Peer-to-Peer 환경에서 중복된 데이터의 갱신 전파 기법

준회원 최 민 영*, 정회원 조 행 래**

Update Propagation of Replicated Data in a Peer-to-Peer Environment

Minyoung Choi* *Associate Member*, Haengrae Cho** *Regular Member*

요 약

P2P(Peer-to-Peer) 시스템은 대용량의 데이터를 공유하는데 유용하며, 네트워크 구조에 따라 중앙 집중형, 구조적 분산형, 그리고 비구조적 분산형으로 분류된다. 이 중 Gnutella와 같은 비구조적 분산형 P2P 시스템은 확장성과 신뢰성 측면에서 장점을 갖지만, 참여하는 노드의 수가 증가함에 따라 데이터를 액세스하는 비용도 증가한다는 문제를 가진다. 데이터 중복을 이용하여 이러한 문제를 해결할 경우 중복된 데이터들의 일관성을 유지하기 위한 기법이 필요하다. 본 논문에서는 특정 노드가 데이터를 갱신할 때 중복된 사본을 저장하고 있는 다른 노드에 전파하기 위한 새로운 갱신 전파 알고리즘을 제안한다. 제안한 알고리즘은 타임스탬프와 push/pull 개념을 조합하여 메시지의 전송 오버헤드를 줄일 수 있다는 장점을 갖는다.

Key Words : Peer-to-Peer, Data Replication, Update Propagation, Consistency Management, Performance Evaluation

ABSTRACT

Peer-to-peer (P2P) systems have become a popular medium through which to share huge amounts of data. On the basis of network topology, P2P systems are divided into three types: centralized, structured distribution, unstructured distribution. Unstructured P2P systems such as Gnutella are novel in the sense that they are extensible and reliable. However, as the number of nodes increases, unstructured P2P systems would suffer from the high complexity of search operations that have to scan the network to find the required data items. Efficient replication of data items can reduce the complexity, but it introduces another problem of maintaining consistency among replicated data items when each data item could be updated. In this paper, we propose a new update propagation algorithm that propagates an updated data item to all of its replica. The proposed algorithm can reduce the message transfer overhead by adopting the notion of timestamp and hybrid push/pull messaging.

Ⅰ. 서 론

P2P(Peer-to-Peer) 시스템은 각 노드들이 가지고 있는 자원을 상호간의 직접적인 연결을 통하여 교환할 수 있는 네트워크 환경으로 자원이나 대용량의 데이터를 공유하는데 유용하다. P2P 시스템의 네트워크는 형성 구조에 따라 중앙 집중형 네트워크와 분산형 네트워크로 분류된다[1]. Napster[2]와 같은 중앙 집중형 네트워크는 중앙 서버가 노드들의 IP 주소와 공유 파일의 인덱스 및 메타 데이터를

※ 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성·지원사업의 연구결과로 수행되었습니다.
* LG CNS 공공사업부 (chalscse@ynu.ac.kr), ** 영남대학교 전자정보공학부 (hrcho@yu.ac.kr)
논문번호 : KICS2006-01-034 접수일자 : 2006년 1월 17일, 최종논문접수일자 : 2006년 4월 11일

관리하는 방식이다. 중앙 집중형 네트워크는 중앙 서버가 모든 노드들의 정보를 관리하기 때문에 검색 시간이 빠르고 자원 관리가 유용한 장점이 있지만, 서버에 병목현상이 발생하여 성능이 저하될 수 있으며 서버가 동작하지 않을 경우 전체 시스템이 중단된다는 단점이 있다.

분산형 네트워크는 노드들이 자체 조직되어 서버와 클라이언트의 역할을 병행하는 방식이며, 특정 토폴로지를 구성하여 규칙적으로 파일을 배치하는 구조적 방식과 토폴로지와 파일의 배치에 제한을 두지 않는 비구조적 방식으로 분류된다. 구조적 방식의 예로 DHT(Distributed Hash Table)기반의 Chord[3]와 CAN[4], 그리고 P-Grid[5] 알고리즘 등을 들 수 있다. 각 노드는 해쉬 함수를 이용하여 키를 분배 받으며, 키를 이용하여 파일을 검색한다. 구조적 방식은 검색 비용이 적은 반면, 노드들이 서로 의존적이므로 접속과 탈퇴에 따른 추가적인 비용이 발생한다. Gnutella[2]와 같은 비구조적인 방식은 노드들의 연결이 임의적이고 파일의 배치가 규칙적이지 않다. 노드들 사이의 의존성이 낮아서 접속과 탈퇴에 큰 영향을 받지 않으므로 확장성이 좋으며 노드 실패나 네트워크 실패에 강해 신뢰성이 좋은 장점을 가진다. 그러나 flooding 방식으로 자원을 검색하기 때문에 메시지가 많이 발생하며 네트워크에 참여하는 노드의 수가 증가함에 따라 원하는 자원을 액세스하는 비용도 증가하는 문제를 가진다.

표준화를 통해 P2P 네트워크의 규모가 충분히 커질 경우에는 규칙적인 토폴로지의 적용이 어려워지므로 확장성이 좋은 비구조적 분산형 P2P 네트워크가 주를 이루게 될 것이다[6]. 다양한 P2P 네트워크 환경에서 자원 검색의 효율을 높이고 응답시간을 단축시키기 위한 많은 연구들이 진행되었다[7,8]. 그 중 한 가지 방법으로 데이터 중복을 이용한 알고리즘이 제안되었으며 많은 연구를 통해 그 효율성이 입증되었다[1,9-13]. 데이터 중복을 이용함으로써 시스템의 내구성과 유용성이 증대되고 응답시간이 단축되며, 성능저하의 원인인 병목현상, 노드실패, 검색실패 등의 문제점을 해결할 수 있다.

본 논문에서는 Gnutella와 같은 비구조적 분산형 P2P 네트워크에서 데이터 중복을 사용할 때 발생하는 사본들의 일관성 문제를 해결하기 위하여 타임스탬프 기반의 하이브리드 push/pull 알고리즘을 제안한다. 제안하는 알고리즘은 노드들의 타임스탬프와 송신노드의 리스트를 push 메시지에 첨부하고, 전체 pull 알고리즘과 선택적 pull 알고리즘을 적극적으로 활용함으로써 메시지의 중복 전송을 줄이고 중복된 데이터의 일관성을 효율적으로 유지시킬 수 있는 장점을 가진다.

본 논문의 구성은 다음과 같다. 먼저 2절에서 P2P 네트워크 환경에서 메시지 전송 방법에 따른 문제점을 살펴보고 기존에 제안된 메시지 전파 기법들을 살펴본다. 3절에서는 본 논문에서 제안한 메시지 전파 알고리즘을 자세히 설명한다. 4절에서는 제안하는 알고리즘의 성능을 평가하기 위하여 개발한 실험 모형에 대해 설명하고 5절에서 실험 결과를 분석한다. 마지막으로 6절에서 결론을 맺는다.

Ⅱ. 관련 연구

본 절에서는 P2P 네트워크 환경에서 메시지의 전파 방법을 소개하고 각 방법에 따른 문제점을 살펴본다. 그리고 기존에 제안된 중복된 데이터의 갱신 전파 기법들을 살펴본다.

2.1 갱신 전파 방법

일반적인 P2P 환경에서는 특정 노드에서 자신이 저장한 지역 데이터를 우선적으로 갱신한 후 다른 노드들에게 갱신 내용을 전파하는 지연 갱신 전파 방식을 사용하며, 전파방법에 따라 push와 pull의 두 가지 기법이 존재한다[14,15]. Push는 갱신을 발생시킨 노드가 이웃 노드에게 갱신 정보를 전파하는 방법이고, pull은 갱신하지 않은 노드가 갱신을 발생시킨 노드에게 갱신 정보를 요청하는 방법이다. 하이브리드 push/pull은 push와 pull을 모두 이용하는 방법이다.

