# Peer-to-Peer 환경에서 중복된 데이타의 갱신 전파 기법

최민영° 조행래
영남대학교 컴퓨터공학과
chalscse°@yumail.ac.kr, hrcho@yu.ac.kr

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

Minyoung Choi° Haengrae Cho
Department of Computer Engineering, Yeungnam University

## 요 약

P2P(Peer-to-Peer) 시스템은 대용량의 데이타를 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 원하는 자원을 액세스하는 비용도 증가하는 문제를 가진다. 데이타 중복을 이용해 이러한 문제를 해결할 경우 중복된 데이타들의 일관성 유지를 위한 기법이 필요하다. 본 논문에서는 특정 노드가 갱신한 데이타를 중복된 사본을 저장하고 있는 다른 노드에 전파하기 위한 하이브리드 push/pull 기반의 갱신 전파 기법을 제안한다.

## 1. 서 론

P2P(Peer-to-Peer) 시스템은 각 노드들이 가지고 있는 자원을 상호간의 직접적인 연결을 통하여 교환할 수 있는 네트워크 환경으로 그 구조에 따라 중앙 집중형 네트워크와 분산형 네트워크로 분류된다[7]. Napster[1]와 같은 중앙 집중형 네트워크는 서버가 노드들의 IP 주소와 공유 파일의 인덱스 및 메타 데이타를 관리하는 방식이다. 중앙 집중형 네트워크는 서버가 모든 노드들의 정보를 관리하기 때문에 검색은 빠르지만, 서버에 병목현상이 발생하며 서버가 동작하지 않을 경우 전체 시스템이 중단되는 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[9]와 CAN[8] 알고리즘을 들 수 있다. 각 노드는 해쉬 함수를 이용하여 키를 분배 받으며, 키를 이용하여 파일을 검색한다. 구조적 방식은 검색 비용이 적은 반면, 노드들이 서로 의존적이므로 연결과 탈퇴에 따른 추가적인 비용이 발생한다. Gnutella[1]와 같은 비구조적인 방식은 노드들의 연결이 임의적이고 파일의 배치가 규칙적이지 않다. 검색을 위한 메시지가 많이 발생하지만, 노드들 사이의 의존성이 낮아서 연결과 탈퇴에 큰 영향을 받지 않고 확장성이 좋은 장점이 있다.

표준화를 통해 P2P 네트워크의 규모가 충분히 커질 경우에는 규칙적인 토폴로지의 적용이 어려워지므로 확장성이 좋은 비구조적 분산형 P2P 네트워크가 주를 이루게 될 것이다[2]. 이러한 구조에서 검색의 효율성과 응답시간의 단축을 위해 데이타 중복이 필요하며 많은 연구를 통해 그 효율성이 입증되었다[7]. 데이타 중복을 이용함으로써 시스템의 내구성과 유용성이 증대되고 응답시간이 단축되며, 성능저하의 원인인 병목현상, 노드실패, 검색실패 등의 문제점을 해결할 수 있다.

이러한 관점에서 본 논문에서는 Gnutella와 같은 비구조적 분산형 P2P 네트워크에서 데이타 중복을 사용함으로써 발생하는 사본들의 일관성 문제를 해결하기 위해 타임스탬프 기반의 하이브리드 push/pull 알고리즘을 제안한다. 제안한 알고리즘은 노드들의 타임스탬프와 송신노드의 리스트를 push 메시지에 첨부하고, 전체 pull과 선택적 pull을 적극적으로 활용함으로써 메시지의 중복전송을 줄이고 중복된 데이타의 일관성을 효율적으로 유지시킬 수 있다는 장점을 갖는다.

## 2. 관련 연구

일반적으로 P2P 환경에서는 지역 노드에서 우선으로 갱신한 후 다른 노드들에게 전파하는 지연갱신 방식을 사용하며, 전파방법에 따라 push와 pull의 두 가지 기법이 있다. Push는 갱신을 발생시킨 노드가 이웃 노드에게 갱신정보를 전파하는 방법이고, pull은 갱신하지 않은 노드가 갱신을 발생시킨 노드에게 갱신내용을 요청하는 방법이다. 하이브리드 push/pull은 push와 pull을 모두 이용하는 방법이다.

[5]에서는 각 노드의 갱신정보로 로그를 구성하여 주기적으로 다른 노드들에게 전파하는 push 기반의 전염 알고리즘을 제안하였다. Push 메시지를 받은 노드는 갱신정보를 다시 이웃노드에게 전파하는 과정을 반복함으로써 전체 네트워크의 모든 노드들에게 갱신정보가 전파될 수 있지만, 메시지의 중복전송이 발생하는 문제점이 있다.

