C-C Connector : 공간 데이터 베이스에서 서버 처리 비용의 분산을 위한 미들웨어

*강 동 재, 정 보 흥, 박 동 선, 배 해 영
*인하대학교 전자계산공학과
baramsori72@cjdream.net

C-C Connector : The Middle Ware for distributing the Cost of Server In Spatial Database

*Dong-Jae Kang, Bo-Heung Chung, Dong-Seon Park, Hae-Young Bae
*Dept. of Computer Science & Engineering Inha University

요 약

공간 데이터와 같은 대용량의 데이터를 처리하는 시스템이나 다수의 클라이언트의 요구가 발생하는 시스템에서는 데이터에 대한 처리 비용 및 전송 비용으로 인한 서버 사이드의 병목 현상 및 질의 처리 속도의 저하라는 문제점을 갖는다.

본 논문에서는 이러한 문제점을 개선하기 위한 방법으로 미들웨어인 C-C Connector의 Method 및 프로토콜을 제안한다. 제안하는 C-C(Client To Client) Connector는 질의의 분석과 CIT(Client Information Table)의 검색을 통하여 인접한 클라이언트들의 캐쉬 데이터를 이용한 질의 처리의 가능 여부를 판단하며 인접 클라이언트와 요구 클라이언트 사이의 Connection을 형성함으로서 클라이언트-클라이언트의 질의 처리 및 데이터 전송 기능을 지원한다. 그리고 이러한 기능의 지원을 위한 서버, C-C Connector, 클라이언트 사이에서 수행되어지는 질의 처리의 프로토콜을 정의한다.

제안된 C-C Connector의 Method 및 프로토콜은 서버 사이드의 처리 비용을 현재 서버에 접속중인 클라이언트들에게 분배함으로서 서버 사이드의 병목현상과 질의 처리 시간의 지연이라는 문제점을 개선할 수 있으며 클라이언트 사이드에서의 반응 속도의 향상과 현재 연결된 시스템의 처리 성능을 최대한 활용할 수 있다는 장점을 갖는다.

1. 서 론

현재 멀티미디어 데이터를 비롯한 지도 데이터 등의 공간 데이터에 있어서 그 사이즈가 계속 증가하고 있기 때문에 이러한 데이터에 대한 운용, 처리 및 전송에 있어서의 부하가 점차로 증가하고 있는 추세이다. 기존 서버의 서비스 방식에는 데이터 전송 방식과 질의 전송 방식 및 하이브리드 방식이 있다[2]. 이러한 서버의 서비스 방식을 공간 데이터와 같은 대용량의 데이터를 처리하는 시스템이나 다수의 클라이언트의 요구가 발생하는 시스템에 적용할 경우 데이터 전송 방식은 서버에서의 데이터 전송 비용이 크고 질의 전송 방식은 서버에서의 질의 처리 비용이 크기 때문에 이로 인한 서버 사이드의 병목 현상 및 질의 처리 속도의 저하라는 문제점을 갖는다[4].

이러한 문제점을 개선하기 위해서 본 논문에서는 대용량의 데이터 베이스를 운용하는 클라이언트-서버 환경의 시스템에서 클라이언트에 캐쉬된 데이터를 이용한 클라이언트-클라이언트의 질의 처리 및 전송을 지원하여 서버의 질의 처리 비용과 데이터 전송 부하를 개선할 수 있는 미들웨어인 C-C Connector(Client to Client Connector)의 Method 및 프로토콜을 연구한다.

공간 데이터와 멀티미디어 데이터의 경우 데이터의 크기가 대용량이고 수정 연산이 빈번히 발생하지 않는다는 점을 고려하여 서버에서 질의 처리 중에서 SELECT 연산의 서버 처리비용과 전송 비용을 줄일 수 있는 미들웨어인 C-C Connector를 설계한다. C-C Connector는 클라이언트로부터 전송된 질의를 분석하고 CIT(Client Information Table)를 검색하여 인접한 클라이언트들의 캐쉬 데이터를 이용한 질의 처리의 가능 여부를 판단한다. 처리가 가능한 경우, 인접 클라이언트와 요구 클라이언트 사이의 Connection을 형성함으로서 클라이언트-클라이언트의 질의 처리 및 데이터 전송 기능을 지원하며, 이는 서버의 질의 처리 및 데이터 전송 비용을 가용한 인접 클라이언트들에게 분산시킨다. 또한 이러한 질의 처리를 효과적으로 수행하기 위해서 서버, C-C Connector, 클라이언트 사이에서 수행되어지는 질의 처리의 프로토콜을 정의한다.