그림 1의 (a)는 push 방식의 갱신 전파 방법에 대한 설명이다. 노드 A가 데이터를 갱신하고 해당 갱신 정보를 이웃 노드 B에게 전파한다. 노드 B는 수신한 갱신 정보를 반영하고 해당 갱신 정보를 이웃 노드에게 전파한다. 이러한 과정을 반복하여 모든 노드에게 갱신 정보를 전파한다. 그림 1의 (b)는 pull 방식의 갱신 전파 방법을 나타낸다. 노드 B가 노드 A에게 갱신 정보를 요청하면 노드 A는 자신의 갱신 정보를 노드 B에게 전파한다.

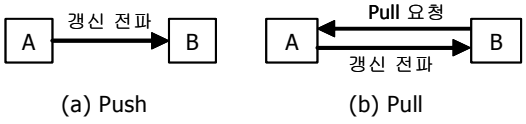

그림 1. 메시지 전파 방법

2.2 갱신 중복 전송

Push 기반의 갱신 전파 기법을 사용하면 동일한 갱신 정보를 중복해서 전송하는 경우가 발생한다. 동일한 갱신 정보를 중복 전송함으로써 네트워크 자원의 낭비가 초래된다. 통신비용이 비싼 무선 환경의 경우 중복 전송으로 인한 자원의 낭비는 심각한 문제가 된다.

그림 2는 갱신 중복 전송을 설명한다. 노드 A가 데이터를 갱신하고 해당 갱신 정보를 이웃 노드 B에게 전파한다. 노드 B는 수신한 갱신 정보를 자신의 지역 데이터에 반영하고 이웃 노드 C와 D에게 수신한 갱신 정보를 다시 전파한다. 노드 B로부터 갱신 정보를 전파 받은 노드 C와 D는 수신한 갱신 정보를 지역 데이터에 반영한 후 갱신 정보를 이웃 노드에게 전파한다. 노드 C와 D는 서로 이웃 노드이므로 노드 B에게서 수신한 갱신 정보를 서로에게 전파한다. 이때 노드 C와 노드 D는 이미 해당 갱신 정보를 지역 데이터에 반영한 상태이므로 이 과정에서 발생되는 갱신 전파는 불필요한 중복 전송이 된다. 네트워크에 참여한 노드의 수와 노드에 연결된 이웃 노드의 수가 많을수록 중복 전송되는 경우도 증가한다.

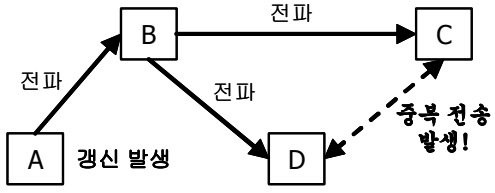

그림 2. 갱신 중복 전송의 예

2.3 기존 연구

[16]에서는 각 노드의 갱신 정보를 로그로 구성하여 주기적으로 이웃 노드들에게 전파하는 push 기반 전염(epidemic) 알고리즘을 제안하였다. Push 메시지를 수신한 노드는 갱신 정보를 자신의 지역 데이터에 반영하고 다시 자신의 이웃 노드에게 전파하는 과정을 반복한다. 이러한 과정을 반복함으로써 전체 네트워크의 모든 노드들에게 갱신 정보를 전파할 수 있다. 그러나 그림 2와 같이 동일한 갱신 정보를 중복해서 전송하는 경우가 발생하여 자원이 낭비되는 문제점이 있다.

Deno 시스템[17]은 서버를 이용한 계층적 네트워크 구조로 구성되어 있으며 서버들 사이의 이벤트 정보를 pull을 이용하여 동기화하였다. 이 경우 pull을 요청할 대상이 최신 데이터를 가짐을 보장할 수 없기 때문에 최신 데이터를 찾는데 걸리는 시간이 길어지며 불필요한 메시지가 발생되는 문제점이 있다. 최신 데이터를 유지하기 위해서 pull의 주기를 짧게 하면 pull 요청 메시지가 많아지며, 반대로 메시지 발생량을 줄이기 위해 pull의 주기를 길게 하면 최신 데이터를 인지하는데 걸리는 시간이 길어지게 된다. 즉, pull 기반 알고리즘은 요청 주기에 따라 발생하는 메시지의 양과 갱신을 인지하는 시간사이의 상관관계가 있기 때문에 비효율적인 경우가 발생한다.

[15]에서는 소유자[1] 노드를 가정한 무효화 메시지 기반의 하이브리드 push/pull 알고리즘을 제안하였다. 모든 데이터마다 자신의 소유자 노드가 존재하며, 소유자 노드만이 데이터를 갱신하고 갱신된 내용을 전파할 수 있다. 소유자 노드로부터 일정 홉(TTL) 내의 노드는 push 방식 통해 무효화 메시지를 전달받고, 나머지 노드는 pull을 적응적으로 사용하여 갱신 정보를 전달받는다. 이 경우 모든 갱신이 소유자 노드를 통해서 전파되므로 소유자 노드가 오프라인이면 갱신할 수 없는 문제가 발생하며, push 알고리즘을 사용함으로써 발생되는 메시지의 중복 전송을 해결하지 못한다. 그리고 무효화 메시지 기반의 알고리즘이므로 무효화된 데이터를 액세스하기 위해서는 다시 소유자 노드와의 통신이 필요하다.

[14]에서는 push 메시지에 수신노드의 리스트를 포함하여 전송하고, 일정시간 전송 받은 push가 없거나 재접속할 경우 pull을 요청하는 하이브리드 push/pull 알고리즘을 제안하였다. Push 메시지의 수신노드 리스트에 포함된 노드에게는 동일한 메시지를 전송하지 않음으로써 중복 전송을 줄일 수 있지만, 네트워크 장애나 노드의 빈번한 접속과 탈퇴로 인해 갱신을 전파 받지 못한 노드가 수신노드의 리스트에 포함되는 문제가 발생한다. 그림 3은 메시지 분실로 인해 미반영 갱신이 발생하는 과정을 설명한다. 노드 A가 데이터를 갱신하고 전파할 push 메시지에 수신 노드의 리스트 [A,B]를 첨부하여 노드 B에게 전파한다. 노드 B는 수신한 갱신 정보를 지역 데이터에 반영하고 push 메시지에 갱신을 전파할 이웃 노드들을 포함한 수신 노드 리스트 [A,B,C,D]를 첨부하여 노드 C와 D에게 전파한다. 이때 메시지 분실로 인해 노드 C가 해당 메시지를

1) "소유자 노드"는 여러 노드에 중복된 데이터를 갱신할 수 있는 권한을 가지는 노드를 의미하며, 갱신된 데이터의 전파는 소유자 노드를 통해서만 가능하다.

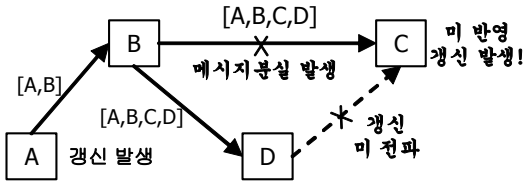

그림 3. 미반영 갱신이 발생하는 예

받지 못했음을 가정하자. 노드 D는 push 메시지의 수신 노드 리스트에 노드 C가 포함되어 있으므로 노드 C가 자신의 이웃 노드이지만 갱신이 반영된 것으로 간주하고 push 메시지를 전파하지 않는다. 반면, 노드 C는 push 메시지를 받지 못했음에도 불구하고 수신 노드 리스트에 포함되어 있기 때문에 이웃 노드 D로부터 갱신 정보를 전파 받지 못해 미 반영된 갱신이 존재하게 된다.

