DOI: 10.3745/KIPSTC.2009.16C.6.761

모바일 P2P 환경에서 네트워크 트래픽을 최소화한 적응적인 Chord

윤 영 효[†] · 곽 후 근[††] · 김 정 길[†††] · 정 규 식[††††]

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, 네트워크의 트래픽은 증가하게 된다. 본 연구자들은 기존 연구에서 리액티브 라우팅 테이블 업데이트 방식을 이용하여 기존 Chord에서의 라우팅 테이블 업데이트에 따른 부하를 줄이는 기법을 제안하였으나, 초당 요청 메시지 개수가 많아지게 되면 기존의 방식보다 트래픽 양이 많아지게 되는 단점을 가진다.

이에 본 논문에서는 전체 네트워크의 트래픽을 줄이기 위한 적응적인 라우팅 테이블 업데이트 방식을 제안한다. 본 연구자들은 제안된 방법에서 초당 요청 메시지의 개수에 따라 라우팅 테이블 업데이트 방식을 바꾸는 것을 제안하였다. 적응적인 Chord는 초당 요청 메시지의 개수가 어느 임계값보다 작아지면 리액티브 Chord를 사용하고, 그 반대의 경우에는 기존의 Chord를 사용하는 방식이다. 실험은 버클리 대학에서 만들어진 Chord 시뮬레이터(I3)를 이용하여 수행하였고, 실험을 통하여 제안된 방식이 기존 방식에 비해 성능이 향상되었음을 확인하였다.

키워드 : 분산 해쉬 테이블, 라우팅 테이블, 리액티브 코드, 적응적인 코드

An Adaptive Chord for Minimizing Network Traffic in a Mobile P2P Environment

Younghyo Yoon[†] · Hukeun Kwak[††] · Cheongghil Kim[†††] · Kyusik Chung[††††]

ABSTRACT

A DHT(Distributed Hash Table) based P2P is a method to overcome disadvantages of the existing unstructured P2P method. If a DHT algorithm is used, it can do a fast data search and maintain search efficiency independent of the number of peer. The peers in the DHT method send messages periodically to keep the routing table updated. In a mobile environment, the peers in the DHT method should send messages more frequently to keep the routing table updated and reduce the failure of a request. Therefore, this results in increase of network traffic. In our previous research, we proposed a method to reduce the update load of the routing table in the existing Chord by updating it in a reactive way, but the reactive method had a disadvantage to generate more traffic than the existing Chord if the number of requests per second becomes large.

In this paper, we propose an adaptive method of routing table update to reduce the network traffic. In the proposed method, we apply different routing table update method according to the number of request message per second. If the number of request message per second is smaller than some threshold, we apply the reactive method. Otherwsie, we apply the existing Chord method. We perform experiments using Chord simulator (I3) made by UC Berkeley. The experimental results show the performance improvement of the proposed method compared to the existing methods.

Keywords : Distributed Hash Table, Routing Table, Reactive Chord, Adaptive Chord

※ 본 연구는 숭실대학교 교내 연구비 지원으로 이루어졌음.
† 준 회 원 : 숭실대학교 정보통신전자공학부 석사과정
†† 정 회 원 : 숭실대학교 정보통신전자공학과 postdoc(교신저자)
††† 종신회원 : 남서울대학교 컴퓨터학과 전임강사
†††† 정 회 원 : 숭실대학교 정보통신전자공학부 교수
논문접수 : 2009년 1월 19일
수 정 일 : 1차 2009년 7월 10일
심사완료 : 2009년 7월 10일

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, Gnutella, Bit-torrent 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, Gnutella, Freenet 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[1], Pastry[2], Tapestry[3], Chord[4] 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[4]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.