Deno 시스템[3]은 pull을 이용하여 노드들 간의 이벤트 정보를 동기화하였다. 이 경우 Pull을 요청할 대상이 최신 데이타를 가짐을 보장할 수 없기 때문에 최신 데이타를 찾는데 걸리는 시간이 길어지며 불필요한 메시지가 발생되는 문제점이 있다. 요청 주기에 따라 발생하는 메시지의 양과 갱신을 인지하는 시간사이의 상관관계가 있기 때문에 비효율적인 경우가 발생한다.

[6]에서는 중앙 소유자[1]를 가정한 무효화 메시지 기반의 하이브리드 push/pull 알고리즘을 제안하였다. 중앙 소유자로부터 일정 홉 내의 노드는 push를 통해 무효화 메시지를 전달받고, 나머지 노드는 pull을 적응적으로 사용하여 갱신 정보를 전달받는다. 이 경우 모든 갱신이 중앙 소유자를 통해서 전파되므로 중앙 소유자가 오프라인이면 갱신할 수 없는 문제가 발생하며, push를 사용함으로써 발생되는 메시지의 중복전송을 해결하지 못한다.

[4]에서는 push 메시지에 수신노드의 리스트를 포함하여 전송하고, 일정시간 push가 없거나 재연결할 경우 pull을 요청하는 하이브리드 push/pull 알고리즘을 제안하였다. 수신노드의

1) "중앙 소유자"는 여러 노드들에 중복된 데이타의 소유자 노드를 의미하며, 갱신된 데이타의 전파는 중앙 소유자를 통해서만 가능하다.

리스트에 포함된 노드에게는 메시지를 전송하지 않음으로써 중복전송을 줄일 수 있지만, 네트워크 장애나 노드의 빈번한 연결과 탈퇴로 인해 갱신을 전파 받지 못한 노드가 수신노드의 리스트에 포함되는 문제가 발생한다. 또한, pull을 실행하는 시점과 요청 대상에 대한 상세 알고리즘은 기술하지 않고 있다. 본 논문에서는 [4]의 아이디어를 확장하여 push 메시지에 송신 노드의 리스트를 전송하고, pull을 적극적으로 이용하는 하이브리드 push/pull 알고리즘을 제안한다.

## 3. 하이브리드 push/pull 기반의 갱신 전파 기법

본 절에서는 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 설명이다. 각 노드마다 타임스탬프가 할당되며, 네트워크 전체 노드의 타임스탬프 테이블을 유지한다. $H_i[k, TS(k)]$에서 $H_i$는 노드 i가 유지하고 있는 전체 노드의 갱신 이력 테이블이며, 그 값은 노드 k가 TS(k)번째 갱신한 데이타 식별자를 나타낸다. $H_i$에서 모든 노드의 최종 TS 값을 모아둔 것이 $TM_i$이며, 노드 k의 값은 $TM_i[k]$이다.

표 1. 알고리즘의 매개변수와 설명

| 매개변수 | 설 명 |
|---|---|
| S | 전체 네트워크에 존재하는 모든 노드 n개의 집합 |
| TTR | Alive 메시지의 전송 주기로 초기값은 TTRmax |
| 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 노드들의 TS 벡터 |
| $TM_i[k]$ | $TM_i$에서 노드k의 타임스탬프 |
| $d_{id}$ | 데이타 식별자 |
| ps(i) | 노드 i가 마지막으로 전파한 push의 발생노드 |
| $V_i(d_{id})$ | 노드 i가 유지하고 있는 $d_{id}$의 value |
| $H_i[k, TS(k)]$ | 노드 i가 유지하고 있는 갱신 이력 테이블로써, 노드 k가 TS(k)시점에서 갱신한 $d_{id}$를 저장 |