이러한 구조와 프로토콜은 클라이언트의 요구를 처리하기 위한 서버 사이드의 처리 및 전송 비용을 현재 서버에 접속중인 클라이언트에 들에게 분배함으로서 서버의 부하를 급격히 감소시키며 대용량 데이터의 처리를 요구하는 다수의 클라이언트 요청에 의한 서버 사이드의 병목현상을 감소시킬 수 있다. 이러한 장점을 통하여 처리 시간의 단축과 클라이언트 사이드에서의 반응 속도를 개선할 수 있으며 현재 연결된 시스템의 처리 성능을 최대한 활용할 수 있다는 장점을 갖는다.

2. 관련 연구

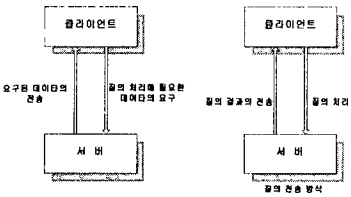
클라이언트-서버 시스템은 업무가 지역적으로 분산된 곳에서 이루어지고 동시에 여러 응용 업무에 공통된 데이터를 사용할 필요성이 있는 환경에서, 복잡한 분산 환경 시스템을 구축하지 않고 개발비용을 최소로 유지하고 성능을 극대화하기 위한 해결책으로 제시되었다. 관련 연구에서는 서버의 데이터 서비스 형태에 따른 클라이언트-서버 DBMS의 클라이언트-서버 질의 처리 방법들을 살펴본다.

클라이언트의 트랜잭션이 서버에게 요구하는 서비스의 형태에는 [그림 2-1]에서와 같이 질의를 서버에게 전송하여 그 결과를 넘겨받는 질의 전송(Query Shipping)방식과 질의를 처리하는데 필요한 데이터를 요청하여 넘겨받는 데이터 전송(Data Shipping)방식이 있다[2].

질의 전송 방식은 표준 클라이언트-서버 DBMS에서 사용하는 방법으로 질의 처리는 서버에서만 이루어지고 서버는 클라이언트에서 발생하는 요구를 기다린다. 질의 전송(Query Shipping)의 장점은 selectivity가 높은 질의들에 대해 데이터 전송 비용을 줄일 수 있는 효과가 있다[4]. 이 경우 시스템 전체의 성능은 서버 프로세스의 성능에 의존하게 되는데,

1) 본 연구는 정보통신부의 대학 S/W 연구센터 지원사업의 연구 결과임

많은 수의 클라이언트들이 동시에 서버에게 트랜잭션 처리를 요청하게 되면, 서버 프로세스가 이들 요청을 실행하기 위한 디스크 입출력, 트랜잭션 관리, 주기억 장치 내의 연산 등으로 인하여 병목현상을 유발하게 되며, 이러한 병목 현상은 시스템 전체 성능에 크게 영향을 주게 된다[4].



[그림 1]서버의 데이타 서비스 형태

데이터 전송 (Data Shipping) 방식은 고성능 클라이언트의 CPU, 메모리, 디스크 자원을 이용하여 질의 처리를 담당함으로써 서버는 클라이언트가 질의 처리를 할 수 있도록 요청한 데이터를 전달해 주는 역할을 수행한 다. 일반적인 속성 정보를 검색할 때는 질의에 필요한 데이터량이 크지 않아 데이터 전송 비용이 작고, 클라이언트가 가진 질의 처리 능력으로 인해 서버의 처리 부하를 줄일 수 있어 효과적이다. 그리고, 클라이언트의 자체 질의 처리 능력을 이용하므로 시스템 확장이 용이한 장점이 있다. 하지만, 참조 데이터 량이 크거나 locality가 낮아 캐쉬 미스가 빈번하게 발생하는 질의의 경우는 데이터 전송에 요구되는 비용이 커지는 문제점이 있다. 현재는 클라이언트-서버 환경에서의 대용량 데이터에 대한 처리를 위해서 이러한 질의 전송과 데이터 전송을 혼합한 하이브리드 방식의 처리 방식이 많이 연구가 되고 있지만 기존의 방식은 모두 질의 처리나 데이터 전송을 위한 비용을 서버에서 해결해야 한다는 한계점을 가진다[3].

