모바일 P2P 환경에서 효율적인 네트워크 자원 활용을 위한 반응적인 코드

(A Reactive Chord for Efficient Network Resource Utilization in Mobile P2P Environments)

윤 영 효 † 곽 후 근 †† 김 정 길 ††† 정 규 식 ††††
(Younghyo Yoon) (Hukeun Kwak) (Cheongghil Kim) (Kyusik Chung)

요 약 분산 해쉬 테이블(DHT : Distributed Hash Table) 기반의 P2P는 기존 Unstructured P2P 방식의 단점을 보완하기 위한 방식이다. DHT 알고리즘을 사용하면 빠른 데이터 검색을 할 수 있고, 피어 개수에 무관하게 검색 효율을 유지할 수 있다. DHT 방식의 피어들은 라우팅 테이블을 최신으로 유지하기 위해 주기적으로 메시지를 보낸다. 모바일 환경의 경우, DHT방식의 피어들은 라우팅 테이블을 최신으로 유지하고 요청 실패를 줄이기 위해서 빠른 주기로 메시지를 보내야 한다. 하지만 이로 인해, 전체 네트워크의 트래픽은 증가하게 된다.

본 논문에서는 리액티브 라우팅 테이블 업데이트 방식을 이용하여 기존 DHT에서의 라우팅 테이블 업데이트에 따른 부하를 줄이는 기법을 제안한다. 주기적으로 자신의 라우팅 테이블을 업데이트하는 기존 방식(Proactive)과 달리, 제안된 방식에서는 데이터 요청이 들어 왔을 때만 라우팅 테이블을 업데이트하는 방식(Reactive)을 사용한다. 제안된 방식은 버클리 대학에서 만들어진 Chord 시뮬레이터(I3)를 이용하여 실험을 수행하였다. 실험을 통하여 제안된 방식이 기존 방식에 비해 성능이 향상되었음을 확인하였다.

키워드 : 모바일 환경, 분산 해쉬 테이블, Chord, 리액티브, 프로액티브, 라우팅 테이블 업데이트

Abstract A DHT(Distributed Hash Table) based P2P is a method that compensates disadvantages of the existing unstructured P2P method. If a DHT algorithm is used, it can do fast data search and maintain search efficiency independent of the number of peers. The peers in a DHT method send messages periodically to keep the routing table updated. In a mobile environment, the peers in a DHT method should send messages more frequently to keep the routing table updated and reduce the failure of requests. However this results in increasing the overall network traffic.

In this paper, we propose a method to reduce the update load of a routing table in the existing DHT by updating it in a reactive way. In the proposed reactive method, a routing table is updated only if a data request is coming whereas it is updated periodically in the existing proactive method. We perform experiments using Chord simulator(I3) made by UC Berkely. The experimental results show the performance improvement of the proposed method compared to the existing method.

Key words : Mobile Environment, Distributed Hash Table, Chord, Reactive, Proactive, Routing Table Update

· 본 연구는 숭실대학교 교내 연구비 지원으로 이루어졌음
· 이 논문은 2008 한국컴퓨터종합학술대회에서 '모바일 환경을 위한 리액티브 방식의 Chord'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 숭실대학교 정보통신전자공학부
yyhpower@q.ssu.ac.kr
†† 정 회 원 : 숭실대학교 정보통신전자공학부 Postdoc
gobarian@q.ssu.ac.kr
(Corresponding author임)
††† 정 회 원 : 남서울대학교 컴퓨터학과 교수
cgkim@nsu.ac.kr
†††† 종신회원 : 숭실대학교 정보통신전자공학부 교수
kchung@q.ssu.ac.kr

논문접수 : 2008년 8월 25일
심사완료 : 2008년 11월 3일

정보과학회논문지: 정보통신 제36권 제2호(2009.4)

1. 서 론

Peer-to-Peer(P2P)는 기존의 인터넷에서 사용되던 클라이언트-서버 환경의 단방향 특성을 극복하고, 서버의 문제로 인해 사용자들이 서비스를 받지 못하게 되는 단일장애점(Single Pont Of Failure) 문제를 해결하기 위해 고안된 분산 컴퓨팅 모델에 기반한 네트워킹 기술이다. P2P 어플리케이션은 정보공유, 파일공유, 대역폭 효율화, 데이터 스토리지, 컴퓨팅 파워 공유 등 많은 부분에서 사용되며, 그 안에서 자원을 관리하는 대안을 제시하고 있다. 특히, P2P 파일공유 어플리케이션은 파일 교환을 위해 사용자들에게 많이 사용되어지고 있다. 대표적인 P2P 파일공유 어플리케이션으로는 eDonkey[1], Gnutella[2], Bit-torrent[3] 등의 어플리케이션들이 있다.

P2P 구조는 크게 Unstructured P2P와 Structured P2P 두 가지로 분류할 수 있다. Unstructured P2P는 전체 네트워크에 대한 정보들이 모든 피어(노드)들에 의해 관리 되거나 한 피어에게 집중되는 반면에, Structured P2P 시스템은 각각의 피어가 전체 네트워크가 아닌 부분적인 네트워크 정보를 유지 및 관리하게 하여 Unstructured P2P 시스템의 단점을 보완한 방식이다. Unstructured P2P에는 Napster[4], Gnutella, Freenet[5] 등 많은 프로토콜이 사용 되고 있다. 그렇지만, 이러한 방식에서는 Flooding 방식으로 인한 많은 트래픽 발생이 큰 문제가 되고 있다. 그러한 문제점을 고치기 위하여 Unstructured P2P 구조에서는 Hybrid P2P 방식을 제안하였다.

Unstructured P2P와는 다르게, Structured P2P에서는 분산 인덱싱을 제공하는 분산 해시 테이블(DHT : Distributed Hash Table)로 컨텐츠와 피어 정보들을 공통의 단일 주소 공간으로 매핑하여 콘텐츠 저장 및 검색이 이루어지는 분산 구조의 컨텐츠-어드레싱 기반 데이터 저장 기법을 제시한다.