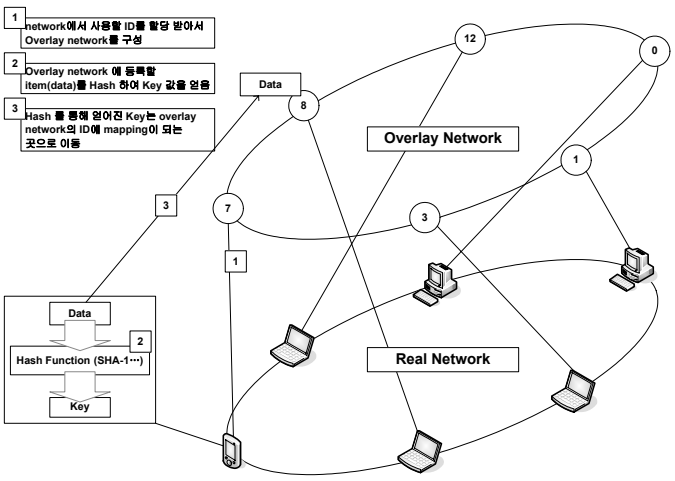
본 연구자는 이러한 Chord의 네트워크 트래픽 문제를 해결하기 위해여 리액티브 Chord 방식을 초기에 제안을 하였다. 그렇지만 이 방식은 기존의 문제점을 해결하였지만 다른 문제점이 발생하게 되었다. 본 논문에서는 기존의 Chord 및 본 연구자가 이전에 제안한 리액티브 Chord[5-6]가 가지는 문제점을 분석하고, 이를 해결하기 위해 적응적인 라우팅 테이블 업데이트 기법을 제안한다. 본 논문의 구성은 다음과 같다. 제 2장에서는 DHT 및 기존 Chord에 대한 연구 방향과 기존 Chord 및 리액티브 Chord의 문제점을 소개한다. 3장에서는 본 논문에서 제안하는 적응적인 Chord를 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 1)은 DHT와 오버레이 네트워크를 보여주는 그림이다. (그림 1)에서는 여러 호스트가 오버레이 네트워크에 등록 되고, 하나의 호스트가 키 값이 8인 Data를 오버레이 네트워크에 넣는 과정을 보여주고 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.



(그림 1) 오버레이 네트워크와 DHT

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[7], 캐싱(Caching) 등의 방식을 사용하여서 빠른 응답을 할 수 있도록 하는 방식도 많이 제안되었다[8].

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 노드에서도 담당하게 함으로써, 요청에 대한 부하를 분산시키는 방식도 제안되었다[9]. 대부분의 부하 분산 기술들이 캐싱 방식으로 이루어지고 있지만, Virtual server를 사용하는 방식도 제안되었다[10]. 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[11]가 진행이 되었다. 그리고 모바일 환경에 따른 검색 실패를 줄이기 위해, Backtracking 방식을 사용하는 Chord [12]도 제안되는 등 모바일 및 무선 환경이 되면서 생기는 문제점들을 생각하고 그것을 해결하기 위한 많은 방식들이 연구되었다.

2.2.4 보안

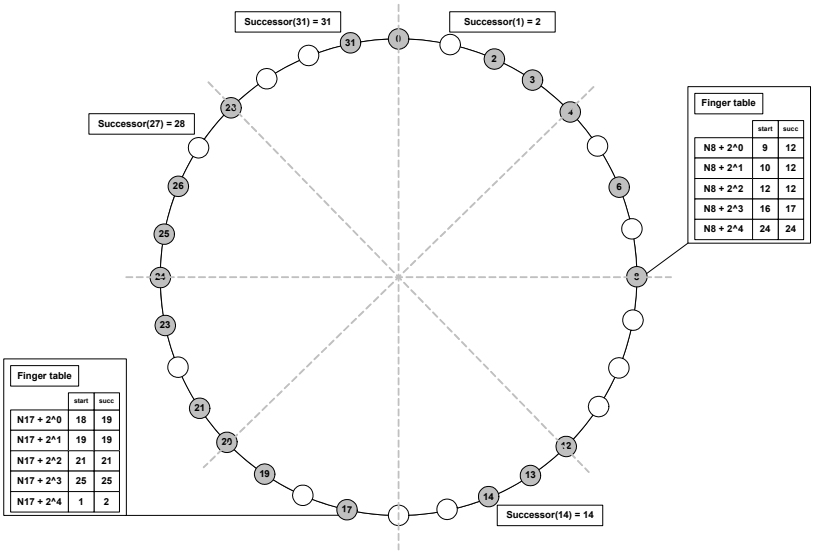
DHT 보안에서는 Sybil attack에 관한 연구가 많이 진행이 되었다. Sybil attack이란 자신이 여러 개의 ID를 가짐으로써 다른 여러 정보들이 자신에게 올 수 있도록 만드는 방식이다. DHT의 경우는 ID를 자신이 만들기 때문에, 자신에게 여러 개의 ID를 할당 할 수 있다. 이것을 해결하기 위하여 쿼리를 보낼 때 하나를 보내는 것이 아니라 여러 개를 보내서 정상적인 호스트로부터 응답을 받을 수 있도록 하는 방식이 제안되었다[13].

