수 있지만 소셜 네트워크 서비스와 같이 대량의 트래픽을 처리해야 하는 시스템에서는 중복적인 데이터 조회를 CACHE 서버가 담당하여 응답해줌으로써 조회보다 입력/수정/삭제에 대한 처리 빈도가 높아지게 되므로 입력/수정/삭제의 성능을 개선시키기 위한 튜닝이 필요하다.

본 논문에서는 소셜 네트워크 게임 서비스와 같이 조회보다 입력/수정/삭제 SQL문의 빈도가 높은 특징을 가지는 데이터베이스에서 대량 트랜잭션을 처리 할 수 있도록 SQL문의 I/O 특징에 의한 모델링을 제안하고 메모리와 디스크의 쓰기 I/O와 트랜잭션 간에 발생하는 Lock의 빈도를 감소시킴으로써 성능을 얼마나 향상시킬 수 있는지 실험을 통하여 확인하고자 한다.

2. 대량 트랜잭션 처리를 위한 모델링 개선 방안

2.1 소셜 네트워크 게임의 일반적인 정규화 모델링

최근 소셜 네트워크 요소를 가지고 있는 스마트폰에 서비스되는 게임 컨텐츠들은 오픈 마켓을 통하여 컨텐츠는 무료로 다운로드 받아서 이용할 수 있으나 유저가 게임 플레이를 하기 위해서는 일정시간이 흘러야 충전되는 포인트를 사용하여야 하며 이 포인트는 결재를 통하여 충전하거나 주변에 동일한 컨텐츠를 이용하는 지인들 간에 선물 시스템을 통하여 충전되기도 한다.

게임을 이용하는 유저는 비용과 시간을 최소화하여 게임을 즐기기 위해서 보다 많은 친구들을 게임에 초대하고 친구들에게 도움을 요청해야만 한다. 또한 게임 플레이 후 획득한 점수나 아이템을 친구들에게 자랑하는 기능을 통하여 함께 게임을 즐기는 친구들과 경쟁을 유도하게 되므로 소셜 네트워크 게임 서비스의 DBMS가 처리해야 하는 트랜잭션양은 다른 서비스들보다 많아지게 된다.

소셜 네트워크 게임에서 유저들은 게임을 플레이하기 위하여 로그인 한 후 마지막 접속한 시간으로부터 현재까지 시간에 대한 누적 포인트를 충전 받게 되며 친구로 등록된 유저에게 포인트 선물을 보낸 후에 자신의 포인트가 모두 소진될 때까지 친구에게 도움을 받으며 게임을 플레이하게 된다. 이러한 유저들의 행동에 사용되는 테이블을 일반적인 정규화 모델링으로 표현하면 표 1과 표 2와 같이 유저 테이블과 친구 리스트 테이블로 나타낼 수 있다. 표 3은 다음의 가정을 기반으로 유저의 행동에 의해 게임에서 실행되는 SQL문을 정의하고 실행 빈도를 가중치로

나타내었으며 유저들 간에 선물을 주고받는 행동은 제외하였다.

가정 1: 유저들은 로그인 후에 친구들의 정보를 1회 조회한다.
가정 2: 유저들은 로그인 후 평균적으로 게임을 10회 플레이한다.
가정 3: 평균적으로 게임 플레이를 300회하면 레벨이 상승한다.

<표 1> 유저 테이블

필드명	타입	사이즈	설명
ID	int	4byte	유저의 고유 식별 번호
Name	char(50)	50byte	게임에 표기되는 유저의 닉네임
Level	smallint	2byte	레벨/일정 Exp가 충족이 되면 1씩 증가함
Cash	bigint	8byte	캐시 머니/결제를 통하여 획득할 수 있는 재화
Gold	bigint	8byte	게임 머니/게임 플레이마다 획득 할 수 있는 재화
Exp	int	4byte	경험치/게임 플레이마다 획득
SPoint	smallint	2byte	게임 플레이에 필요한 포인트/일정 시간이 지나면 증가하며 1Spoint를 소모하여 게임을 플레이할 수 있음
BestScore	bigint	8byte	유저의 한주의 최고 기록
LastLogin	datetime	8byte	마지막 로그인 시간
RegDate	datetime	8byte	가입 날짜