DHT를 이용한 검색 기법이나, DHT 관리 기법에 따라 여러 가지 어플리케이션 사례가 존재한다. CAN[6], Pastry[7], Tapestry[8], Chord[9] 등 많은 방식들이 DHT 방식으로 제안되었다. 본 논문에서는, DHT 알고리즘 중 많은 곳에서 연구가 진행 되고 있고, 간단하고 쉬운 알고리즘인 Chord를 기반으로 연구를 수행하였다.

Chord[9,10]는 버클리 대학에서 만들어진 방식으로, n-bit의 원형 주소 공간에 각각의 노드와 데이터의 키 값을 할당하는 방식이다. 각각의 노드와 데이터의 키 값으로부터 해싱 함수를 이용하여 주소 공간으로의 매핑이 이루어지며, 또한 주소 공간 내에 존재하는 데이터 키 k 보다 크거나 같은 노드를 k의 Successor라고 명명하여 라우팅의 효율을 높이는 데 사용한다.

본 논문에서는 기존의 Chord가 가지는 문제점을 분석하고, 이를 해결하기 위해 새로운 Reactive 테이블 업데이트 기법을 제안한다. 본 논문의 구성은 다음과 같다. 제2장에서는 Structured P2P와 기존 Chord에 대한 이론과 연구방향 그리고 문제점을 소개한다. 3장에서는 기존 Chord에서의 문제점을 해결하는 새로운 기법을 설명하고, 4장에서는 실험 및 토론을, 5장에서는 결론 및 향후 연구 방향을 제시한다.

2. 기존 연구

2.1 Structured P2P (DHT : Distributed Hash Table)

DHT를 사용하여 만들어진 P2P 네트워크는 하나의 오버레이 네트워크가 만들어지게 되고, 데이터 검색은 오버레이 네트워크 위에서 이루어지게 된다. 그림 1은 DHT와 오버레이 네트워크를 보여주는 그림이다. 그림 1에서는 여러 호스트가 오버레이 네트워크에 등록 되고, 하나의 호스트가 키 값이 8인 Data를 오버레이 네트워크에 넣는 과정을 보여 주고 있다.

이 기법은 최대 검색 횟수 O(logN)으로 데이터 검색이 가능하기 때문에, 검색 효율에 상관없이 피어 개수를 임의로 증가 시킬 수 있다. 그렇지만 Unstructured P2P에서는 다양한 데이터의 속성 값을 이용하여 다양한 쿼리가 가능했던 반면에, 분산 해시 테이블을 사용함 으로 인하여 특정 키 값만을 사용한 검색을 함으로써 쿼리가 단순화되는 단점이 있다.

2.2 DHT 연구 동향

2.2.1 홉 수 감소

DHT에 관한 연구가 진행 되면서 홉 수를 줄이기 위한 연구는 계속되고 있고, 그 방식도 다양하게 제안되었다. 여러 개의 Chord ring을 만들어서 검색 시에 요청된 값이 보이면 바로 알려주는 방식을 사용하여 홉 수를 줄이고자 하는 방식이 제안되었고[11], 캐싱(Caching) 등의 방식을 사용하여서 빠른 응답을 할 수 있도록 하는 방식도 많이 제안되었다[12].

2.2.2 부하 분산

네트워크에서는 특정 파일에 대한 요청이 많아지는 현상이 발생하고, 그에 따른 문제점들을 해결하기 위해서 부하 분산과 관련된 연구가 많이 진행되었다. 특히, 많은 파일 전송 어플리케이션에서 사용 될 가능성이 있는 DHT에서는 이런 부분이 큰 과제로 남아 있다. 부하 분산을 위하여 많이 사용하는 방식은 캐싱을 이용한 방식이다. 자신이 담당하고 있는 데이터를 다른 노드에게도 넣어 둠으로써 자신이 아닌 다른 노드가 처리 할 수 있도록 만드는 방식이 캐싱 방식이다. 데이터를 캐싱 하는 방식 말고도, 요청에 대한 부하를 분산하기 위하여서 라우팅 인덱스(Chord에서는 Finger table)를 다른 노드

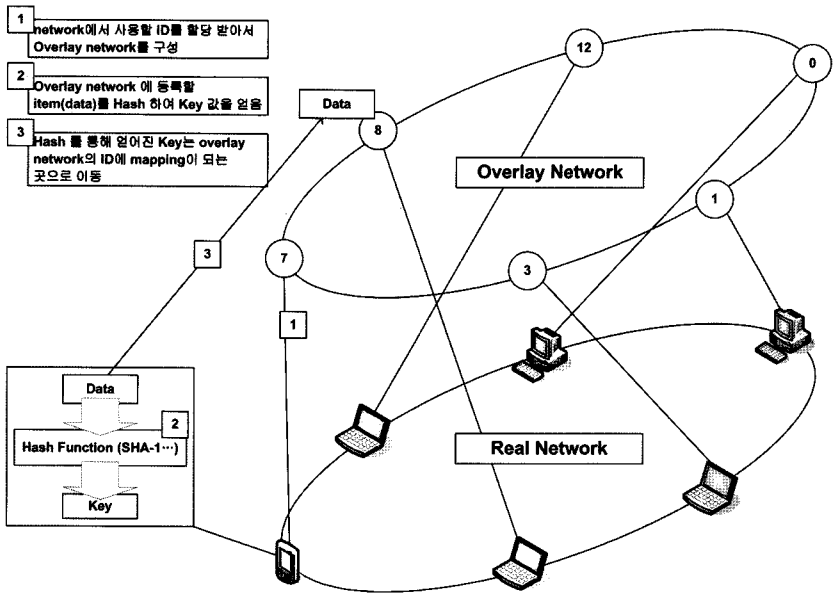


그림 1 오버레이 네트워크와 DHT

에서도 담당하게 함으로써, 요청에 대한 부하를 분산시키는 방식도 제안되었다[13]. 대부분의 부하 분산 기술들이 캐싱 방식으로 이루어지고 있지만, Virtual server를 사용하는 방식도 제안되었다[14]. 이 방식은 노드 하나가 하나의 ID를 가지는 것이 아니라 여러 개의 ID를 가짐으로써, 노드가 담당하는 데이터의 종류를 많게 하는 방식이다.