2.3 기존 Chord

2.3.1 전체 구조

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

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 2)를 보면, 5개의 행으로 이루어진 Finger table



(그림 2) Chord 네트워크 (n=5)

을 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 2)에서 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 노드는 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.

2.3.2 Stabilization 함수

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 해당 하는 Successor를 주기적으로 찾아서 새로 등록을 하는 일이다. 이 메시지는 Ping 메시지를 주기적으로 보내는 것보다 많은 트래픽이 발생한다. (그림 3)은 Ping 메시지와 Find_successor 메시지를 보여주기 위한 Stabilization에 관한 그림이다.

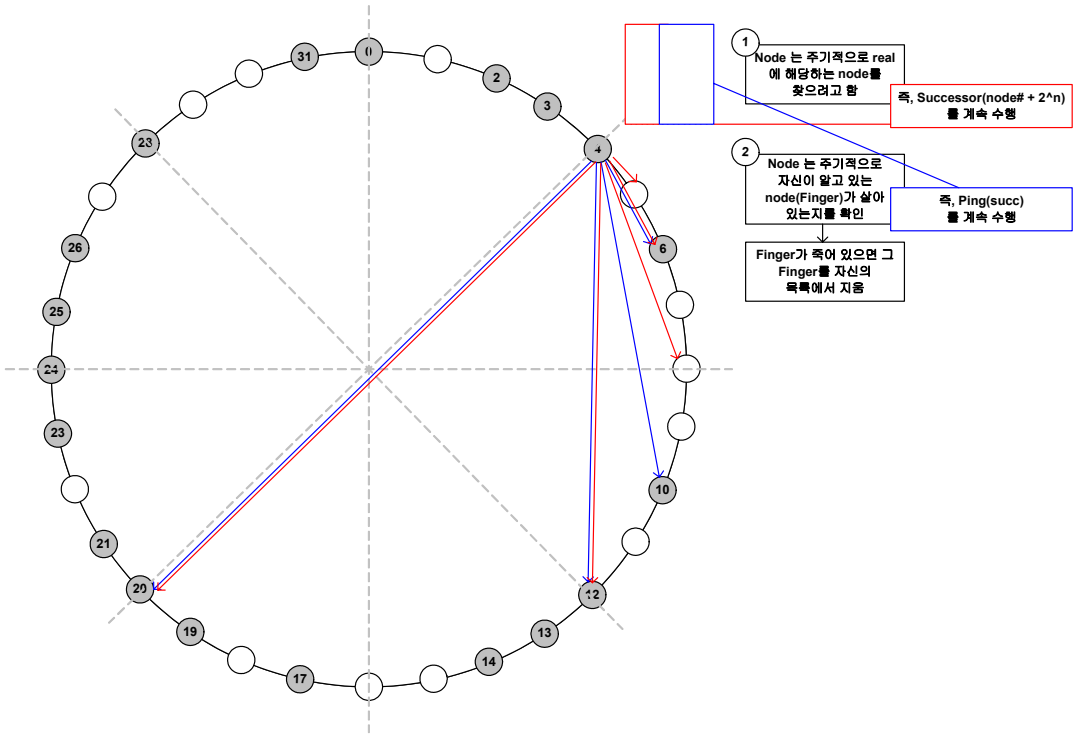
Chord에서는 (그림 3)의 1번처럼 Start 값을 넣어서 보내는 Find_successor 메시지와 2번처럼 Successor가 살아 있는지 확인을 하기 위한 Ping 메시지가 있다. 후자의 경우는 정해진 노드에게 하나의 메시지만을 보내서 확인을 하기 때문에 문제가 발생하지 않지만, 전자의 경우는 문제가 발생한다. Successor를 찾는 메시지를 보내기 위해서 자신은 하나의 메시지만 발생시키지만 다음 노드로 가게 되면 그 노드 역시 똑같은 과정을 반복하기 때문에, 전체 네트워크에서는 하나가 아닌 여러 개의 메시지가 발생하게 된다.