본 논문에서는 [14]의 아이디어를 확장하여 push 메시지에 송신노드의 리스트를 전송하고, pull을 전체 pull과 선택적 pull로 나누어 적극적으로 이용하는 하이브리드 push/pull 알고리즘을 제안한다.

Ⅲ. 타임스탬프를 이용한 하이브리드 push/pull 기반의 갱신 전파 기법

본 절에서는 타임스탬프를 이용한 하이브리드 push/pull 기반의 갱신 전파 모델을 제안하고, 이를 바탕으로 push 메시지에 송신노드의 리스트를 첨부하는 방식, 전체 pull 방식, 그리고 선택적 pull 방식을 제안한다. 표 1은 알고리즘에 사용되는 매개변수에 대한 설명을 나타낸다. 각 노드마다 타임스탬프가 할당되며, 각 노드는 네트워크 전체 노드의 타임스탬프 테이블을 유지한다. H_i는 노드 i가 유지하고 있는 전체 노드의 갱신 이력 테이블이며, $H_i[k, TS(k)]$의 값은 노드 k가 TS(k)번째 갱신한 데이터 식별자를 나타낸다. 노드 i의 갱신 이력 테이블 H_i에서 모든 노드의 최종 타임스탬프(TS) 값을 모아 둔 것이 타임스탬프 테이블(TM_i)이며, 노드 i가 전송받은 노드 k의 갱신 정보에 대한 최종 타임스탬프 값은 $TM_i[k]$이다.

제안한 갱신 전파 기법은 초기화 과정과 push/pull 기반의 갱신 전파 알고리즘, 그리고 고립 발견 알고리즘으로 구성된다. 초기화 과정은 새로운 노드가 P2P 네트워크에 처음 접속할 때 수행되며, 표 1의 매개변수들을 초기화한다. 구체적으로 노드 i가 네트워크에 처음 접속할 때 Gnutella Ping/Pong 프로토콜[6,18]을 이용하여 이웃 노드의 집합 N(i)를

표 1. 알고리즘의 매개변수와 설명

매개변수	설 명
S	네트워크에 존재하는 모든 노드들의 집합
TTR	Alive 메시지의 전송 주기
N(i)	노드 i의 이웃 노드의 집합
P(i)	노드 i에서 발생된 갱신이 반영된 노드들의 집합
L(i)	노드 i의 N(i) 중 활동 상태인 노드의 수
TS(i)	노드 i의 타임스탬프
TM_i	노드 i가 유지하고 있는 전체 노드의 최종 TS 테이블
$TM_i[k]$	TM_i에서 노드k의 타임스탬프
d_{id}	데이터 식별자
ps(i)	노드 i가 마지막으로 전파한 push의 발생 노드
$V_i(d_{id})$	노드 i가 유지하고 있는 d_{id}의 value
$H_i[k, TS(k)]$	노드 i의 갱신 이력 테이블로써, 노드 k가 TS(k)시점에서 갱신한 d_{id}를 저장

파악하고, N(i)에 포함된 노드로부터 TTR 주기를 수집한다. 그리고 TS(i)와 TM_i 테이블의 모든 항목들은 0으로 초기화된다. 또한 H_i 테이블은 null 값으로 초기화된다. 이후 중복된 데이터의 갱신 전파 과정은 3.1절의 push 알고리즘과 3.2절의 pull 알고리즘을 이용하여 수행되고, 노드 i와 N(i)들 간에 연결성 검사 및 재연결 과정은 3.3절의 고립 발견 알고리즘에서 수행된다. 초기화 단계에서 중복 데이터의 수집은 3.2절의 전체 pull 알고리즘을 통하여 이루어질 수 있다. 본 절의 나머지 부분에서 각 알고리즘들을 구체적으로 살펴보도록 한다.

3.1 Push 알고리즘

각 노드는 지역 데이터를 갱신한 후 이웃 노드에게 갱신 정보를 전파해야한다. 본 논문에서는 push 방식을 이용하여 갱신을 전파한다. 그림 4는 본 논문에서 제안한 push 알고리즘의 예를 설명한다. 노드 A가 특정 데이터를 갱신하면 자신의 타임스탬프를 단조 증가한 후, 타임스탬프 테이블에 반영한다. 그리고 갱신을 전파하기 위해 push 메시지를 생성

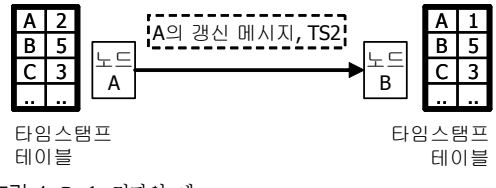

그림 4. Push 전파의 예

• Push 발생 : 노드 i가 d_{id}를 new_value로 갱신하고 push하는 과정

```
UPDATE(d_id, new_value)
  TS(i)++;
  H_i[i, TS(i)]=d_id, TM_i[i]=TS(i), P(i)={i};
  V_i(d_id)=new_value;
  To all k∈N(i), Send(i, i, d_id, V_i(d_id), TS(i), P(i));
```

• Push 수신 : 노드 k가 m으로부터 i의 갱신에 대한 push를 수신

```
RECEIVE_PUSH(m, i, d_id, V_i(d_id), TS(i), P(i))
  IF (TS(i)==TM_k[i]+1) THEN
    TTR=TTR_max;
    H_k[i, TS(i)]=d_id, V_k(d_id)=V_i(d_id);
    ps(k)=i, TM_k[i]=TS(i);
    P(i)=P(i)∪{k};
    To all m∈N(k)-P(i),
        Send(k, i, d_id, V_i(d_id), TS(i), P(i));
  ELSE IF (TS(i)==TM_k[i]) THEN return;
  ELSE IF (TS(i)>TM_k[i]+1) THEN
    Call Request_Selective_Pull(k, i, TM_k[i]) to m;
```

그림 5. Push 알고리즘

하여 노드 B에게 전송한다. 노드 B는 전송받은 push 메시지에서 노드 A의 타임스탬프와 자신의 타임스탬프 테이블에 저장된 노드 A의 타임스탬프를 비교한다. 그림 4에서는 노드 B의 타임스탬프 테이블의 노드 A의 타임스탬프가 1로써 push 메시지의 타임스탬프 2가 유효하므로 push를 반영한다.

그림 5는 노드 i가 d_{id}를 갱신하고 이웃 노드에게 갱신 정보를 전파하는 push 알고리즘을 나타낸다. 갱신 노드 i는 자신의 타임스탬프를 단조증가 시키고 N(i)에게 갱신된 내용을 push한다. 노드 m으로부터 노드 i의 갱신 정보를 push 받은 노드 k는 자신의 타임스탬프 테이블 $TM_k[i]$와 메시지의 타임스탬프 TS(i)를 비교하여 갱신 반영, 전파종료 또는 선택적 pull을 결정한다. TS(i)가 $TM_k[i]$보다 1이 크면 TTR을 TTR_{max}로 설정한 후 갱신을 반영하고 P(i)에 자신을 포함하여 전파한다. TS(i)가 $TM_k[i]$와 같으면 중복 전송된 메시지이므로 전파를 종료하며, TS(i)가 $TM_k[i]$보다 2이상 클 경우에는 수신하지 못한 메시지가 존재하므로 push한 노드에게 선택적 pull을 요청한다. [예 1]은 그림 6의 노드 2가 d_{id}를 갱신한 후 push하는 과정을 설명한다.