2.2.3 모바일 환경

전체 네트워크의 환경이 유선에서 무선으로, 정적 노드에서 동적 노드로 바뀌게 됨에 따라 모바일 환경에서의 DHT 역시 많은 연구가 진행되었다. MANET에서 성공률을 높이고 라우팅을 최적화하기 위해, 라우팅 프로토콜인 AODV(Ad-hoc On-demand Distance Vector)와의 연동을 가진 연구[15]가 진행이 되었다. 그리고 모바일 환경에 따른 검색 실패를 줄이기 위해, Backtracking 방식을 사용하는 Chord[16]도 제안되는 등 모바일 및 무선 환경이 되면서 생기는 문제점들을 생각하고 그것을 해결하기 위한 많은 방식들이 연구되었다.

2.2.4 보안

DHT 보안에서는 Sybil attack에 관한 연구가 많이 진행이 되었다. Sybil attack이란 자신이 여러 개의 ID를 가짐으로써 다른 여러 정보들이 자신에게 올 수 있도록 만드는 방식이다. DHT의 경우는 ID를 자신이 만들기 때문에, 자신에게 여러 개의 ID를 할당할 수 있다. 이것을 해결하기 위하여 쿼리를 보낼 때 하나를 보내는 것이 아니라 여러 개를 보내서 정상적인 호스트로부터 응답을 받을 수 있도록 하는 방식이 제안되었다[17].

2.3 기존 Chord

기존의 Chord는 그림 2와 같이 구성이 된다. 그림 2는 n = 5 일 때 Chord 네트워크를 구성한 주소 공간의 예이다. Chord 네트워크에 들어갈 수 있는 노드의 총 개수는 2^5인 32개이다.

Chord에서 각각의 노드들은 Finger table이라는 라우팅 테이블을 가지고 있다. Finger table은 주소공간의 크기에 따라서 테이블의 전체 크기가 변하게 된다. 테이블 크기는 주소 공간의 크기가 n-bit일 경우 n개의 행으로 이루어지게 된다. 그림 2를 보면, 5개의 행으로 이루어진 Finger table을 볼 수 있다.

Chord에서는 데이터 요청 메시지를 보낼 때, Finger table을 보고 메시지를 보내게 된다. 하나의 노드가 메시지를 보내려고 할 때, 가장 먼저 요청 메시지를 해쉬하여 키 값을 얻어 오도록 한다. 그 다음에 자신의 Finger table과 키 값을 비교한다. 자신이 가지고 있는 Finger table에서 그 키 값 보다 작은 값 중에서 가장 큰 값인 Successor로 메시지를 보내도록 한다. 그 메시지를 받은 노드는 앞에서의 Finger table을 확인하는 과정을 반복하여 그 메시지를 지정된 장소로 보내게 된다.

그림 2에서 노드 8번이 19번 키 값을 찾고자 하는 예를 들어 보면, 먼저 노드 8번은 데이터를 해쉬하여 19라는 값을 얻어 오게 된다. 노드 8번은 자신의 Finger table을 보고 19번 보다 작은 값 중의 가장 큰 값인 17번 Successor로 요청 메시지를 보내게 된다. 17번 노드

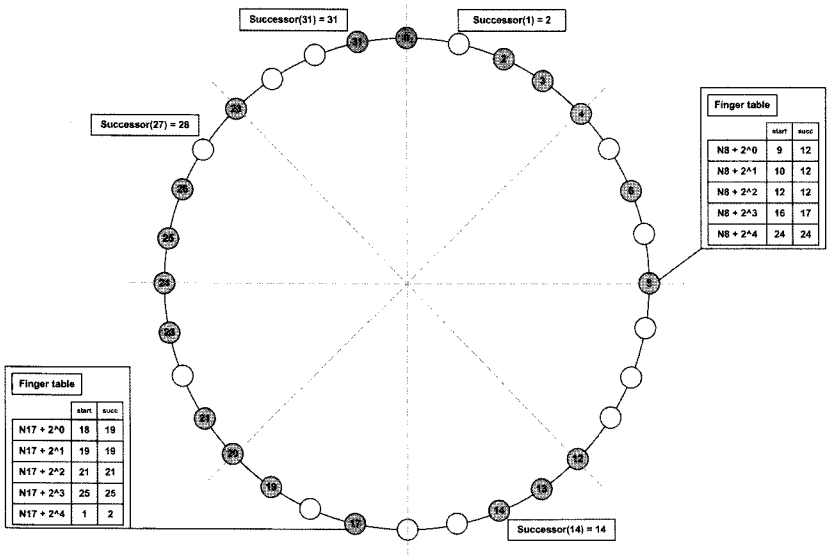


그림 2 Chord 네트워크 (n = 5)

는 그 메시지를 받고 다시 자신의 Finger table을 확인한다. 자신의 Finger table에는 19번 Successor에 관한 정보가 있다. 그 노드는 최종 목적지인 19번 노드로 메시지를 보내게 된다.

Chord에서는 노드의 실패에 대해서 파급효과를 최소화하기 위해서 Stabilization 함수를 주기적으로 호출한다. Stabilization 함수는 크게 두 가지 일을 수행한다. 한 가지는 자신이 알고 있던 Successor가 살아 있는지를 확인하기 위해 Ping 메시지를 주기적으로 보내는 것이다. 만약 Ping 메시지에 대한 응답이 오지 않으면, 그 Successor를 자신의 Finger table에서 지우게 된다. 다른 한 가지는 자신의 Start에 해당하는 Successor를 주기적으로 찾아서 새로 등록을 하는 일이다. 이 메시지는 Ping 메시지를 주기적으로 보내는 것보다 많은 트래픽이 발생한다. 그림 3은 Ping 메시지와 Find_successor 메시지를 보여주기 위한 Stabilization에 관한 그림이다.