2.4 리액티브 Chord

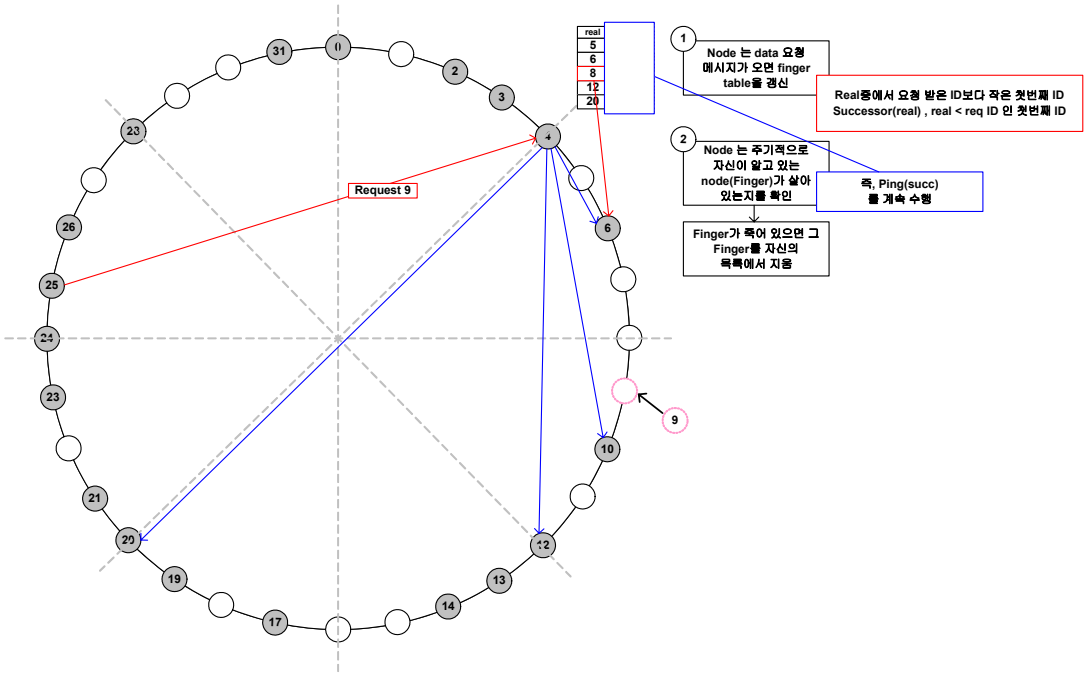
2.4.1 전체 구조

리액티브 Chord에서는 Stabilization에서 발생하는 Find_successor 메시지를 통해 주기적으로 업데이트하는 방식에서 Finger table을 데이터 요청 시에 업데이트하는 방식으로 바꾸는 것을 제안한다. (그림 4)는 리액티브 Chord의 전체적인 그림이다.

리액티브 Chord에서는 데이터 요청 메시지가 왔을 경우에만 Find_successor 메시지를 보내서 Finger table을 업데이트한다. 이전 Chord에서의 Ping 메시지는 그대로 사용을 한다. 자신이 Find_successor 메시지를 보낼 노드가 없는 상태라면, 메시지를 보내도 응답이 오지 못한다. 그렇기 때문



(그림 3) Chord에서 Stabilization



(그림 4) 리액티브 Chord에서 Finger table 업데이트

에 노드가 살아있는지를 확인하기 위해서 Ping 메시지는 그대로 사용을 한다.

2.4.2 동작 과정

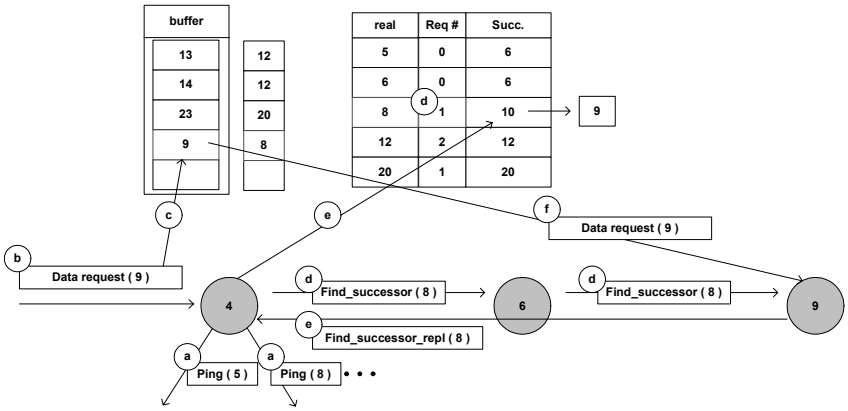
리액티브 Chord의 동작 과정은 (그림 5)과 같다.