<표 2> 친구 리스트 테이블

필드명	타입	사이즈	설명
ID	int	4byte	유저의 고유 식별 번호
FriendID	int	4byte	친구의 고유 식별 번호
GiftDate	datetime	8byte	마지막 Spoint 선물 날짜

<표 3> 게임에서 실행되는 SQL문

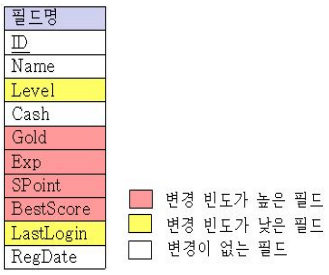
번호	가중치	설명	SQL 문
-	0	로그인	생략
1	30	포인트 충전 로그인 시간 갱신	UPDATE [유저 테이블] SET Spoint = Spoint + 10, LastLogin = GETDATE() Where ID = ?
2	330	유저정보 조회	SELECT Name, Level, Cash, Gold, Exp, SPoint, BestScore, LastLogin From [유저 테이블] Where ID = ?
3	300	포인트 차감	UPDATE [유저 테이블] SET Spoint = Spoint - 1 Where ID = ?
4	300	유저정보 갱신	UPDATE [유저 테이블] SET Gold = Gold + 1000, Exp = Exp + 100 Where ID = ?
5	300	최고 점수 갱신	UPDATE [유저 테이블] SET BestScore = ? Where ID = ? AND BestScore < ?
6	1	Level 상승	UPDATE [유저 테이블] SET Level = Level +1 Where ID = ?
7	330	친구 정보 조회	SELECT a.Name, a.Level, b.GiftDate FROM [유저 테이블] as a, [친구 리스트 테이블] WHERE a.ID = b.FriendID AND b.ID = ?

동시에 게임을 플레이하는 유저가 많아질수록 유저 테이블은 빈번하게 실행되는 표 3의 3 ~ 5번 UPDATE SQL문에 의해 ID에 해당하는 레코드가 배타적 Lock이 설정되고 7번 SQL문은 자신과 친구 관계를 맺고 있는 유저의 정보를 범위 조회하게 되므로 블로킹으로 대기할 확률이 높아지게 된다. 또한 빈번하게 변경되는 Gold, Exp, Spoint, BestScore 필드에 의해 버퍼 캐시의 Dirty 블록 수가 증가되면서 Check Point로 디스크의 데이터 파일과 동기화되어야 하는 블록 수도 증가한다. Check Point에 의해 실행되는 디스크 I/O는 랜덤 쓰기 I/O이므로 대량 트랜잭션을 처리해야 하는 데이터베이스 시스템에서는 쓰기 I/O의 수와 용량이 클수록 디스크 병목으로 인한 성능 저하가 발생할 확률이 높아지게 된다.

2.2 제안하는 SQL문의 I/O 특징에 의한 모델링

소셜 네트워크 게임의 일반적인 모델링에서 대량 트랜잭션을 처리하는데 있어서 불필요한 디스크 쓰기 I/O의 증가와 Lock에 의한 블로킹으로 하드웨어의 자원이 낭비될 수 있음을 2.1절에서 살펴보았다. DBMS의 한정된 하드웨어 자원이 효율적으로 활용될 수 있도록 SQL문의 I/O 특징을 분석하여 모델링함으로써 SQL문의 성능을 개선하여 DBMS가 처리할 수 있는 트랜잭션 양을 증가시킬 수 있다.

그림 1은 표 1의 유저 테이블에 정의된 필드들의 변경 빈도를 보여준다. 표 3의 1 ~ 6번 SQL문은 자신의 트랜잭션에만 영향을 주나 7번 SQL문은 다른 유저의 Name, Level 정보를 조회하기 위하여 유저 테이블을 JOIN하는 과정에서 다른 트랜잭션에서 빈번하게 변경되는 필드에 의해 배타적 Lock의 영향을 받게 되므로 7번 SQL문은 불필요한 대기 시간이 발생하게 된다.