Chord에서는 그림 3의 1번처럼 Start 값을 넣어서 보내는 Find_successor 메시지와 2번처럼 Successor가

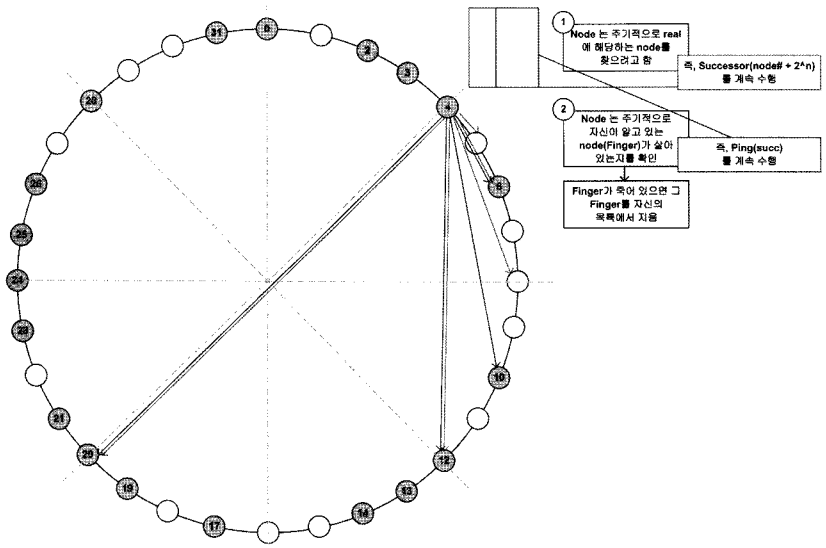


그림 3 Chord에서 Stabilization

살아 있는지 확인을 하기 위한 Ping 메시지가 있다. 후자의 경우는 정해진 노드에게 하나의 메시지만을 보내서 확인을 하기 때문에 문제가 발생하지 않지만, 전자의 경우는 문제가 발생한다. Successor를 찾는 메시지를 보내기 위해서 자신은 하나의 메시지만 발생시키지만 다음 노드로 가게 되면 그 노드 역시 똑같은 과정을 반복하기 때문에, 전체 네트워크에서는 하나가 아닌 여러 개의 메시지가 발생하게 된다.

2.4 접근 방식

2.4.1 기존 Chord의 문제점

모바일 환경처럼 노드의 Join/Leave가 자주 발생하는 환경에서는, 데이터 요청 메시지의 실패가 일어나지 않게 하기 위해서, 자신의 테이블을 최신 정보로 유지해야 한다. 이를 위해서는 Stabilization 주기를 빠르게 하는 방식이 있다. 그렇지만 그 주기가 빨라지게 되면 정해진 시간에 하나의 노드가 처리해야 하는 메시지의 양은 지수적으로 증가하게 된다.

그림 4(a)에서 x 축은 노드의 Join/Leave 정도(초당 빠져나간 노드의 개수)이고, y 축은 그에 따른 메시지 성공률을 보여 주는 그림이다. 1초, 2초, 4초는 Stabilization 주기를 나타낸다. 그림 4(a)에서 노드 Join/Leave가 많은 환경 일수록 요청 메시지의 성공률이 떨어지는 것을 확인 할 수 있다. 그리고 성공률을 높이기 위해서 Stabilization 주기가 빨라져야하는 것을 볼 수 있다. 초당 0.3개의 노드의 Join/Leave가 일어날 경우, Stabilization 주기가 1초 인 경우는 0.7(70%)의 성공률을 보여 주지만, 똑같은 경우에서 Stabilization 주기가 2초 인 경우는 0.59(59%)의 성공률을 보여 준다. 결국 노드의 빈번한 Join/Leave가 발생하는 모바일 환경에서 요청 실패를 줄이기 위해서는 Stabilization 주기가 빨라져야 한다.

Stabilization 주기가 빨라지게 되면 하나의 노드가 초당 처리해야 하는 메시지의 개수는 크게 증가 하게 된다. 그림 4(b)는 Stabilization 주기에 따른 메시지 양의 변화를 보여주는 그래프이다. x 축은 Stabilization 주기(초)를 나타내고, y 축은 하나의 노드가 초당 처리하는 메시지의 개수이다. 그림을 보면 Stabilization 주기가 느려질수록(그래프에서 오른쪽으로 갈수록) 하나의 노드가 처리하는 메시지의 양은 줄어들지만, Stabilization의 주기가 빨라지게 되면 하나의 노드가 처리해야 하는 메시지 양은 크게 증가 하는 것을 볼 수 있다.

2.4.2 본 논문의 접근 방식

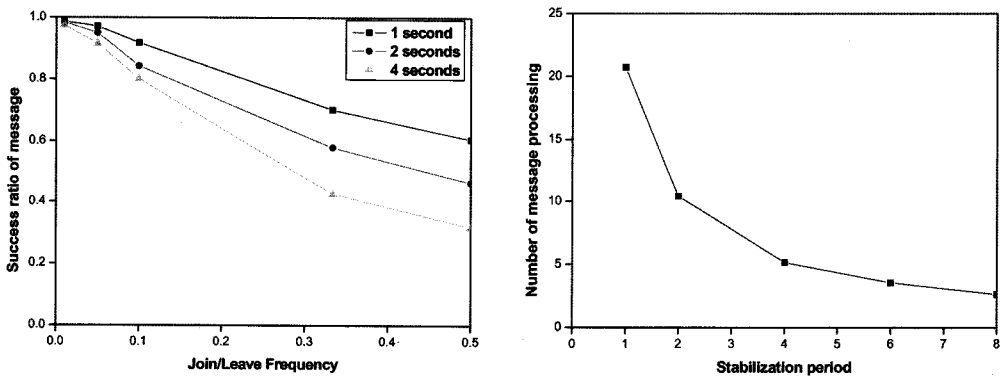
본 논문에서는, 데이터 요청이 왔을 때만 Finger table을 업데이트하는 Reactive한 방식을 제안한다. 이러한 방식을 사용하면 기존 Chord에서 발생하는 문제를 해결할 수 있다.