평소에 노드는 주기적으로 자신이 유지하고 있던 Successor에게 Ping 메시지를 보내서 죽었는지 살았는지를 확인한다. 만약 죽었다면, 자신의 테이블에서 지운다. 하나의 노드가 다른 노드로부터 데이터 요청을 받았을 때, 노드는 받은 데이터 요청 메시지를 Buffer에 넣어둔다. 이 때, 노드는 자신의 Real 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 리액티브 Chord에서는 이전에 요청했던 것과 똑같은 것을 요청해야 하면 Find_successor 메시지를 보내지 않고 바로 Buffer에 넣는다. 한번만 보내고 나중에 안 보내면 다시 업데이트할 수 없는 문제가 생기기 때문에 요청을 보내지 않는 수를 제한한다. n개의 숫자를 제한하게 되면 그 요청에 대해서 n번의 요청이 오기 전 까지는 Find_successor 메시지를 한번만 보내고, n번 요청 후에 다시 보내게 된다. 위에서 보낸 Find_successor는 나중에 Find_successor_reply를 받는다. 이 메시지를 받았을 때, 자신의 Finger table을 업데이트하고, 버퍼에 넣어 두었던 데이터 요청 메시지를 다음 노드로 전달하게 된다. 전달 받은 노드는 위와 같은 과정을 반복 하게 된다.

2.5 접근 방식

2.5.1 기존 Chord의 문제점

모바일 환경처럼 노드의 Join/Leave가 자주 발생하는 환

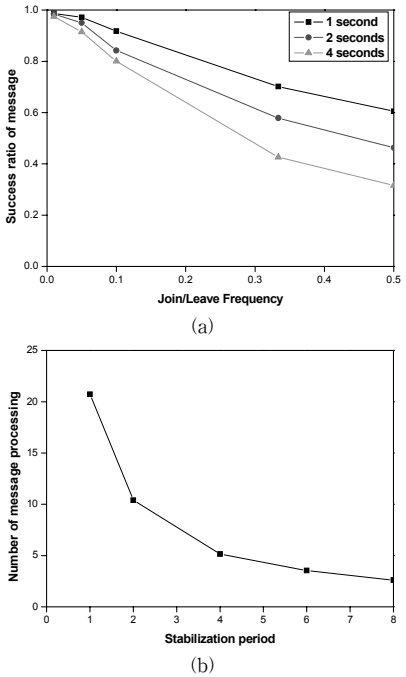


(그림 5) 리액티브 Chord의 동작 과정

경에서는, 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 6)(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 6)(a)에서 노드 Join/Leave가 많은 환경일수록 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 6)(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.



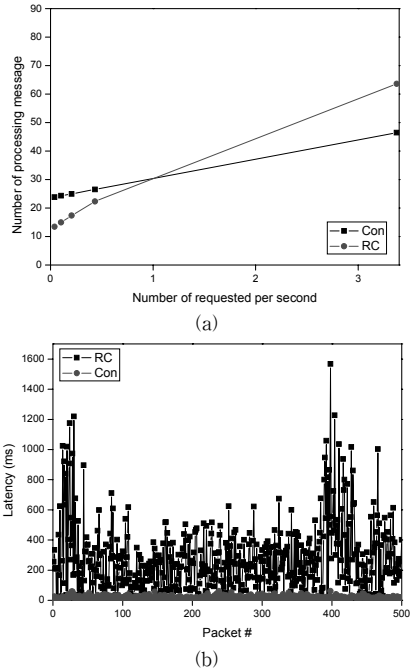
(그림 6) Stabilization 주기에 따른 요청 메시지 실패량(a) 및 처리량(b)

2.5.2 리액티브 Chord의 문제점

기존 Chord의 문제를 해결하기 위해 네트워크 트래픽을 줄이기 위한 리액티브 Chord가 제안이 되었지만, 그 역시 아래와 같은 두 가지 문제점을 가진다. 하나는 요청 메시지가 많아질 경우 트래픽의 문제와 다른 하나는 요청 응답 시간의 문제이다.