[예 1] 노드 2가 d_{id}를 갱신하면 TS(2)를 1증가 시킨다. $H_2[2, TS(2)]$에 갱신된 d_{id}를 저장하고 $TM_2[2]$에 TS(2)를 저장한다. 노드 5,7,8,9에 자신을 추가하여 <2, 2, d_{id}, $V_2(d_{id})$, TS(2), P(2)>를

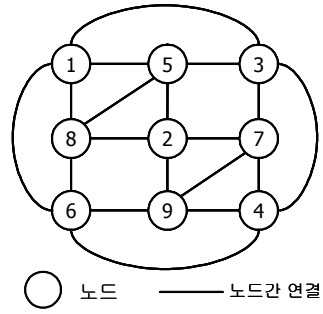

그림 6. 네트워크 토폴로지의 예

N(2)에게 전송한다. 메시지를 받은 노드 5는 TS(2)와 $TM_5[2]$를 비교한다. TS(2)가 $TM_5[2]$+1과 같으면 TTR_{max}를 TTR에 저장한 후, $H_5[2, TS(2)]$에 d_{id}를 저장하고 $TM_5[2]$에 TS(2)를 저장한다. $V_5(d_{id})$에 $V_2(d_{id})$를 저장하고 P(2)에 5를 추가하여 <5, 2, d_{id}, $V_2(d_{id})$, TS(2), P(2)>를 N(5)-P(2)인 노드 1,3,8에게 전송한다. TS(2)가 $TM_5[2]$와 같으면 이미 반영된 것이므로 전파를 종료하며, $TM_5[2]$+1 보다 크면 하나 이상의 push정보를 놓친 것이 되므로 노드 2에게 선택적 pull을 요청한다. 노드 7,8,9도 동일한 방법으로 push를 진행한다. [예 1] 끝

3.2 Pull 알고리즘

Pull 알고리즘은 일정 시점 이후에 갱신된 모든 노드의 갱신 정보를 전파하는 전체 pull 방식과 일정 시점 이후에 갱신된 특정 노드의 갱신 정보를 전파하는 선택적 pull 방식으로 구성된다. 노드가 네트워크에 접속할 때와 고립된 후 재접속할 때는 전체 pull을 사용하며, push 메시지와 alive 메시지를 통해 미반영된 정보를 발견한 경우에는 선택적 pull을 사용한다.

그림 7은 전체 pull 알고리즘의 과정을 나타낸다. 전체 pull 알고리즘을 실행할 경우 pull을 요청하는 시점 이전의 모든 노드들의 갱신정보를 전송받을 수 있다. 노드 i가 고립을 발견하면 Gnutella Ping/Pong 프로토콜[6,18]을 이용해 네트워크에 재접속하고, 확보된 이웃 노드 k에게 전체 pull 메시지 <i, TM_i>를 전송한다. 노드 k는 자신의 타임스탬프 테이블(TM_k)와 pull 메시지의 타임스탬프 테이블(TM_i)를 비교하여 노드 i의 TM_i 이후에 갱신된 정보로 구성된 전체 pull 응답 메시지인 <m, d_{id}, $V_m(d_{id})$, TS(m)>의 집합을 전송한다. 노드 i가 네트워크에 접속하여 전체 pull을 요청하는 경우도 위와

- 전체 pull 요청 수신: 노드 k가 i의 전체 pull 요청을 처리

```
RECEIVE_ENTIRE_PULL_REQUEST(i, TMi)
  PR={ };
  FOR (s=1; s≤n; s++)
    IF (s≠i && TMk[s] > TMi[s]) THEN
      FOR (t=TMi[s]+1; t≤TMk[s]; t++)
        PR = PR ∪ <s, Hk[s,t], Vk(Hk[s,t]), t>;
  Send(PR) to i;
```

- 전체 pull 응답 수신: 노드 i가 k에게서 받은 전체 pull의 결과를 처리하는 과정

```
RECEIVE_ENTIRE_PULL_RESPONSE(PR)
  FOR EACH (s, d, v, t) ∈ PR
    TMi[s]=t;
    Hi[s,t]=d;
    Vi(d)=v;
```

그림 7. 전체 pull 알고리즘

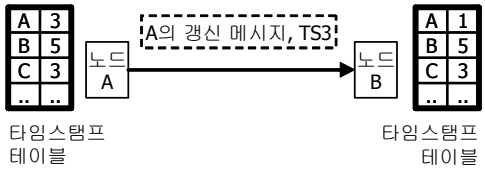

그림 8. 선택적 pull 요청이 발생하는 예

동일한 방법으로 진행된다.

선택적 pull은 push 메시지나 alive 메시지를 통해 미반영된 갱신 정보를 발견할 경우 실행한다. 그림 8에서 노드 A가 특정 데이터를 갱신하면 자신의 타임스탬프를 단조 증가한 후, 타임스탬프 테이블에 반영한다. 그리고 갱신을 전파하기 위해 push 메시지를 생성하여 노드 B에게 전송한다. 노드 B는 전송받은 push 메시지의 노드 A의 타임스탬프와 자신의 타임스탬프 테이블의 노드 A의 타임스탬프를 비교한다. 노드 B의 타임스탬프 테이블의 노드 A의 타임스탬프는 1이고 push 메시지의 타임스탬프는 3이므로 노드 B는 노드 A의 타임스탬프 2에서의 갱신 정보가 미반영 되었음을 인지하게 된다. 이 경우 노드 B는 노드 A에게 선택적 pull을 요청하여 미반영된 갱신 정보를 전송받는다.

선택적 pull 알고리즘의 자세한 과정이 그림 9에 나타난다. 노드 k가 선택적 pull을 이용해 반영되지 않은 노드 i의 갱신정보를 받기위해 push 메시지를 보낸 노드 m에게 선택적 pull 요청 메시지 <k, i, $TM_k[i]$>를 전송하면, 노드 m은 메시지의 타임스탬프 $TM_k[i]$이후에 갱신된 노드 i의 갱신 정보 <i, d_{id}, $V_m(d_{id})$, TS(i)>로 선택적 pull 응답 메시지를 생성하고 노드 k에게 전송한다.

- 선택적 pull 요청 수신: 노드 k가 m에게 i의 갱신 정보를 선택적 pull로 요청한 경우, m이 pull 요청을 처리하는 과정

```
RECEIVE_SELECTIVE_PULL_REQ(k, i, TMk[i])
  PR={ };
  FOR (t=TMk[i]+1; t≤TMm[i]; t++)
    PR = PR ∪ <i, Hm[i,t], Vm(Hm[i,t]), t>;
  Send(PR) to k;
```

- 선택적 pull 응답 수신: 노드 k가 m에게서 수신한 선택적 pull의 응답을 처리하는 과정

```
RECEIVE_SELECTIVE_PULL_RESPONSE(PR)
  FOR EACH (s, d, v, t) ∈ PR
    TMk[s]=t;
    Hk[s,t]=d;
    Vk(d)=v;
```

그림 9. 선택적 pull 알고리즘

3.3 고립 발견 알고리즘

비구조적 분산형 P2P 환경에서는 자원을 관리하는 중앙 서버가 없으므로 노드가 네트워크 장애나 이웃 노드들의 탈퇴로 인하여 이웃 노드와의 연결을 잃고 고립되는 문제가 발생할 수 있다. 그림 10은 alive 메시지의 송수신과 고립발견 알고리즘을 설명한다. 노드 i는 TTR이 0이 되면 이웃 노드에게 alive 메시지 <i, ps(i), $TM_i[ps(i)]$, type=0>를 전송하고 응답을 대기한다. Alive 메시지를 받은 노드는 메시지의 type이 0이면 그 값을 1로 하여 alive 메시지에 응답하고 $TM_i[ps(i)]$가 자신의 타임스탬프 테이블의 값 보다 크면 노드 i에게 선택적 pull을 요청한다. 노드 i는 타임아웃까지 대기한 후 고립 발견 과정을 실행한다. 타임아웃 동안 계산된 L(i)를 검사하여 연결이 최대 이웃 수의 반미만으로 떨어지면 부족한 이웃 노드를 확보한다. L(i)가 0으로 모든 연결을 잃은 경우에는 재접속한 후 임의의 이웃 노드에게 전체 pull을 요청한다. [예 2]는 그림 6의 노드 7이 alive 메시지를 이용하여 고립을 발견하는 과정을 설명한다.