3. 제안된 방식

3.1 전체 구조

제안된 방식에서는 Stabilization에서 발생하는 Find_successor 메시지를 통해 주기적으로 업데이트하는 방식에서 Finger table을 데이터 요청 시에 업데이트하는 방식으로 바꾸는 것을 제안한다. 그림 5는 제안한 방식의 전체적인 그림이다.

제안된 방식에서는 데이터 요청 메시지가 왔을 경우에만 Find_successor 메시지를 보내서 Finger table을 업데이트한다. 그림 5에서 Request 9가 노드 4번으로 들어왔을 때 제안된 방식은 Find_successor 메시지를 보내게 된다. 이 Find_successor 메시지는 보내져 온 메시지의 Key 값 보다 작은 real 값 (그림 5에서는 8)을 찾는 메시지이고, successor중 8보다 작은 노드인 6으로 보내지게 된다. 그림 5의 1번 과정이 이와 같은 과



(a) (b)

그림 4 Stabilization 주기에 따른 요청 메시지 실패량(a) 및 처리량(b)

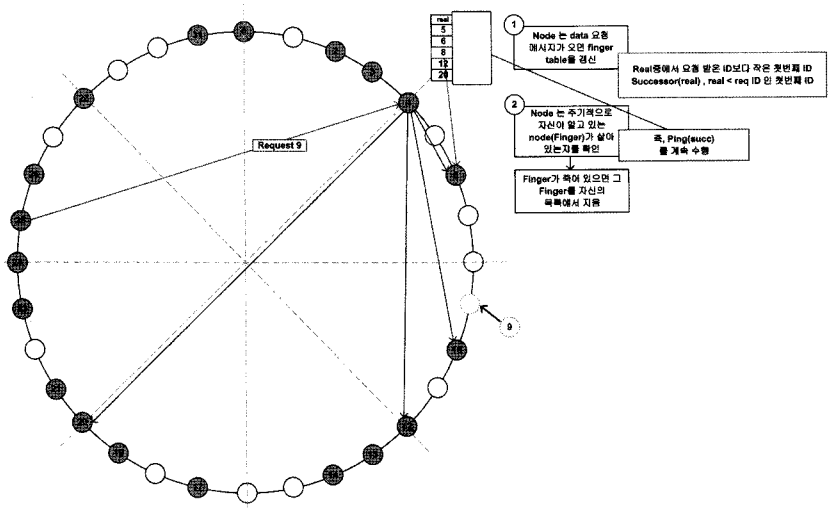


그림 5 제안된 방식에서 Finger table 업데이트

정이다. 이전 Chord에서의 Ping 메시지는 그대로 사용을 한다. 자신이 Find_successor 메시지를 보낼 노드가 없는 상태라면, 메시지를 보내도 응답이 오지 못한다. 그렇기 때문에 노드가 살아있는지를 확인하기 위해서 Ping 메시지는 그대로 사용을 한다. 이 Ping 메시지는 Find_successor 메시지에 비하면 네트워크에서 차지하는 비중은 크지 않기 때문에 크게 문제 되지는 않는다. 이 과정은 기존 Chord에서 Stabilization 주기에 따라서 보내게 되며, 자신의 Finger Table에 있는 모든 Successor에게 보내게 된다. 그림 5의 2번 과정이 이와 같은 과정이다. 그림 6은 Stabilization 주기에 따라서 하나의 노드가 처리하는 Ping 메시지와 Find_successor 메시지의 개수의 차이를 보여 준다. x 축은 Stabilization 주기를 나타내고, y 축은 메시지 양을 나타낸다. 그림 6

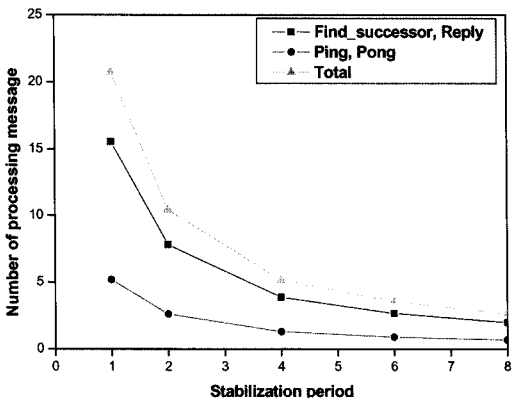


그림 6 Stabilization 주기에 따른 Ping, Find_successor 메시지

을 보면 Find_successor의 양이 Ping 메시지에 비해 메시지 양이 3배 정도 많으며, 하나의 노드가 처리하는 전체 메시지의 대부분이 Find_successor 메시지인 것을 알 수 있다. 예를 들어, Stabilization 주기가 1초 일 때, 하나의 노드가 초당 처리하는 전체 미시지의 개수는 22개이고, Find_successor 메시지와 Find_successor_reply 메시지 개수의 합은 16개, Ping 메시지와 Pong 메시지 개수의 합은 6개이다.

3.2 동작 과정

제안된 방식의 동작 과정은 그림 7과 같다.

- 단계 a. 노드는 주기적으로 자신이 유지하고 있던 Successor에게 Ping 메시지를 보내서 죽었는지 살았는지를 확인한다. 만약 죽었다면, 자신의 테이블에서 지운다. 이 과정은 기존 Chord에서 사용하는 방식을 그대로 사용한 것이다.
- 단계 b. 다른 노드로부터 데이터 요청을 받는다.
- 단계 c. 노드는 받은 데이터 요청 메시지를 Buffer에 넣어둔다. Buffer에는 받은 메시지에 포함된 Hash값(그림 7에서 9)과 그 값이 전달될 곳의 real값(그림 7에서 8)이 저장 된다.
- 단계 d. 노드는 자신의 Start 값 중에 요청 받은 메시지의 키 값보다 작은 값 중 가장 큰 값을 찾는 Find_successor 메시지를 보낸다. 하지만 모든 요청에 대해서 Find_successor를 보내게 되면, 요청이 많을 시에 문제가 생길 수 있다. 그래서 제안하는 방식에서는 이전에 요청했던 것과 똑같은 것을 요청해야 하면 Find_successor 메시지를 보내지 않고 바로 Buffer에 넣는다. 한 번만 보내고 나중에 안 보내면 다시 업데이트할 수 없는 문제가 생기기 때문에 요청을