(그림 1) 유저 테이블의 필드 변경 빈도

유저 테이블과 친구 리스트 테이블의 관계에서 불필요한 블로킹이 최소화될 수 있도록 SQL문의 I/O 특징을 고려하여 유저 테이블을 리모델링하면 필드에 변경이 거의 발생하지 않는 표 4의 테이블과 필드의 변경이 빈번하게 발생하는 표 5의 테이블로 분리할 수 있으며 표 6은 테이블의 분리로 인하여 변경되는 SQL문을 보여준다.

<표 4> 변경이 적은 유저 테이블

필드명	타입	사이즈	설명
ID	int	4byte	유저의 고유 식별 번호
Name	char(50)	50byte	게임에 표기되는 유저의 닉네임
Level	smallint	2byte	레벨/일정 Exp가 충족이 되면 1씩 증가함
Cash	bigint	8byte	캐시 머니/결제를 통하여 획득할 수 있는 재화
RegDate	datetime	8byte	가입 날짜

<표 5> 변경이 많은 유저 테이블

필드명	타입	사이즈	설명
ID	int	4byte	유저의 고유 식별 번호
Gold	bigint	8byte	게임 머니/게임 플레이마다 획득 할 수 있는 재화
Exp	int	4byte	경험치/게임 플레이마다 획득
SPoint	smallint	2byte	게임 플레이에 필요한 포인트/일정 시간이 지나면 증가하며 1Spoint를 소모하여 게임을 플레이할 수 있음
BestScore	bigint	8byte	유저의 한주의 최고 기록
LastLogin	datetime	8byte	마지막 로그인 시간

<표 6> 게임에서 실행되는 변경된 SQL문

번호	가중치	설명	SQL 문
-	0	로그인	생략
1	30	포인트 충전 로그인 시간 갱신	UPDATE [변경이 적은 유저 테이블] SET Spoint = Spoint + 10, LastLogin = GETDATE() Where ID = ?
2	330	유저정보 조회	SELECT a.Name, a.Level, a.Cash, b.Gold, b.Exp, b.SPoint, b.BestScore, a.LastLogin From [변경이 적은 유저 테이블] as a, [변경이 많은 유저 테이블] as b Where a.ID = b.ID AND a.ID = ?
3	300	포인트 차감	UPDATE [변경이 많은 유저 테이블] SET Spoint = Spoint - 1 Where ID = ?
4	300	유저정보 갱신	UPDATE [변경이 많은 유저 테이블] SET Gold = Gold + 1000, Exp = Exp + 100 Where ID = ?
5	300	최고 점수 갱신	UPDATE [변경이 많은 유저 테이블] SET BestScore = ? Where ID = ? AND BestScore < ?
6	1	Level 상승	UPDATE [변경이 적은 유저 테이블] SET Level = Level +1 Where ID = ?
7	330	친구 정보 조회	SELECT a.Name, a.Level, b.GiftDate FROM [변경이 적은 유저 테이블] as a, [친구 리스트 테이블] WHERE a.ID = b.FriendID AND b.ID = ?

유저 테이블이 분리되면서 실행 빈도가 높은 2번 SQL문에 JOIN이 추가되어 2번 SQL문에 대한 처리 비용이 기존보다 증가하게 되지만 다음의 두 가지 성능 이점으로 허용이 가능한 수준으로 본다.

첫 번째로 친구의 정보를 조회하기 위하여 실행되던 7번 SQL문은 Gold, Exp, SPoint, BestScore 필드가 빈번하게 변경되면서 발생하던 다른 트랜잭션의 배타적 Lock으로부터 블로킹을 회피할 수 있게 되며 낮은 빈도로 변경되는 Level 필드에 의한 배타적 Lock의 영향만 받게 되므로 블로킹에 의한 대기 빈도와 평균 대기 시간이 감소하게 되어 SQL문의 처리 성능이 향상된다.

두 번째로 기존의 유저 테이블은 레코드의 크기가 102byte이었으나 유저 테이블이 분리되면서 각 테이블의 레코드 크기는 72byte와 34byte가 된다. 따라서 빈번한 업데이트로 변경되어야 하는 레코드의 크기가 102byte에서 34byte로 축소되어 Check Point에 의해 디스크의 데이터 파일의 블록과 동기화해야 하는 버퍼 캐시의 블록 수가 약 1/3로 줄어들게 되므로 읽기 I/O보다 상대적으로 처리 속도가 느린 메모리나 디스크의 쓰기 I/O의 횟수와 용량이 줄어들게 되므로 디스크 I/O 호출로 인한 대기 시간이 감소하여 하드웨어의 입출력에 대한 성능이 향상된다.