3. C-C Connector의 구성 Method와 Protocol

본 장에서는 다수의 클라이언트의 요구가 발생할 때 또는 대용량의 데이터 처리를 요구하는 클라이언트들에 의해서 유발된 과도한 서버의 처리 및 전송 비용 때문에 발생하는 서버 처리 속도의 지연과 병목 현상을 방지하고 트랜잭션 처리 성능의 개선을 위한 방식으로 C-C Connector라는 미들웨어를 제안한다. 본문에서 클라이언트의 요구 처리에 관한 간략한 기술과 C-C Connector의 구성 Method에 대한 각각의 역할과 정책에 대해서 살펴보고 C-C Connector를 이용한 요구 처리의 프로토콜을 제안한다.

3.1 적용 분야와 요구 처리의 전체 개요

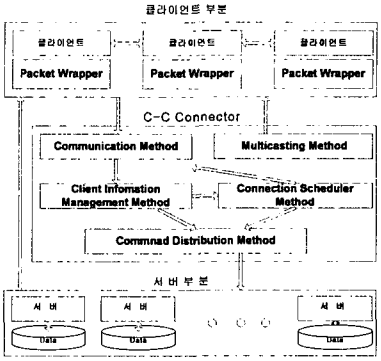
C-C Connector는 클라이언트-서버 환경을 위한 미들웨어이며 공간 데이터와 멀티미디어 데이터와 같은 수정이 빈번히 발생하지 않는 대용량의 데이터를 접근하는 그룹 단위의 작업이 이루어지는 환경에 적합하다. 그룹 단위의 작업이 이루어지는 C/S 환경인 경우 그룹 내의 클라이언트들이 접근하는 데이터들이 유사하기 때문에 인접 클라이언트의 캐쉬 데이터를 사용하여 질의 처리가 가능하다. 또한 본 논문에서 적용된 클라이언트의 경우 데이터 전송 방식을 지원할 수 있는 질의 처리기를 가지고 있다고 가정한다.

각각의 클라이언트에는 질의 처리기가 존재하므로 클라이언트에서 발생하는 질의는 해당 클라이언트에서 파싱을 한다. 클라이언트의 요구가 발생할 때 클라이언트의 요구가 해당 클라이언트 사이드의 캐쉬된 데이터로서 처리가 가능한 경우 클라이언트 사이드의 질의 처리기를 통해서 요구를 처리한다.

해당 클라이언트에 필요한 데이터가 존재하지 않는 경우 C-C Connector를 통하여 다른 클라이언트 사이드의 캐쉬된 데이터가 가용한지의 여부를 판단하여 가능한 경우 클라이언트 사이의 Connection을 형성함으로서 인접 클라이언트의 캐쉬 데이터를 통하여 처리할 수 있다. 이 경우 인접 클라이언트 사이드에 존재하는 질의 처리기를 통해서 처리를 한 후에 결과만을 전송함으로서 전송 비용을 최소화한다. 인접 클라이언트 사이드에 처리를 위한 데이터가 존재하지 않는 경우는 일반적인 C/S 환경에서의 처리와 동일한 방식으로 서버에 처리를 요청하게 되며 서버의 서비스 방식에 따라서 처리에 필요한 데이터 또는 결과를 전송 받게 된다.

3.2 C-C Connector Method 의 구성 Method

본 장에서는 C-C Connector를 구성하는 각각의 Method들과 그 역할 및 정책에 대해서 고려한다. 제안하는 방식을 적용하기 위해서는 Component 형식의 C-C Connector의 Method들의 지원이 요구되며 [그림 2]와 같은 구조로 구성된다.



[그림 2] C-C Connector의 구성 Method들과 처리 흐름도

Communication Method는 클라이언트의 요청을 받아드리기 위한 일반적인 모듈이며 Packet Wrapper로부터 전송된 패킷에서 추가된 헤더 부분을 추출하는 역할을 한다.