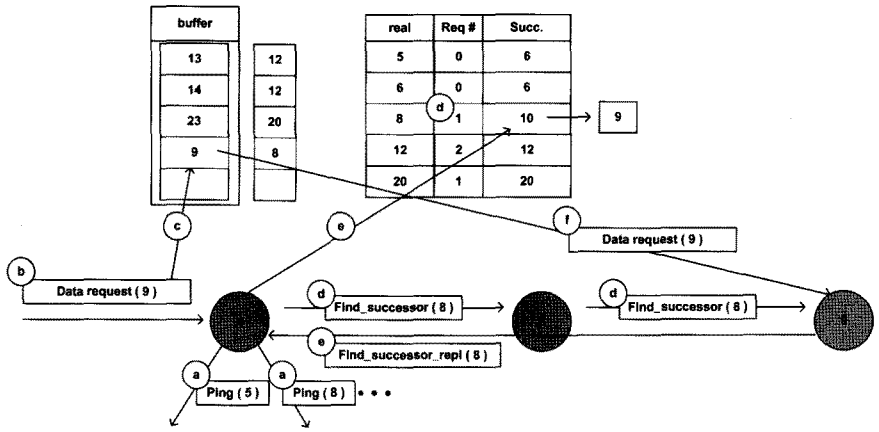


그림 7 제안된 방식의 동작 과정

보내지 않는 수를 제한한다. n개의 숫자를 제한하게 되면 그 요청에 대해서 n번의 요청이 오기 전까지는 Find_successor 메시지를 한 번만 보내고, n번 요청 후에 다시 보내게 된다.

- 단계 e. 단계 d에서 보낸 Find_successor는 나중에 Find_successor_reply를 받는다.
- 단계 f. 노드는 Find_successor_reply를 받았을 때, 자신의 Finger table을 업데이트하고, 단계 c에서 넣어 두었던 데이터 요청 메시지를 다음 노드로 전달하게 된다.
- 단계 g. 전달 받은 노드는 위와 같은 과정을 반복하게 된다.

3.3 정성적 비교

표 1은 기존 방식과 제안된 방식을 Finger table 업데이트 방식, Find_Successor 메시지의 발생 개수 및 모바일 환경 관점에서 정리한 것이다. 제안한 방식은 노드가 요청 메시지를 많이 보내게 되면 그만큼 요청 메시지는 증가한다. 기존 방식은 주기적으로 Find_successor 메시지를 보내서 자신의 Finger table을 업데이트하기 때문에 데이터의 요청 개수에 따라서 메시지가 더 발생하지 않는다. 그렇지만 기존 방식은 노드가 많아지게 되면 전체 네트워크에서 Find_successor 메시지는 증가하게 된다.

4. 실험 및 토론

4.1 실험 환경

실험은 버클리에서 나온 어플리케이션인 I3에서 Chord 부분의 시뮬레이터를 통하여 실험을 수행하였다[18]. 실험에서 사용한 총 메시지의 양은 아래와 같이 계산이 되었다. 총 노드의 수는 1,000대의 노드를 대상으로 실험을 했으며, 실험에서 사용한 변수 및 값은 표 2와 같다.

표 2 실험에 사용된 변수 및 값

변수	값
Stabilization 주기	0.5초, 1초(Default주기), 2초, 4초
메시지 요청 주기	0.1, 0.2, 0.25, 0.45, 0.55, 1.25 (요청 / 초 / 노드)
노드의 Join/Leave 정도	0.01, 0.05, 0.1, 0.3, 0.5 (Join/Leave가 일어난 노드 수 / 초)

총 메시지 개수 = Ping 메시지 개수 + Pong 메시지 개수 + Stabilization 메시지 개수(옆의 노드를 찾는 것) + Stabilization 응답 메시지 개수 + Find_successor 메시지 개수 + Find_successor_ Reply 메시지 개수 + 데이터 요청 메시지 개수

표 2에서 Stabilization 주기가 변화하면 기존 방식에서는 Ping 메시지 주기와 Find_successor 메시지 주기가 변하게 되고, 제안된 방식에서는 Ping 메시지의 주기가 변하게 된다. 이러한 변수들의 값은 다른 논문에서 자주 사용 되는 변수를 기반으로 사용하였다[19,20].

4.2 실험 결과

4.2.1 데이터 요청이 없을 경우

그림 8은 데이터 요청이 없을 경우 하나의 노드에서 발생하는 메시지의 양을 보여 준다. x 축은 Stabilization 주기이고, y 축은 하나의 노드에서 초당 발생하는 메시지의 양이다. 그림과 같이 요청이 없는 상태에서

표 1 기존 방식과 제안된 방식의 차이

	Finger table 업데이트 방식	Find_successor 메시지 발생 수	모바일 환경 (Join/Leave에 따른)
기존 방식	Proactive	노드 수에 비례	비례
제안된 방식	Reactive	요청 메시지의 개수에 비례	비례하지 않음

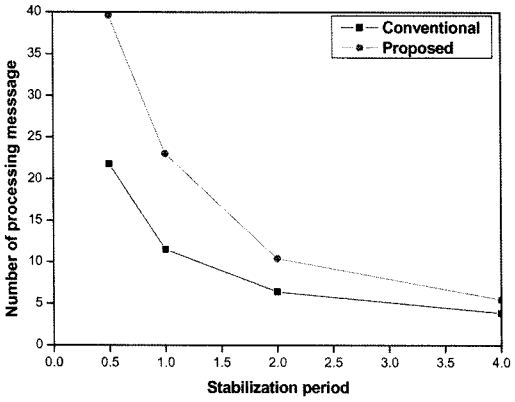


그림 8 데이터 요청이 없을 경우

Finger table을 유지하기 위해서 사용하는 메시지는 기존 방식이 2배 정도 메시지가 많다. 예를 들어, Stabilization의 주기가 1초일 경우 기존 방식은 23개의 메시지를 처리하지만, 제안된 방식은 12개의 메시지만을 처리 한다.