3. 모델링에 따른 SQL문 처리 성능 실험

3.1 실험환경

본 실험은 일반적인 정규화 모델링과 제안하는 SQL문의 I/O 특징에 의한 모델링의 성능을 확인하기 위하여 Microsoft의 Windows Server 2008 R2 OS 환경에서 SQL Server 2008 R2 Enterprise DBMS 서버와 부하를 발생시키기 위한 서버로 구성하였다.

DBMS 서버에는 표 1의 유저 테이블과 표 2의 친구 리스트 테이블을 생성한 일반적인 모델링 데이터베이스와 표 4와 표 5의 유저 테이블과 표 2의 친구 리스트 테이블을 생성한 제안하는 모델링 데이터베이스를 생성하였다. 각 데이터베이스에는 표 3과 표 6의 SQL문이 유저들에 의해 실행되는 형태와 유사하게 실행될 수 있도록 유저고유번호와 SQL문의 실행 순서를 변수로 정의하여 무작위로 실행되면서 SQL문의 실행 비율이 가중치에 의해 유지되도록 저장 프로시저를 구현하였다. 표 7은 각 테이블의 기본키 범위와 데이터 수를 나타내며 데이터는 동일하다.

<표 7> 테이블별 데이터 수

데이터베이스	테이블	기본키 범위	데이터 수
일반적인 모델링	유저 정보	1 ~ 500000	500000
	친구 리스트1	1 ~ 100000	3000000
	친구 리스트2	1 ~ 300000	9000000
	친구 리스트3	1 ~ 500000	15000000
제안하는 모델링	변경이 적은 유저 정보	1 ~ 500000	500000
	변경이 많은 유저 정보	1 ~ 500000	500000
	친구리스트1	1 ~ 100000	3000000
	친구리스트2	1 ~ 300000	9000000
	친구리스트3	1 ~ 500000	15000000

부하 발생 서버에서는 SQL문의 성능 편차를 줄이기 위해서 Microsoft의 OSTRESS를 이용하여 DBMS에 생성한 저장 프로시저를 50만번 반복 실행한 후 누적 값에 대한 성능 통계를 SQL Server에서 제공하는 동적 관리 뷰인 dm_exec_query_stats를 이용하여 측정하였다.[4]

본 실험에서는 실제 서비스에서 유저들이 동시에 발생시키는 대량 트랜잭션을 실험 환경으로 구현하는데 한계가 있으므로 표 8의 실험 시나리오 같이 트랜잭션 간에 교착의 빈도를 데이터의 수에 의해 단계별로 조정하였으며 트랜잭션 처리량이 증가되면서 변경되는 성능의 비교를 위해서 DBMS 연결 수를 100, 200, 300, 400, 500으로 증가시키며 실험을 하였다.

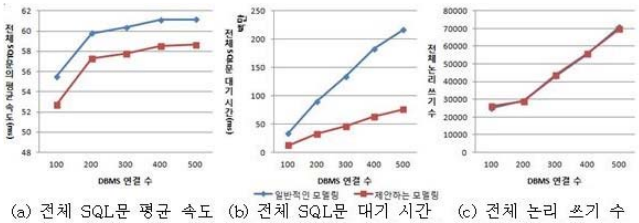
<표 8> 실험 시나리오

단계	목적	유저 데이터 수	친구 데이터 수
1	높은 빈도의 트랜잭션 교착	10만	300만
2	중간 빈도의 트랜잭션 교착	30만	900만
3	낮은 빈도의 트랜잭션 교착	50만	1500만