Command Distribution Method는 시스템의 확장성을 고려한 부분으로 서버가 여러 개의 데이터베이스를 가지는 경우를 위한 확장성을 고려한 Method 이다. 각각의 데이터 베이스에 존재하는 데이터들의 정보를 가지고 있으며 요청되는 데이터에 따라서 적절한 데이터 베이스로의 접근을 허용하게 된다. 나머지 Method에 대한 역할 및 정책은 아래와 같다

3.2.1 Packet Wrapper Method

시스템의 확장성을 높이기 위해서 기존의 시스템에서 사용하는 packet의 형식을 그대로 사용할 수 있도록 하기 위한 Method 이며 C-C Connector에서 요구되는 정보를 기존의 패킷에 추가한다.

패킷에 추가된 헤더 부분은 CIT(Client Information Table)을 구성하는 정보들을 위한 것으로 클라이언트에서 질의의 파싱후 얻어지는 정보의 일부를 추가한다. 헤더에 포함되어지는 정보는 클라이언트 정보, 테이블 정보, 질의 조건 정보(Information for Predicate)들로 구성된다.

3.2.2 Client Information Management Method

클라이언트 사이의 Connection의 형성을 위한 클라이언트의 정보와 각각의 클라이언트에 캐쉬된 데이터 관련 정보를 관리함으로서 Connection Scheduler Method에게 효과적인 Connection을 판단할 수 있는 정보를 제공하며 사용 가능 데이터를 가진 클라이언트가 여럿인 경우 Connection에 사용할 클라이언트를 결정하여 클라이언트 리스트를 Connection Scheduler Method에게 전달한다.

CIT 테이블에는 클라이언트가 서버에 접속 시에 클라이언트가 등록되어지며 서버로부터의 SELECT 연산이 발생할 때에 클라이언트에 캐쉬되어지는 데이터의 변화를 기록한다. 이러한 클라이언트의 캐쉬 데이터에 대한 정보는 클라이언트의 접속 해제 시에 제거하여 다른 클라이언트가 접속을 해지한 클라이언트에 접속을 시도하는 오류를 방지한다.

[그림3]에서 클라이언트의 정보는 인접 클라이언트와의 접속을 위해서 요구되는 IP와 Port, 해당 클라이언트에 연결된 인접 클라이언트의 수와 같은 정보들이며 테이블 정보는 이름, 크기, 캐쉬된 영역 등이며 질의 조건 정보는 질의 처리 시에 사용한 조건절의 정보이며 필드명, 연산자, 피 연산자와 같은 정보들로 구성된다.

클라이언트 정보	테이블 정보	질의 조건 정보	Validation Check

[그림 3] CIT 의 구성 정보

클라이언트로부터의 요구가 발생하면 데이터베이스의 수정 연산인 경우 서버에서 처리하도록 하며 SELECT 연산인 경우 [그림 3]의 CIT를 검색하여 해당 연산이 클라이언트 캐쉬 데이터를 사용하여 연산이 가능한지의 여부를 분석한다. 이 분석에서는 캐쉬 정보에서 요구된 테이블의 존재 여부를 확인하고 캐쉬된 테이블의 구성 데이터가 질의를 위해서 요구되는 데이터와 같거나 요구된 데이터를 포함하는 데이터인 경우에 사용 가능 판단을 하게 된다.

이것은 질의 조건 정보들을 비교함으로서 포함관계를 파악할 수 있으며 마지막으로 Validation 정보를 확인하여 서버 데이터와의 일치성 여부를 확인 후 질의에 사용여부를 판단하게 된다.

여러 개의 클라이언트가 검색되어지는 경우 Client Number Connected

의 수를 참조하여 0이거나 가장 낮은 값을 가진 클라이언트의 정보를 반환한다.