4.2.2 데이터 요청이 있을 경우

그림 9는 하나의 노드가 초당 보내는 메시지의 양에 따라서 처리하는 총 메시지의 개수를 보여준다. 기존의 방식은 요청 메시지가 증가함에 따라 처리하는 메시지 양이 작은 기울기로 증가하는 반면, 제안한 방식은 요청 메시지가 증가함에 따라 처리하는 메시지 양이 기존 방식보다 큰 기울기로 증가하게 된다. 기존 방식에서 default 값인 1을 기준으로 보게 된다면 노드가 초당 0.55개 이상의 메시지 요청을 보낼 경우 기존의 방식보다 많은 트래픽을 발생 하는 문제가 생기지만, P2P 관련 통계 자료들을 본다면 보통 하나의 노드가 초당 데이터를 요청하는 개수는 0.02-0.2개이다[21,22]. 즉, 보통의 평범한 환경에서는 기존 방식 보다 적은 트래픽이 발생하게 되며, 모바일 환경을 가정하여 기존 방식에서 Stabilization 주기를 빠르게 한다면, 더 좋은 결과를 얻을 수 있다. x 축은 초 당 하나의 노드가 보내는 하는 데이터 요청 양을 나타내며, y 축은 초당 하나의 노드가 처리하는 메시지의 총 양이다.

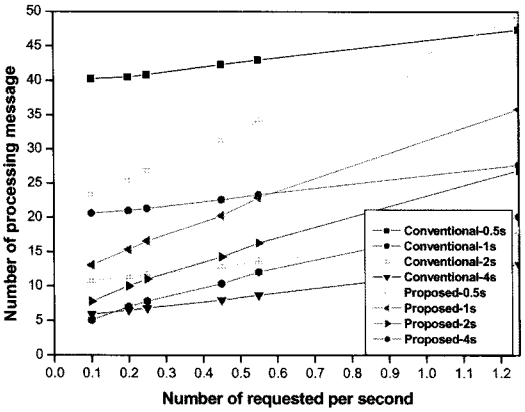


그림 9 하나의 노드가 초당 처리 하는 총 메시지의 개수

4.2.3 정량적 비교

그림 10은 Stabilization 주기가 변함에 따라 제안된 방식이 기존 방식보다 좋아지게 되는 한계값을 보여준다. 기존 방식은 Stabilization 주기가 빨라지게 되면, 그만큼 처리해야 하는 메시지의 개수는 지수적으로 증가하게 된다. 반면에 제안된 방식은 발생하는 메시지가 데이터 요청 개수에 비례하게 된다. 그래프에서 볼 수 있듯이, Stabilization 주기가 증가함에 따라 처리해야 하는 메시지의 개수가 크게 증가하기 때문에, 제안된 방식이 기존 방식보다 좋은 성능을 낼 수 있는 점은 커지게 된다. 즉, Stabilization 주기가 빨라질수록 제안된 방식은 더 좋은 성능을 내게 된다. 예를 들어, Stabilization 주기가 1초일 경우 초당 요청 개수가 0.55개를 넘을 경우 기존방식이 더 좋고, 0.5초일 경우는 초당 요청 개수가 1.15를 넘을 경우 기존 방식이 더 좋게 된다. x 축은 1/Stabilization 주기 이며, y 축은 초당 요청 개수를 나타낸다.

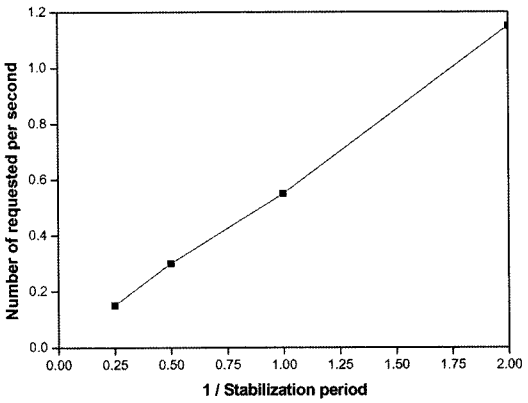


그림 10 Stabilization 주기에 따라 제안된 방식이 기존 방식보다 좋아지는 한계 값

4.3 토론

제안된 방식의 장점은, 모바일 환경의 P2P에서 기존의 방식보다 Finger table을 업데이트하는데 사용하는 메시지의 양을 줄여서 전체 P2P 네트워크의 트래픽을 줄일 수 있는 것이다. 기존 DHT 방식의 P2P에서는 하나의 노드가 처리하는 메시지의 양이 많지 않아서 크게 문제가 되지 않을 것이라는 생각되었지만, 모바일 환경이 되게 되면, 네트워크를 유지하기 위해서 발생시켜야

하는 메시지가 많아질 수 있기 때문이다.

제안된 방식은 요청 메시지가 많아지게 되면, 기존의 방식보다 더 좋지 않은 성능을 낼 가능성이 있다. 그렇지만, 그러한 부분은 데이터 요청 메시지를 받았을 때, 보내는 Find_successor 메시지를 보내는 정도를 줄임으로써, 문제를 해결할 수 있으리라 생각한다.

5. 결 론

본 논문에서는, 기존 Chord에서 Finger table 업데이트를 위한 오버헤드를 줄이기 위하여 Reactive하게 Finger table을 업데이트 하는 방식을 제안하였다. 기존 방식이 주기적인 메시지를 사용하는데 비해서 제안된 방식은 요청에 따라서 메시지가 발생이 되기 때문에, 주기적인 메시지에 대한 오버헤드를 줄일 수 있게 되었다. 제안된 방식은 실험을 통하여 기존의 Chord 방식보다 메시지 처리량이 줄었음을 확인하였다. 하나의 노드가 처리하는 메시지가 줄어들게 되는 것은 네트워크 전체에서 차지하는 트래픽양이 줄어드는 것을 의미한다.