3.2 실험 결과 측정

1) 시나리오 1단계 실험

그림 2는 실험 시나리오 1단계의 성능 평가 실험 결과이다. 1단계 실험은 SQL문을 처리하는데 사용되는 블록의 수는 적으나 조회와 변경되는 데이터의 밀집도가 높으므로 트랜잭션 간의 교착이 높게 발생하는 특징을 가지고 있다. 실험 결과, (a)와 같이 전체 SQL문의 평균 실행 속도의 성능은 일반적인 모델링보다 제안하는 SQL문의 I/O 특징에 의한 모델링이 동일한 트랜잭션 처리량에서 2 ~ 3ms 향상 되었으며 (b)와 같이 트랜잭션 처리량이 증가하면서 SQL을 처리하는 동안 발생하는 전체 대기 시간이 일반적인 모델링에 비해 제안하는 모델링에서 적은 증가량을 보였다. 데이터의 블록 수는 실험 시나리오 중 가장 적으므로 SQL문을 처리하면서 변경되는 블록 수는 (c)와 같이 두 모델링에서 거의 동일하였다.



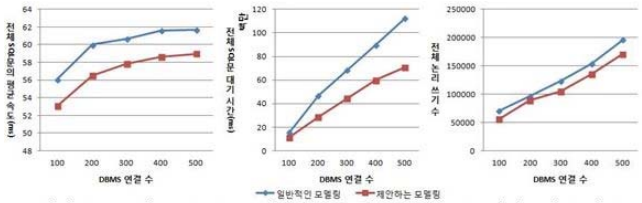
(a) 전체 SQL문 평균 속도 (b) 전체 SQL문 대기 시간 (c) 전체 논리 쓰기 수

(그림 2) 시나리오 1단계 실험 결과

2) 시나리오 2단계 실험

그림 3은 실험 시나리오 2단계의 성능 평가 실험 결과이다. 2단계 실험은 1단계 실험보다 사용되는 블록의 수가 많고 트랜잭션 간의 교착 빈도는 낮으며 3단계 실험보다 사용되는 블록의 수가 적고 트랜잭션 간의 교착 빈도는 높은 특징을 가지고 있다. 2단계 실험에서는 (a)와 같이 일반적인 모델링보다 제안하는 모델링의 평균 SQL문 처리 시간에 대한 성능이 2 ~ 3ms 향상되었다. 트랜잭션 처리량이 증가하면서 두 모델링에서 SQL문을 처리하는데

발생한 전체 대기 시간은 (b)와 같이 일반적인 모델링보다 제안하는 모델링이 적게 증가하는 것을 확인하였으며 1단계 실험보다 트랜잭션 간에 교착이 감소하면서 두 모델링 간의 전체 대기 시간에 대한 증가폭은 줄어들었다. 일반적인 모델링에 비해 제안하는 모델링에서 (c)와 같이 동일한 트랜잭션 처리량에서 변경되는 블록의 수가 적었다.

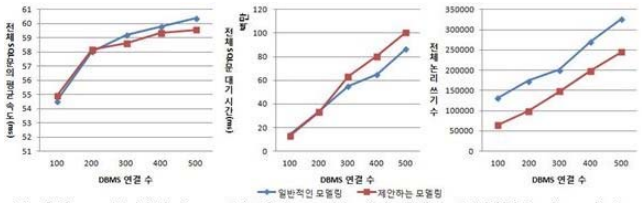


(a) 전체 SQL문 평균 속도 (b) 전체 SQL문 대기 시간 (c) 전체 논리 쓰기 수

(그림 3) 시나리오 2단계 실험 결과

3) 시나리오 3단계 실험

그림 4는 실험 시나리오 3단계의 성능 평가 실험 결과이다. 3단계 실험은 다른 단계의 실험에 비해 SQL문을 처리하는데 사용되는 블록의 수가 가장 많으며 조회와 변경되는 데이터의 밀집도가 가장 낮으므로 트랜잭션 간의 교착이 적게 발생하는 특징을 가지고 있다. (a)와 같이 트랜잭션의 처리량이 적은 DBMS의 연결 수 100에서 일반적인 모델링이 제안하는 모델링에 비해 SQL문의 처리 속도에 대한 성능이 우수하였으나 DBMS의 연결 수 200부터 트랜잭션 처리량이 증가하면서 일반적인 모델링에 비해 제안하는 모델링의 성능이 향상되었다. 전체 대기 시간은 1 ~ 2단계에 비해 트랜잭션의 교착의 빈도가 줄어들면서 (b)와 같이 일반적인 모델링보다 제안하는 모델링에서의 대기 시간이 높았다. 또한 (c)와 같이 변경되는 블록의 수는 일반적인 모델링에 비해 제안하는 모델링에서 적게 변경되었다.