### 3.1 Push 알고리즘

노드 i가 $d_{id}$를 갱신할 경우 갱신을 반영하고 이웃 노드에게 갱신 정보를 전달해야 한다. 그림 1은 push 알고리즘의 과정을 설명한다.

```
• Push 발생: i가 d_id를 new_value로 갱신하고 push하는 과정
  UPDATE(d_id, new_value)
    TS(i)++; H_i[i, TS(i)]=d_id; TM_i[i]=TS(i);
    V_i(d_id)=new_value; P(i)={i};
    To all k∈N(i), Send(i, i, d_id, V_i(d_id), TS(i), P(i));

• Push 수신: k가 m으로부터 i의 갱신에 대한 push를 수신하는 과정
  RECEIVE_PUSH(m, i, d_id, V_i(d_id), TS(i), P(i))
    IF (TS(i)==TM_k[i]+1) THEN
      TTR=TTRmax;
      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 1. push 알고리즘

갱신 노드 i는 자신의 타임스탬프를 단조증가 시키고 N(i)에게 갱신된 내용을 push한다. Push를 받은 노드는 자신의 TM과 메시지의 타임스탬프를 비교하여 반영, 전파종료 또는 선택적 pull을 결정한다. 메시지의 타임스탬프가 TM의 값 보다 1이 크면 TTR을 TTRmax로 설정한 후 push를 반영하고 P(i)에 자신을 포함하여 전파한다. TM의 값과 같으면 중복 전송된 메시지이므로 전파를 종료하며, TM의 값보다 2이상 클 경우에는 수신하지 못한 메시지가 존재하므로 push한 노드에게 선택적 pull을 요청한다. **[예 1]**은 그림 2의 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 N(2)에게 전송한다. 메시지를 받은 노드 5는 TS(2)와 $TM_5[2]$를 비교한다. TS(2)가 $TM_5[2]$+1과 같으면 TTRmax를 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**

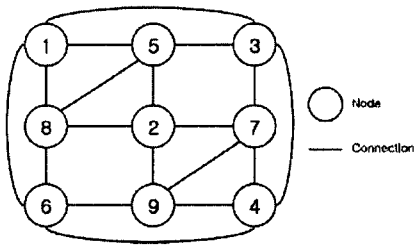


그림 2. 네트워크 토폴로지의 예

### 3.2 Pull 알고리즘

Pull 알고리즘은 전체 pull 방식과 선택적 pull 방식으로 구성된다. 노드가 네트워크에 처음 연결할 때와 고립된 후 재연결할 때는 전체 pull을 사용하며, push와 alive 메시지를 통해 미반영된 정보를 발견하면 선택적 pull을 사용한다. 전체 pull의 경우, 노드 i가 고립을 발견하면 Gnutella Ping/Pong 프로토콜[2]을 이용해 재연결하고, 확보된 이웃 노드 k에게 <i, $TM_i$>를 전송한다. 노드 k는 $TM_k$와 $TM_i$를 비교하여 i에게 $TM_i$이후의 갱신들로 구성된 <m, $d_{id}$, $V_m(d_{id})$, TS(m)>의 집합을 전송한다. 그림 3은 전체 pull 알고리즘의 과정을 설명한다.

```
• 전체 pull 요청 수신: k가 i의 전체 pull 요청을 처리하는 과정
  RECEIVE_ENTIRE_PULL_REQUEST(i, TM_i)
    PR={ };
    FOR (s=1; s≤n; s++)
      IF (s≠i && TM_k[s] > TM_i[s]) THEN
        FOR (t=TM_i[s]+1; t≤TM_k[s]; t++)
          PR=PR∪<s, H_k[s,t], V_k(H_k[s,t]), t>;
    Send(PR) to i;

• 전체 pull 응답 수신: i가 k에게서 받은 전체 pull의 결과를 처리하는 과정
  RECEIVE_ENTIRE_PULL_RESPONSE(PR)
    FOR EACH (s, d, v, t)∈PR;
      TM_i[s]=t; H_i[s,t]=d; V_i(d)=v;
```

그림 3. 전체 pull 알고리즘

노드 k가 선택적 pull을 이용해 반영되지 않은 노드 i의 갱신정보를 받기위해 push 메시지를 보낸 노드 m에게 <k, i, $TM_k[i]$>를 전송하면, 노드 m은 $TM_k[i]$이후에 갱신된 i의 정보로 구성된 <i, $d_{id}$, $V_m(d_{id})$, TS(i)>의 집합을 전송한다. 그림 4는 선택적 pull 알고리즘의 과정을 설명한다.

```
• 선택적 pull 요청 수신 : k가 m에게 i의 갱신정보를 선택적 pull
  방식으로 요청한 경우, m이 pull의 요청을 처리하는 과정
  RECEIVE_SELECTIVE_PULL_REQUEST(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: 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 4. 선택적 pull 알고리즘

3.3 고립의 발견

네트워크 장애나 노드들의 탈퇴로 인하여 이웃 노드와의 연결을 잃고 고립되는 문제가 발생할 수 있다. 노드 i는 TTR이 0이 되면 이웃 노드에게 <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 메시지를 전송하고 $TM_i$[ps(i)]가 자신의 TM 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 일정 수준 이하로 떨어지면 부족한 이웃 노드를 확보한다. L(i)가 0으로 모든 연결을 잃은 경우에는 재연결한 후 임의의 이웃 노드에게 전체 pull을 요청한다. 그림 5는 alive 메시지의 송수신과 고립발견 알고리즘을 설명한다. **[예 2]**는 그림 2의 노드 7이 alive 메시지를 이용해 고립을 발견하는 과정을 설명한다.