향후 연구 방향으로는, 현재 가지고 있는 데이터 요청에 따라서 증가하는 Find_successor 메시지 양에 대한 단점을 극복하기 위하여, Find_successor를 보내는 양을 상황에 맞춰 변화시켜서 현재 발생 하는 메시지 양을 줄이고, 데이터 요청 메시지가 많아져도 기존의 방식보다 적은 트래픽이 발생하게 하려한다. 그리고 최종적으로 제안된 방식을 사용하여, 모바일 환경에 따른 실패를 더 줄이고, 부하 분산이나 홉 수를 줄일 수 있는 방식에 관한 연구를 진행할 것이다.

참 고 문 헌

[1] eDonkey, "http://www.donkeyp2p.com/".

[2] Gnutella. "http://www.gnutella.com/".

[3] Bittorrent, "http://bitconjurer.org/BitTorrent/".

[4] Napster. "http://www.napster.com/".

[5] I. Clarke et al., Freenet: A Distributed Anonymous Information Storage and Retrieval System, available at http://freenetproject.org/freenet.pdf, 1999.

[6] S. Ratnasamy et al., "A Scalable Content Addressable Network," Proc. ACM SIGCOMM, pp. 161-72, 2001.

[7] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," In IFIP/ACM Int'l Conf. on Distributed Systems Platforms (Middleware), pp. 329 - 350, Nov. 2001.

[8] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz, "Tapestry: A Resilient Global-scale Overlay for Service Deployment," IEEE Journal on Selected Areas in Communications, Vol.22, No.1, pp. 41-43, 2004.

[9] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications," In IEEE/ACM Transactions on Networking, Vol.12, No.2, pp. 205-218, Apr 2004.

[10] H. Park, K. Park, "P2P Technology Trend and Application to Home Network," 전자통신동향분석, Vol.21, No.5, Oct 2006.

[11] P. Flocchini and A. Nayak, "Enhancing Peer-to-Peer Systems Through Redundancy," IEEE Journal on Selected Areas in Communications, Vol.25, No. 1, pp. 15-24, Jan 2007.

[12] S. Serbu, S. Bianchi, P. Kropf, and P. Felber, "Dynamic Load Sharing in Peer-to-Peer Systems: When Some Peers Are More Equal than Others," IEEE Internet Computing, Vol.11, No.4, pp. 53-61, July-Aug. 2007.

[13] H. Cai and J. Wang, "Caching routing indices in structured P2P overlays," International Conference on Parallel Processing, pp. 521-528, June 2005.

[14] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in dynamic structured P2P systems," INFOCOM, pp. 2253-2262, March 2004.

[15] M. Dischinger, "Mobility Enhancements to an Approach for Structured Overlay Routing in Wireless Mobile Ad Hoc Networks," Diploma Thesis, System Architecture Group, University of Karlsruhe, Nov. 2005.

[16] S. Lee and J. Jang, "Backtracking Chord over Mobile Ad-hoc Network," 한국정보과학회 춘계학술대회, pp. 517-519, 2004.

[17] R. Baden, A. Bender, D. Levin, R. Sherwood, N. Spring, and B. Bhattacharjee, "A Secure DHT via the Pigeonhole Principle," Digital Repository at the University of Maryland, Relation UM Computer Science Department, CS-TR-4884, Sep. 2007.

[18] I. Stoicam, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," ACM SIGCOMM Computer Communication Review, Vol.32, pp. 73-86, 2002.

[19] C. Cramer and T. Fuhrmann, "Performance Evaluation of Chord in Mobile Ad Hoc Networks," ACM International Conference on Mobile Computing and Networking, pp. 48-53, 2006.

[20] C. Tang, M. Buco, and R. Chang, "Low traffic overlay networks with large routing tables," ACM Special Interest Group on Measurement and Evaluation, Vol.33, pp. 14-25, 2005.

[21] W. Acosta and S. Chandra, "Trace Driven Analysis of the Long Term Evolution of Gnutella Peer-to-Peer Traffic," Lecture Notes in Computer Science, Vol.4427, pp. 42-51, 2007.

[22] K. Gummadi, R. Dunn, and S. Saroiu, "A Measurement Study of Napster and Gnutella as

Examples of Peer-to-Peer File Sharing Systems," Symposium on Operating Systems Principles, Oct. 2003.

윤 영 효

2006년 숭실대학교 정보통신전자공학부 졸업(학사). 2006년 3월~현재 숭실대학교 정보통신전자공학부 대학원 석사과정 관심분야는 네트워크 컴퓨팅 및 보안

곽 후 근

1996년 호서대학교 전자공학과 졸업(학사). 1998년 숭실대학교 전자공학과 대학원 졸업(석사). 1998년~2006 숭실대학교 전자공학과 대학원 졸업(박사). 1998년 8월~2000년 7월 (주) 3R 부설 연구소 주임 연구원. 2006년 3월~현재 숭실대학교 정보통신전자공학과 대학원, postdoc. 관심분야는 네트워크 컴퓨팅 및 보안

김 정 길

2003년 8월 연세대학교 컴퓨터학과 석사. 2006년 8월 연세대학교 컴퓨터학과 공학박사. 2006년 9월~2007년 8월 연세대학교 컴퓨터학과 박사후 연구원. 2007년 9월~2008년 2월 연세대학교 컴퓨터학과 연구교수. 2008년 3월~현재 남서울대학교 컴퓨터학과, 전임강사. 연구분야는 멀티미디어 임베디드 시스템, 컴퓨터구조

정 규 식

1979년 서울대학교 전자공학과(공학사) 1981년 한국과학기술원 전산학과(이학석사). 1986년 미국 University of Southern California(컴퓨터공학석사). 1990년 미국 University of Southern California(컴퓨터공학박사). 1998년 2월~1999년 2월 미국 IBM Almaden 연구소 방문 연구원. 1990년 9월~현재 숭실대학교 정보통신전자공학부, 교수. 관심분야는 네트워크 컴퓨팅 및 보안