(a) 전체 SQL문 평균 속도 (b) 전체 SQL문 대기 시간 (c) 전체 논리 쓰기 수

(그림 4) 시나리오 3단계 실험 결과

3.3 실험 결과 비교 분석

본 실험을 통하여 제안하는 SQL문의 I/O 특징에 의한 모델링은 DBMS의 트랜잭션 처리량이 많고 트랜잭션 간의 교착 빈도가 높게 발생할수록 일반적인 모델링에 비해 SQL문 처리 속도의 성능이 향상되는 것을 확인할 수 있었으며 트랜잭션 처리량이 증가하면서 발생하는 트랜잭션 간의 교착에 의한 대기 시간이 일반적인 모델링에 비해 적게 증가하는 것을 확인하였다. 하지만 데이터의 양이 많고 트랜잭션 처리량이 적은 대용량 데이터베이스에는 본 논문에서 제안하는 모델링은 성능을 저하시키게 되므로 적절하지 않은 모델링으로 확인하였다. 결론적으로 트랜잭션 처리량이 많고 트랜잭션 간의 교착이 높게 발생하는 소셜 네트워크 서비스에서 제안하는 SQL문의 I/O 특징을 고려한 모델링으로 DBMS의 성능을 향상 시킬 수 있으며 동일한 하드웨어에서 보다 많은 양의 트랜잭션을 처리할 수 있다고 판단한다.

4. 결론

본 논문에서는 RDBMS가 빠른 속도로 발전하는 하드웨어 성능에도 불구하고 대량 트래픽 처리에 대한 어려움이 발생하는 원인을 RDBMS의 데이터 처리 과정에서 구조적으로 발생하는 블록 수의 증가와 배타적 Lock으로 인한 병목으로 보고 소셜 네트워크 게임의 일반적인 모델링의 예제를 통하여 이를 최소화할 수 있도록 SQL문의 I/O 특징을 고려한 모델링을 제안하고 일반적인 정규화 모델링과의 성능 실험을 통하여 비교하고 분석 하였다.

실험 결과, DBMS의 트랜잭션 처리량이 많고 트랜잭션 간의 교착 빈도가 높을수록 SQL문의 조회/입력/수정/삭제에서 발생하는 I/O의 비율과 테이블간의 관계를 고려하여 모델링을 함으로써 일반적인 모델링에 비해서 SQL문의 성능을 향상시킬 수 있음을 확인하였으며 변경되는 블록의 최소화와 트랜잭션 처리량의 증가로 발생하는 트랜잭션 간의 교착에 의한 대기 시간의 감소로 디스크 쓰기 I/O와 CPU 사용이 줄어들게 되므로 동일한 하드웨어에서 DBMS는 보다 많은 양의 트랜잭션을 처리할 수 있을 것으로 판단된다.

본 논문에서는 실제 소셜 네트워크 서비스에서 유저들이 만들어 내는 복잡한 대량 트랜잭션을 실험 환경으로는 재연하기 어렵기 때문에 유저의 데이터양을 축소하고 Microsoft의 OSTRESS를 이용하여 제한된 인위적인 트래픽의 환경에서 실험을 하여 결론을 도출하였다. 향후 실제 소셜 네트워크 서비스에서 유저에 의해 만들어지는 복잡한 대량 트랜잭션 환경에서의 연구가 필요하다.

참고문헌

[1] 한국데이터베이스진흥원 편집부, SQL 전문가 가이드 2010 Edition, 한국데이터베이스진흥원, 2010.

[2] http://www.dbguide.net

[3] http://msdn.microsoft.com/ko-kr/library/ms187024(v=sql.105).aspx

[4] http://msdn.microsoft.com/ko-kr/library/ms189741.aspx

[5] 디엔에이(DeNA), 모바게를 지탱하는 기술, 제이펍, 2013.

[6] Baron Schwartz, Peter Zaitsev, Vadim Tkachenko, Jeremy D. Zawodny, and Arjen Lentz, MySQL 성능 최적화, 위키북스, 2010.