3.2.3 Connection Scheduler Method

Client Information Management Method(CIMM)로부터의 정보를 이용하여 사용 가능한 데이터를 가진 클라이언트와 요청 클라이언트 사이의 요구 처리 및 Connection을 관리하는 Method이며 CIMM에서 검색된 다음과 같은 경우들에 대한 제어/처리를 수행한다.

case1) 요청된 table들이 하나의 클라이언트에 존재하는 경우
case2) 요청된 table이 두 개의 클라이언트에 나뉘어져 존재하는 경우
case3) 요청된 table 중 하나가 서버에 존재하는 경우
case4) 요청된 table 모두가 서버에 존재하는 경우

위의 경우에 대한 처리는 3.4절의 프로토콜에 따른다.

3.2.4 Multicasting Method

클라이언트 상의 캐쉬 데이터에 대한 가용성을 높이기 위한 Method로서 데이터 베이스에 변경이 발생했을 경우 변경된 데이터를 캐쉬하고 있는 클라이언트를 CIT로부터 검색하여 실시간으로 캐쉬 데이터를 update하기 위한 정보를 전송하기 위한 Method 이다.

이러한 Multicasting은 네트웍 트래픽을 발생시킬 수 있으므로 import / Bulk Insert 연산 시에는 제공하지 않으며 Notify Message 만을 해당 클라이언트로 전송함으로서 클라이언트에서 원하는 시점에 변경된 데이터에 대한 요구를 허용하도록 한다.

3.3 구성 모듈간의 처리 흐름

클라이언트에서 서버로의 Request가 발생하면 클라이언트 부분의 Packet Wrapper가 해당 packet에 헤더를 추가하여 C-C Connector로 전송한다. Communication Method가 클라이언트로부터 전송된 packet을 받아서 PW_Header를 추출하고 Client Information Manager Method가 연산 분석 및 현재 접속중인 클라이언트 사이드의 캐쉬 데이터를 사용해서 처리가 가능한 질의인지의 여부를 판단한다.

클라이언트 사이드의 캐쉬를 사용 가능한 질의이면 Connection Scheduler Method가 각각의 경우에 대해서 3.4절의 프로토콜대로 처리를 하도록 한다. 데이터베이스에 대한 수정 연산이거나 가용한 클라이언트의 캐쉬 데이터가 없는 경우는 일반적인 C/S 환경에서의 처리와 같이 서버에서 처리하도록 하며 수행 후에는 Multicasting Method를 통하여 캐쉬 데이터에 대한 변경을 함으로서 캐쉬의 일관성을 실시간으로 유지하여 캐쉬 데이터의 가용성을 높인다.

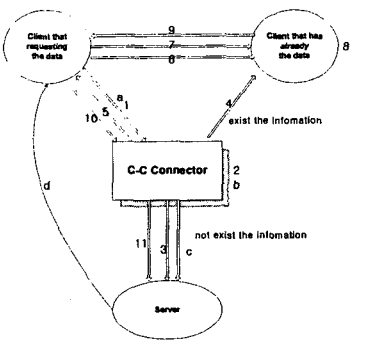
3.4. 프로토콜

본 장에서는 앞에서 기술한 C-C Connector를 이용한 질의 처리 방식을 고려한다. 이는 C-C Connector를 매개로 서버, C-C Connector, 클라이언트 사이의 효과적인 요구 처리를 위한 프로토콜이며 하나의 클라이언트에 질의 처리를 위한 데이터가 모두 존재하는 경우와 여러 클라이언트에 나뉘어져 있는 경우로 구분하여 고려하도록 한다.

3.4.1 요구 데이터가 하나의 클라이언트 사이드에 존재하는 경우

[그림 5]는 대상 클라이언트가 하나인 경우의 처리 흐름도이며 다음과 같은 처리를 수행한다. CLIENT1은 데이터에 대한 처리를 요청하는 클라이언트이고 CLIENT2 는 요청된 데이터를 캐쉬하고 있는 클라이언트라고 가정한다.