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

• Alive 메시지 수신: k가 i로부터 alive 메시지를 수신하는 과정
  RECEIVE_ALIVE(i, ps(i), TMi[ps(i)], TYPE)
    IF (TYPE==1) 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: i가 L(i)이용해 고립을 발견하는 과정
  DETECT_ISOLATION( )
    IF (L(i)==0) THEN
      N(i)=Gnutella_Ping_Protocol();
      Call Request_Entire_Pull(i, TMi) to a node among N(i);
    ELSE IF (L(i) is smaller than half of full connectivity)
      THEN N(i)=Gnutella_Ping_Protocol();
    L(i)=0;
```

그림 5. 고립과 메시지분실 발견 알고리즘

**[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 고립상태 이므로 재연결한 후 확보된 이웃노드에게 전체 pull을 요청한다. 만약, L(7)이 2미만이면 부족한 이웃노드만 확보한다.**[예 2]** 끝

3.4 알고리즘 고찰

본 논문에서 제안한 알고리즘의 특징은 다음과 같다. 첫째, 각 노드의 타임스탬프를 이용해 중복된 데이타의 일관성을 유지한다. 갱신정보를 타임스탬프와 함께 전송함으로써 서로 다른 노드들의 갱신정보에 대해 타임스탬프의 정보까지는 일관성을 보장할 수 있다. 둘째, push 메시지에 송신노드의 리스트를 추가하여 전송한다. 수신노드의 리스트를 전송하는 방법은 메시지를 적게 발생시키는 반면에 메시지 손실과 장애로 갱신 내용을 전달 받지 못한 노드가 수신노드의 리스트에 포함되는 문제가 있다. 송신노드의 리스트를 전송함으로써 메시지의 중복전송을 줄이는 동시에 장애와 메시지 분실로 push를 전송받지 못한 노드를 줄일 수 있지만, 수신노드의 리스트를 전송하는 방식보다 메시지의 중복전송이 증가하는 단점을 가진다. 본 논문에서는 pull을 적극적으로 활용하므로 수신노드의 리스트를 전송하는 방식을 적용하여도 좋은 성능을 보일 것으로 예상된다. 마지막으로, 고립발견 알고리즘을 이용해 고립과 메시지 분실을 발견한다. 이것은 본 논문에서 제안한 기법이 노드의 추가 및 삭제가 빈번한 P2P 환경에 적합하며, 메시지 분실이 빈번한 무선 ad-hoc 네트워크나 센서 네트워크 기반의 P2P 환경도 지원할 수 있음을 의미한다.

## 4. 결론 및 향후과제

본 논문에서는 Gnutella와 같은 비구조적 분산형 P2P 네트워크에서 데이타 중복을 사용하여 발생되는 사본들의 일관성 문제를 해결하기 위해 타임스탬프 기반의 하이브리드 push/pull 알고리즘을 제안하였다. 기존의 알고리즘은 노드들의 연결과 탈퇴가 빈번한 P2P환경에는 좋은 성능을 나타내기 어렵다. 본 논문에서 제안한 알고리즘은 노드들의 타임스탬프와 송신노드의 리스트를 push 메시지에 첨부하고, 전체 pull과 선택적 pull을 적극적으로 활용함으로써 중복된 데이타의 일관성을 효율적으로 유지할 수 있다. 본 논문의 향후과제는 제안한 알고리즘을 구현하고 성능을 정량적으로 분석하여 기존의 알고리즘들과 비교하는 것이다.

## 참 고 문 헌


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

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

[3] U. Cetintemel, P. J. Keleher, B. Bhattacharjee, and M. J. Franklin, "Deno: A Decentralized, Peer-to-Peer Object- Replication System for Weakly Connected Environments," IEEE Trans. Computers, 52(7), 2003.

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

[5] J. Holliday, R. Steinke, D. Agrawal, and A. E. Abbadi, "Epidemic Algorithms for Replicated Databases," IEEE Trans. Knowledge and Data Eng., 15(3), 2003.

[6] J. Lan, X. Liu, P. Shenoy and K. Ramamritham, "Consistency Maintenance In Peer-to-Peer File Sharing Networks," Proc. of 3rd IEEE Workshop on Internet Applications, 2003.

[7] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks," Proc. of 16th ICS, 2002.

[8] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker, "A Scalable Content-Addressable Network," Proc. of ACM SIGCOMM Conference on Application, 2001.

[9] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications," IEEE Trans. Networking, 11(1), 2003.