[예 2] 노드 7의 현재 잔여 TTR을 1분으로 가정하자. TTR이 0이 될 동안 push를 받지 못하면 N(7)={2,3,4,9}에게 <7, ps(7), $TM_7[ps(7)]$, 0>를 전송한다. 노드 2,3,4,9는 메시지의 type이 0이므로 type을 1로 하여 노드 7에게 alive 메시지를 응답으로 전송하고, 자신의 TM[ps(7)]보다 $TM_7[ps(7)]$이 크면 노드 7에게 선택적 pull을 요청한다. 노드 7은 도착한 alive 메시지로 L(7)을 계산하고, 선택적 pull을 검사한다. 타임아웃이 지나면 고립 발견 과정을 실행하여 계산된 L(7)이 0이면 고립

```
• Alive 메시지 송신: 노드 i가 노드 k에게 alive
  메시지를 전송
  SEND_ALIVE( )
     To all k∈N(i), Send(i, ps(i), TMi[ps(i)], 0);
     Wait(TIMEOUT);   DETECT_ISOLATION( );

• Alive 메시지 수신: 노드 k가 노드 i로부터 alive
  메시지를 수신
  RECEIVE_ALIVE(i, ps(i), TMi[ps(i)], TYPE)
     IF (TYPE==1) THEN L(k)++;
     ELSE  Send(k, ps(k), TMk[ps(k)], 1) to i;
     IF (TMi[ps(i)] > TMk[ps(i)]) THEN
       Call Request_Selective_Pull(k, ps(i), TMk[ps(i)]) to i;

• 고립의 발견: 노드 i가 L(i)를 이용하여 고립을 발견
  DETECT_ISOLATION( )
     IF (L(i)==0) THEN
       N(i)=Gnutella_Ping_Protocol();
       Call Request_Entire_Pull(i, TMi) to a node∈N(i);
     ELSE IF (L(i) < 이웃 노드의 최대 수 / 2) THEN
       N(i)=Gnutella_Ping_Protocol();
     L(i)=0;
```

그림 10. 고립 발견 알고리즘

된 상태이므로 재접속한 후 확보된 이웃 노드에게 전체 pull을 요청한다. 만약, L(7)이 2미만이면 부족한 이웃 노드만 확보한다. [예 2] 끝

Ⅳ. 실험 모형

본 절에서는 제안한 알고리즘의 성능을 정량적으로 분석하기 위한 성능 평가 모형을 설명한다. 이를 위해 4.1절에서는 비구조적 분산형 P2P 환경을 모델링한 성능 평가 모형에 대해 설명하고, 4.2절은 모의실험에 사용된 입력 매개 변수를 설명한다. 마지막으로 4.3절은 성능 평가 지수에 대해 설명한다.

4.1 성능 평가 모형

본 논문에서 제안한 알고리즘과 기존의 갱신 전파 기법들의 성능을 비교하기 위한 P2P 환경의 성능 평가 모형이 그림 11에 나타난다. 성능 평가 모형은 이산-사건 실험 프로그램인 CSIM[19]을 이용하여 구현하였다.

비구조적 분산형 P2P 환경은 중앙 서버가 없고 노드들의 상호 동작으로 운영되기 때문에 모든 기능은 각 노드에서 독립적으로 수행된다. 각 노드는 Use 프로세스, Pull 프로세스, Update 프로세스로 구성된다. 프로세스별로 갱신 발생, 메시지 전파 및 반영, pull 발생 및 alive 메시지 발생의 역할을 수행한다.

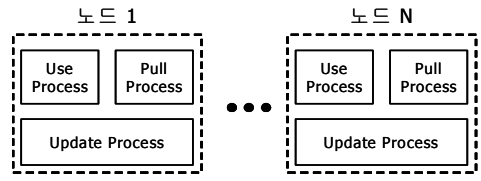

그림 11. P2P 환경의 성능 평가 모형

Use 프로세스는 노드가 생성될 때 할당받은 갱신을 실험시간 동안 발생한다. 갱신을 발생시키면 이웃 노드의 update 프로세스에게 push 메시지를 전송한다. Push 메시지를 수신한 노드의 update 프로세스는 메시지를 분류하여 해당 메시지가 push 메시지임을 인지하고 push 메시지를 분석하여 갱신 작업을 수행한다. Update 프로세스는 메시지를 수신하여 메시지를 분류한 후 메시지의 데이터를 분석하여 해당 작업을 수행하는 역할을 한다. 본 실험에서 발생하는 모든 메시지는 update 프로세스에게 전송 되고 update 프로세스는 수신한 메시지에 따라 작업을 수행한다. Pull 프로세스는 네트워크에 접속했을 때 전체 pull을 실행하며, TTR 시간 동안 수신된 push 메시지를 기반으로 고립발견 알고리즘을 수행한다.

4.2 입력 매개 변수

본 논문에서 사용되는 시스템 구성 변수와 노드 구성 변수를 표 2에서 요약한다. 전체 노드의 수와 이웃 노드의 수는 일반적인 P2P 시스템의 성능 평가 실험에서 사용되는 것과 같이 500개와 8개를 가정한다. 데이터 아이템의 수는 1000개이며 갱신 확률은 100%를 가정한다. 갱신될 데이터의 수는 NumDataItem × UpdateRate로 계산된다. 노드 접속 확률은 전체 노드 중에서 온라인 상태의 노드를 가리키며 일반적인 P2P 시스템이 20%의 접속 확률을 가지므로 본 연구에서도 20%의 접속 확률을 가정한다. 노드 탈퇴 확률은 온라인 상태의 노드가 탈퇴할 확률로써 0~50%를 가정한다. 메시지 분실 확률은 메시지를 전송할 때 메시지가 분실될 확률을 의미하며 0~30%를 가정한다. 접속 시간은 노드 유형이 join인 노드가 실험시간 중에 접속하는 시간으로 500~2000초를 가정한다. 탈퇴 시간은 노드 유형이 leave인 노드가 할당된 갱신을 다 수행하고 탈퇴하는 시간으로 50~200초를 가정한다. 갱신 시간은 네트워크에 접속 중인 노드가 할당된 갱신을 발생시키는 시간으로 120~240초를 가정한다. 노드 구성을 위한 변수로는 노드 유형이 있다. 노드 유형에는 항

표 2. 입력 매개 변수

CPUSpeed	노드 CPU의 속도	1 GIPS
NetBandWidth	네트워크 데이터 전송 속도	100 Mbps
NumNode	노드의 수	500 개
NumNeighbor	이웃 노드의 수	8 개
MinDiskTime	최소 디스크 액세스 시간	0.01 초
MaxDiskTime	최대 디스크 액세스 시간	0.03 초
NumDataItem	데이터의 수	1000 개
MsgInst	메시지당 명령 수	22000 개
PerIOInst	디스크 I/O를 위한 명령 수	5000 개
TTRMax	최대 TTR의 시간	100 초
OnlinRate	노드 접속 확률	20 %
UpdateRate	데이터 갱신 확률	100 %
LeaveRate	노드 탈퇴 확률	0~50 %
MsgLossRate	메시지 분실 확률	0~30 %
MsgDelay	메시지 전송 지연	1~5 ms
Join Duration	노드 접속 시간	500~2000 초
Leave Duration	노드 탈퇴 시간	50~200 초
Update Duration	갱신 발생 시간	120~240 초
NodeType	노드의 유형	on, join, leave

상 온라인 상태인 노드(on)와 실험 도중 네트워크에 접속하는 노드(join), 실험 도중에 탈퇴하는 노드(leave)로 구성된다.