1. CLIENT1의 질의 발생. C-C Connector에 패킷 전송.(1, a)
2. C-C Connector는 연산의 분석과 클라이언트 정보의 검색 수행.(2, b)
- 캐쉬사용이 가능한 연산이고 CLIENT 2가 존재하는 경우
3. 서버 사이드에 transaction 관리를 위한 정보의 전송(3)
4. CLIENT2 에 Connection Preparation 메시지와 데이터 관련 정보를 전송(4)
5. CLIENT 1에게 CLIENT 2에 대한 정보를 전송.(5)
6. CLIENT2에 대한 Connection을 형성.(6)
7. 클라이언트 1에서 파싱된 질의 정보를 전송(7)
8. CLIENT2에서 질의 처리.(8)
9. 처리 결과의 전송(9), CLIENT2는 Connection 해제.
10. CLIENT1은 작업 완료를 C-C Connector에 통보.(9)
11. 서버에 트랜잭션 완료 메시지 전송.
- 캐쉬사용이 불가능한 연산이거나 클라이언트(CLIENT 2)가 존재하지 않는 경우
3. C-C Connector는 클라이언트의 요청을 서버로 전달.(c)
4. 서버에서 처리하여 결과나 데이터를 클라이언트1으로 전송.(d)



[그림 4] 요구된 데이타가 하나의 클라이언트에 존재하는 경우의 처리 흐름

3.4.2 요구된 데이터가 하나 이상의 클라이언트에 존재하는 경우

질의 처리의 요구를 위해서는 역시 C-C Connector에 먼저 패킷을 전송하게 되며 질의가 클라이언트의 캐쉬 데이터를 사용 가능한 연산인 경우 C-C Connector에서 요구되는 데이터가 두 개의 클라이언트(CLEINT2, CLIENT3)에 나뉘어져 존재하는 경우이다. 이러한 경우 CIT의 데이터 크기와 MBR의 정보를 이용하여 두 클라이언트가 가진 데이터의 사이즈를 비교한다. CLEINT2가 큰 사이즈의 데이터를 캐쉬하고 있다면 C-C Connector는 먼저 CLIENT2과 CLIENT3 사이에 Connection을 형성하게 되며 CLEINT3에서 CLEINT2로 요구되는 영역만큼의 데이터를 전송한다. 다음으로 CLEINT2와 CLEINT1사이에 Connection을 형성함으로서 요구되는 데이터를 클라이언트2가 가지고 있는 CLEINT2에서 질의 처리를 하고서 CLEINT1으로 결과를 전송한다. 나머지의 부분은 [그림 5]와 동일한 프로토콜을 따른다.

4. 결 론

본 논문에서는 많은 클라이언트의 요청이나 빈번한 대용량 데이터의 처리 연산 때문에 발생하는 서버 사이드의 질의 처리 및 데이터 전송 비용을 줄이기 위한 방식으로서 C-C Connector라는 미들웨어의 Method들과 이를 통한 서버, C-C Connector, 클라이언트 사이의 프로토콜을 제안하였다.

C-C Connector는 서버 사이드의 비용을 서버에 접속하고 있는 클라이언트들에게 분산시킴으로서 서버 사이드에서의 처리 비용과 병목 현상을 완화시킬 뿐만 아니라 처리 속도의 향상 및 클라이언트 사이드의 가용한 리소스를 효과적으로 사용할 수 있다는 장점을 갖는다.

향후 연구로는 수정 연산에서의 비용 감축을 위한 C-C Connector의 처리 방식에 대한 고려와 수정이 빈번히 발생하는 경우의 정책에 대한 연구가 요구된다.

5. 참고 논문

[1] Ralph Wittmann, martina Zitterbart, "Multicast Communication Protocol and Applications", Morgan Kaufmann, 7-36, 53-119, 1999

[2] Michael J. Franklin, Mechael J. Carey, Miron Livny "Transactional Client-Server Cache Consistency: Alternatives and Performance", ACM, 1997

[3] Yongdong Wang, Lawrence A.Rowe, "Cach Consistency and Concurrency Control in a Client-Server DBMS Architecture", ACM, pp367-379

[4] 강 규원, "클라이언트-서버 환경을 지원하는 GIS에서의 역할 분담 스케줄러", 인하대학교 전자계산공학과 공학석사학위논문, 1998

[5] Jayavel Shanmugasundaram, Arvind Mithrakashyap, Rajendran Sivasankaran, Krithi Ramamritham, "Efficient Concurrency Control for Broadcast Environments", ACM, 1999

[6] Guting, R.H., and M. Schneider, "Realms: A Foundation for Spatial Data Types in Database Systems", In Proc. 3rd Intl. Symposium on Large Databases, Singapore, 1993, 14-35.