4.3 성능 평가 지수

본 논문에서는 메시지의 분실이 잦은 불안정한 환경과 노드의 접속과 탈퇴가 빈번한 유동적인 환경에서 중복된 데이터의 갱신 전파를 수행하는 알고리즘들의 성능을 평가한다. 모의실험에서 사용되는 주요 성능 평가 지수는 모든 갱신의 전파가 완료되는데 까지 발생하는 전체 메시지 수와 각 노드별 미반영된 갱신 수이다. 노드 탈퇴 확률과 메시지 분실 확률에 따른 총 메시지 발생량과 미반영 갱신 수를 이용하여 성능을 비교하도록 한다.

V. 실험 결과 분석

본 절에서는 4절에서 설명한 성능 평가 모형을 이용하여 비구조적 분산형 P2P 환경에서 중복된 데이터의 갱신 전파 기법들을 비교한다. 본 논문에서 수행한 실험은 다음과 같다. 첫째, 제안한 알고리즘의 메시지 발생량 측면의 성능 검증을 위해 실험시간 동안 발생하는 전체 메시지의 양을 비교한다. 둘째, 갱신 전파의 효율성을 비교하기 위해 메시지 분실이 빈번한 환경에서 노드의 접속과 탈퇴에 따른 미반영 갱신 수를 측정하여 비교한다. 본 논문에서는 순수 push 기반 알고리즘(PO: Push Only)과 수신노드의 리스트를 push 메시지에 첨부하는 하이브리드 알고리즘(LRA: List of Receivers Approach), 그리고 본 논문에서 제안한 알고리즘(PRC: Peer-to-Peer Replication Control)을 구현하였다. Pull 기반 알고리즘은 시스템의 구조가 다르고 소유자 노드를 가정해야 하므로 비교하지 않는다.

5.1 실험 1: 메시지 분실 확률과 노드 탈퇴 확률에 따른 메시지 발생량

본 절에서는 제안한 알고리즘의 메시지 발생량 측면의 효율성을 검증한다. 실험을 위하여 메시지 분실 확률이 0.0인 안정적인 환경과 0.3인 불안정한 환경을 가정하고, 각 환경에서 노드 탈퇴 확률에 따른 메시지 발생량을 측정하여 각 알고리즘의 메시지 발생량 측면의 성능을 비교한다.

그림 12는 노드 탈퇴 확률에 따른 메시지 발생량을 측정한 실험 결과이다. 메시지 분실 확률을 0.0으로 주어 메시지 분실이 전혀 없는 안정적인 네트워크 상태를 가정한다. 이러한 환경에서 노드 탈퇴 확률을 0.0~0.5로 변화시키면서 결과를 측정하였다. 실험 결과 PO의 경우 평균 71만개의 메시지를 발생 시켰으며, LRA는 평균 36만개의 메시지를 발생시킴을 확인할 수 있었다. PRC의 평균 발생 메시지 수는 58만 여개로 LRA보다는 22만 여개가 많지만 PO보다는 13만개 정도의 메시지를 덜 발생시킴으로써 메시지 발생량의 관점에서 20%이상 효율적인 것으로 나타났다.

그림 13은 메시지 분실 확률을 0.3으로 주어 메시지 분실이 잦은 불안정한 네트워크 상태를 가정한다. 이러한 환경에서 노드 탈퇴 확률을 0.0~0.5로

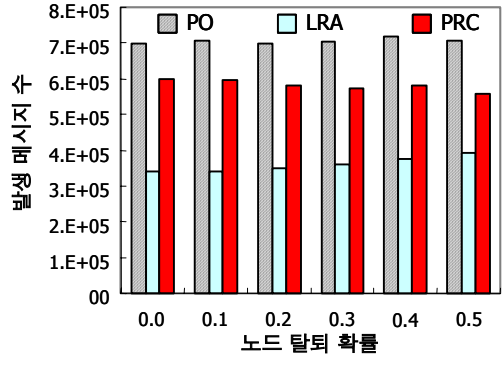

그림 12. 안정적인 환경에서 노드 탈퇴 확률에 따른 발생 메시지 수 (MsgLossRate = 0.0)

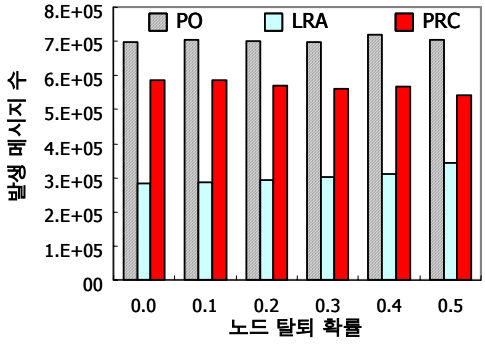

그림 13. 불안정한 환경에서 노드 탈퇴 확률에 따른 발생 메시지 수 (MsgLossRate = 0.3).

변화시키면서 결과를 측정하였다. 실험 결과 PO는 70만 여개, LRA는 30만 여개, PRC는 56만 여개로 그림 12에 비해 다소 적은 수치이다. PO와 PRC의 경우 메시지 분실로 인해 중복 전송되는 빈도가 줄어들기 때문이며, LRA의 경우 메시지 분실로 인해 갱신 정보를 전파 받지 못해 push 메시지의 전파를 진행 시키지 못하는 경우가 발생하기 때문이다. 결론적으로 각 알고리즘은 메시지 분실 확률과 노트 탈퇴 확률에 영향을 받지 않고 일정량의 메시지를 발생시킴을 확인할 수 있다.

5.2 실험 2: 갱신 전파의 효율성

본 절에서는 제안한 알고리즘의 갱신 전파 측면의 효율성을 검증하기 위하여 다음과 같은 실험을 수행한다. 첫째, 메시지 분실이 없는 안정적인 네트워크 환경에서 노드 탈퇴 확률에 따른 노드별 평균 미반영 갱신 수를 구한다. 둘째, 노드의 접속과 탈퇴가 없는 고정적인 네트워크 환경에서 메시지 분실 확률에 따른 노드별 평균 미반영 갱신 수를 구한다. 셋째, 메시지 분실 빈도가 잦은 불안정한 네트워크 환경에서 노드 탈퇴 확률에 따른 노드별 평균 미반영 갱신 수를 구한다. 넷째, 노드의 접속과 탈퇴가 빈번한 유동적인 네트워크 환경에서 메시지 분실 확률에 따른 노드별 평균 미반영 갱신 수를 구한다. 본 실험을 통해서 제안한 알고리즘이 분실 확률이 높은 불안정한 환경과 노드 탈퇴 확률이 높은 유동적인 환경에서도 갱신 전파를 보장함을 증명한다.

그림 14는 메시지 분실 확률을 0.0으로 주어 메시지 분실이 전혀 없는 안정적인 네트워크 환경에서 노드 탈퇴 확률을 0.0~0.5로 변화시키면서 노드별 평균 미반영 갱신 수를 측정한 결과이다. 메시지 분실 확률이 0.0인 안정적인 네트워크 환경이기 때문에 노드 탈퇴 확률이 0.0인 고정적인 네트워크를

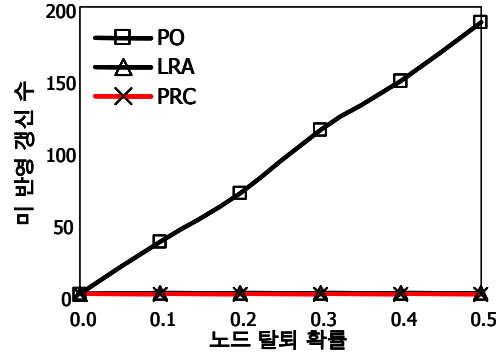

그림 14. 안정적인 환경에서 노드 탈퇴 확률에 따른 노드별 미반영 갱신 수 (MsgLossRate = 0.0)

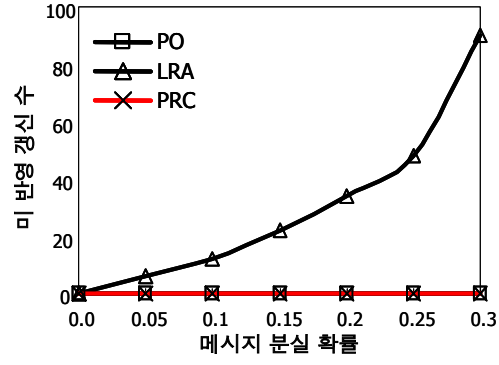

그림 15. 고정적인 환경에서 메시지 분실 확률에 따른 노드별 미반영 갱신 수 (LeaveRate = 0.0)

형성하는 경우에는 모든 노드가 갱신 정보를 완전히 전달 받는다. 그러나 노드 탈퇴 확률이 증가함에 따라 PO의 경우 미반영 갱신 수가 증가하는 것으로 나타났다. PO의 경우 pull을 전혀 사용하지 않기 때문에 실험 도중 접속한 노드는 접속시점 이전에 발생한 어떠한 갱신에 대해서도 그 내용을 전파 받을 수 없다. 이와는 달리 LRA와 PRC의 경우 실험 도중 접속한 노드는 전체 pull을 실행하므로 접속시점 이전의 모든 갱신 정보를 전달 받을 수 있고, 그 결과 0.5의 높은 노드 탈퇴 확률에도 대부분의 갱신 정보를 전파할 수 있었다.

그림 15는 노드 탈퇴 확률이 0.0으로 노드의 접속과 탈퇴가 없는 고정적인 네트워크 환경에서 메시지 분실 확률을 0.0~0.3으로 변화시키면서 노드별 평균 미반영 갱신 수를 측정한 결과이다. LRA의 경우 메시지 분실 확률이 증가함에 따라 미반영 갱신 수가 급증하는 결과를 나타내었다. 이러한 결과는 LRA 알고리즘이 push 메시지에 수신노드의 리스트를 첨부하기 때문에 발생한다. 즉, 메시지의 중복 전송을 줄이기 위하여 push 메시지에 수신노드

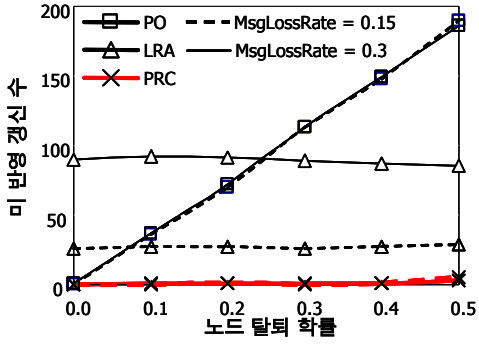

그림 16. 불안정한 환경에서 노드 탈퇴 확률에 따른 노드별 미반영 갱신 수 (MsgLossRate = 0.15 또는 0.3)

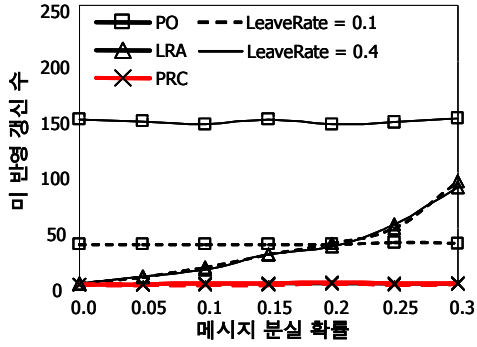

그림 17. 유동적인 환경에서 메시지 분실 확률에 따른 노드별 미반영 갱신 수 (LeaveRate = 0.1또는 0.4)

의 리스트를 첨부하였지만, 메시지 분실로 인한 갱신 미반영 노드가 리스트에 포함되는 상황을 고려하지 않기 때문이다. 반면에 PO와 PRC의 경우 메시지 분실 확률이 0.3으로 불안정한 환경에서도 대부분의 갱신정보가 전파되었다. PO의 경우 그림 13에서와 같이 평균 메시지 발생량이 70만 여개 이며 PRC의 경우 56만 여개로 PO보다 14만개 정도가 적기 때문에 동일한 성능을 보이는 PRC가 더 효율적임을 확인할 수 있다.

그림 16은 메시지 분실 확률이 0.15와 0.3으로 메시지의 분실이 빈번한 불안정한 네트워크 환경에서 노드 탈퇴 확률을 0.0~0.5로 변화시키면서 노드별 평균 미반영 갱신 수를 측정한 결과이다. PO의 경우 메시지 분실 확률이 0.15일 경우와 0.3일 경우 모두 유사한 결과를 나타내었다. 즉, PO는 메시지 분실 확률에 상관없이 노드 탈퇴 확률에만 영향을 받는 결과를 보인다. PO는 동일한 메시지를 중복 전송하는 경우가 많기 때문에 push 메시지가 분실되더라도 대부분의 갱신 정보의 전파를 보장한다. 반면에 노드 탈퇴 확률이 증가함에 따라 미반영 갱신 수가 증가함을 보였다. PO의 경우 실험 도중 접속한 노드는 접속 시점 이전의 갱신 정보를 전파 받지 못하기 때문에 노드 탈퇴 확률이 높아짐에 따라 미반영 갱신 수 또한 증가하게 된다.

LRA의 경우 메시지 분실 확률이 0.15인 환경에서는 노드 탈퇴 확률에 상관없이 노드별 평균 25여개의 미반영 갱신이 발생하며, 메시지 분실 확률이 0.3인 환경에서는 노드 탈퇴 확률에 상관없이 노드별 평균 90여개의 미반영 갱신이 발생하였다. LRA의 경우 실험 도중 접속한 노드는 임의의 노드에게 pull을 요청하여 접속 시점 이전의 갱신 정보를 전파 받을 수 있기 때문에 노드 탈퇴 확률이 높은 유동적인 환경에서도 성능에 큰 영향을 받지 않았다. 반면에 메시지 분실 확률이 0.15일 때와 0.3일 때, 각각 평균 25개와 90개의 미반영 갱신이 발생하였다. LRA는 동일한 push 메시지의 중복 전송을 줄이기 위해서 push 메시지에 수신 노드의 리스트를 첨부하여 리스트에 포함된 노드에게는 메시지를 전송하지 않는다. 따라서 메시지 분실로 인해 수신 노드의 리스트에 포함된 노드가 실제로는 갱신 내용을 전파 받지 못하는 경우가 발생한다. LRA는 메시지가 분실될 경우를 고려하지 않았기 때문에 메시지 분실 확률이 높아질수록 미반영 갱신 수 또한 증가하는 결과를 나타내었다.

PRC의 경우 메시지 분실 확률과 노드 탈퇴 확률에 영향을 받지 않고 대부분의 갱신 정보가 전파되는 결과를 보였다. 실험 도중 접속한 노드는 전체 pull을 사용하여 접속시점 이전의 모든 갱신을 전달받으므로 노드 탈퇴 확률이 0.5인 유동적인 환경에서도 갱신 전파를 보장한다. 또한, push와 alive 메시지의 타임스탬프 정보를 통해 미반영 갱신을 발견하면 선택적 pull을 실행하므로 메시지의 분실 확률이 0.3인 불안정한 환경에서도 대부분의 갱신 정보의 전파를 보장한다.

그림 17은 노드 탈퇴 확률이 0.1인 환경과 0.4인 환경에서 메시지 분실 확률을 0.0~0.3으로 변화시키면서 노드별 평균 미반영 갱신 수를 측정한 결과이다. 실험 결과에 대한 분석은 그림 16과 동일하다. 즉, PRC가 유동적이고 불안정한 네트워크에서도 대부분의 갱신 정보의 전파를 보장함을 확인할 수 있었다.

Ⅵ. 결 론

본 논문에서는 Gnutella와 같은 비구조적 분산형 P2P 네트워크에서 데이터 중복을 사용하여 발생되는 사본들의 일관성 문제를 해결하기 위하여 타임스

탬프 기반의 하이브리드 push/pull 알고리즘을 제안하였다. 제안한 알고리즘의 특징은 다음과 같다. 첫째, 각 노드의 타임스탬프를 이용하여 중복된 데이터의 일관성을 유지한다. 갱신 정보를 타임스탬프와 함께 전송함으로써 서로 다른 노드들의 갱신 정보에 대해 전송된 타임스탬프 시점까지는 일관성을 보장할 수 있다. 둘째, push 메시지에 송신노드의 리스트를 추가하여 전송한다. 송신노드의 리스트를 전송함으로써 메시지 분실이나 노드 실패로 인해 갱신 정보를 전파 받지 못하는 노드가 발생되는 경우를 방지하여 완전한 전송을 보장한다. 또한 송신노드의 리스트에 포함된 노드에게는 동일한 메시지를 전송하지 않음으로써 메시지의 중복 전송을 줄일 수 있다. 셋째, 고립발견 알고리즘을 주기적으로 실행하여 안정적인 네트워크 상태를 유지한다. 이웃 노드와의 일정 수 이상의 연결을 보장함으로써 노드의 고립을 미연에 방지하며 노드 실패나 메시지 분실에 대해 내구력 있는 네트워크 환경을 구성할 수 있다.

본 논문에서는 제안한 알고리즘에 대해 다양한 환경에서 실험을 수행하였으며, 실험 결과를 요약하면 다음과 같다. 첫째, 노드 탈퇴 확률이 높은 유동적인 환경에서 갱신정보의 전파를 보장한다. 새롭게 접속한 노드는 전체 pull을 실행하여 접속시점 이전의 갱신정보를 반영함으로써 접속과 탈퇴가 빈번한 환경에서도 갱신정보 전파를 보장한다. 둘째, 메시지 분실 확률이 높은 불안정한 환경에서 갱신정보의 전파를 보장한다. Push 메시지의 타임스탬프 정보를 이용하여 미반영 갱신정보를 발견하고 선택적 pull을 실행함으로써 메시지 분실 확률이 높은 불안정한 환경에서도 안정적인 갱신정보 전파를 보장한다. 셋째, 메시지 발생량을 줄인다. Push 메시지에 송신노드의 리스트를 첨부하여 리스트에 포함된 노드에게는 메시지를 전송하지 않음으로써 메시의 중복 전송을 줄인다.

본 논문에서 제안한 알고리즘은 노드의 접속과 탈퇴가 빈번한 P2P 네트워크의 특징을 고려하였기 때문에 기존에 제안된 갱신전파 기법들과 비교하였을 때 그 성능이 우수함을 확인 할 수 있다. 또한 무선 ad-hoc 네트워크나 센서 네트워크와 같이 메시지 분실이 빈번한 불안정한 환경에서도 좋은 성능을 보일 수 있음을 확인할 수 있다.

참 고 문 헌

[1] Q. Lv, et al, “Search and Replication in Unstructured Peer-to-Peer Networks,” *Proc. of 16th ICS*, 2002.

[2] K. Aberer and M. Hauswirth, “An Overview on Peer-to-Peer Information Systems,” *Proc. of Workshop on Distributed Data and Structures*, 2002.

[3] I. Stoica, et al, “Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications,” *IEEE Trans. Networking*, 11(1), 2003.

[4] S. Ratnasamy, et al, “A Scalable Content-Addressable Network,” *Proc. of ACM SIGCOMM*, 2001.

[5] K. Aberer, et al, “P-Grid : A Self- organizing Structured P2P System,” *SIGMOD Record*, 32(3), 2003.

[6] S. Androutsellis-Theotokis, and D. Spinellis, “A Survey of Peer-to-Peer Content Distribution Technologies,” *ACM Computing Surveys*, 36(4), 2004.

[7] K. Samant and S. Bhattacharyya, “Topology, Search, and Fault Tolerance in Unstructured P2P Networks,” *Proc. of 37th IEEE Conf. on System Sciences*, 2004.

[8] B. Yang and H. Garcia-Molia, “Improving Search in Peer-to-Peer Networks,” *Proc. of 22nd ICDCS*, 2002.

[9] E. Cohen and S. Shenker, “Replication Strategies in Unstructured Peer-to-Peer Networks,” *Proc. of ACM SIGCOMM*, 2002.

[10] F. M. Cuenca-Acuna, R. P. Martin, and T. D. Nguyen, “Autonomous Replication for High Availability in Unstructured P2P Systems,” *Proc. of 22nd SRDS*, 2003.

[11] V. Gopalakrishnan, et al, “Adaptive Replication in Peer-to-Peer Systems,” *Proc. of 24th ICDCS*, 2004.

[12] X. Liu, G. Yang, and D. Wang, “Stationary and Adaptive Replication Approach to Data Availability in Structured Peer-to-Peer Overlay Networks,” *Proc. IEEE ICON*, 2003.

[13] K. Ranganathan, A. Iamnitchi, and I. Foster, “Improving Data Availability through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities,” *Proc. of 2nd CCGRID*,

2002.

[14] A. Datta, M. Hauswirth, and K. Aberer, "Updates in Highly Unreliable, Replicated Peer-to-Peer Systems," *Proc. of 23rd ICDCS*, 2003.

[15] J. Lan, et al, "Consistency Maintenance In Peer-to-Peer File Sharing Networks," *Proc. of 3rd IEEE Workshop on Internet Applications*, 2003.

[16] J. Holliday, et al, "Epidemic Algorithms for Replicated Databases," *IEEE Trans. Knowledge and Data Eng*., 15(3), 2003.

[17] U. Cetintemel, et al, "Deno: A Decentralized, Peer-to-Peer Object- Replication System for Weakly Connected Environments," *IEEE Trans. Computers*, 52(7), 2003.

[18] *The Gnutella Protocol Specification* v0.4, Document Revision 1.2.

[19] H. Schwetmann, *User's Guide of CSIM18 Simulation Engine*, Mesquite Software, Inc. 1996.

최 민 영 (Minyoung Choi) 준회원

2004년 영남대학교 컴퓨터공학과 (공학사)

2006년 영남대학교 컴퓨터공학과 (공학석사)

2006년~현재 LG CNS 공공사업부 응용개발 1팀 System Engineer

<관심분야> P2P 시스템, 분산 데이터베이스, 성능 평가

조 행 래 (Haengrae Cho) 정회원

1988년 서울대학교 컴퓨터공학과 (공학사)

1990년 한국과학기술원 전산학과 (공학석사)

1995년 한국과학기술원 전산학과 (공학박사)

1995년~현재 영남대학교 전자정보공학부 교수

<관심분야> Mobile Computing, 분산 데이터베이스, 고성능 트랜잭션 처리, 성능 평가