(그림 7)(a)는 노드 수가 1,000개 일 경우 초당 요청 메시지 개수에 따라 처리하는 메시지 양을 나타내는 그림이다. x축은 초당 하나의 노드가 보내는 요청 메시지 개수를 나타내고, y축은 노드 하나가 초당 처리한 메시지 개수를 나타낸다. 그림처럼 노드 하나가 초당 요청한 메시지의 개수가 0.9개를 넘어가게 되면, 기존의 Chord보다 노드 하나가 처리하는 메시지 개수가 많아지게 된다.

(그림 7)(b)는 1,000개의 노드가 있는 Chord 네트워크에서 500개의 메시지 요청을 하였을 때, 각 메시지의 응답 시간을 나타내는 그림이다. 그림에서 Con은 기존 코드를 나타내고, RC는 리액티브 Chord를 나타낸다. x 축은 요청 메시지의 번호를 나타내고, y 축은 각 요청 메시지의 응답 시간을 나타낸다. 그림에서 보여 주듯이 리액티브 Chord는 원래의 Chord 보다 요청 메시지의 응답 시간과 표준 편차가 굉장히 크다. 수치적으로 기존 Chord는 평균 18.43ms 정도가 걸리고 표준 편차는 11.66ms인 반면에 Reactive Chord는 평균이 289.41ms이고 표준 편차는 213.92ms이다. 이렇게 큰 요청에 대한 응답 지연 시간은 네트워크를 이용하는데 큰 불편함을 가져오게 된다.



(그림 7) 리액티브 Chord에서 트래픽 문제(a)와 요청 응답 시간 문제(b)

2.5.3 본 연구의 접근 방식

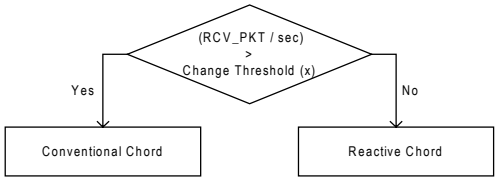
리액티브 Chord 방식의 장점은, 모바일 환경의 P2P에서 기존의 방식보다 Finger table을 업데이트하는데 사용하는 메시지의 양을 줄여서 전체 P2P 네트워크의 트래픽을 줄일 수 있는 것이다. 기존 유선 방식의 P2P에서는 하나의 노드가 처리하는 메시지의 양이 많지 않아서 크게 문제가 되지 않을 것이라는 생각되었지만, 모바일 환경이 되게 되면, 네트워크를 유지하기 위해서 발생시켜야 하는 메시지가 많아질 수 있기 때문이다. 그렇지만 리액티브 Chord는 데이터 요청이 많아지게 되면, 네트워크의 트래픽이 기존의 Chord에 비해 커진다는 단점이 있다. 본 논문에서는 그러한 리액티브 Chord의 한계적인 문제점을 해결하기 위하여 기존의 Chord 방식과 리액티브 Chord 방식이 상황(데이터 요청)에 맞춰서 변할 수 있도록 하는 적응적인 Chord 방식을 제안한다.

3. 적응적인 Chord

3.1 전체 구조

적응적인 Chord는 기존의 Chord 방식과 리액티브 Chord의 알고리즘이 상황(초당 들어오는 메시지의 개수) 변하는 방식(상황에 따라 네트워크 트래픽을 적게 사용하는 방식을 사용)을 제안한다. (그림 8)은 적응적인 Chord의 전체적인 그림이다.

적응적인 Chord는 초당 요청메시지의 개수가 x 값 보다 커지게 되면 기존의 Chord 알고리즘을 따르게 되고, 초당 요청메시지의 개수가 x 값 보다 작아지게 되면 리액티브 Chord 알고리즘을 따르게 되는 방식을 제안한다.



(그림 8) 적응적인 Chord의 전체적인 그림

3.2 임계값 선택 (Changing Threshold)

적응적인 Chord는 현재 상황에 맞춰 알고리즘이 변해야 하는 방식이다. 기존 방식의 Chord와 리액티브 Chord는 초당 들어오는 요청 메시지의 개수에 따라서 그 성능 차이를 보이게 된다. 그렇기 때문에, 본 논문에서는 얼마만큼의 초당 요청이 들어오게 되면, 기존의 방식에 비해서 성능이 나빠지는지를 알아보고, 그 임계점을 기준으로해서 기존의 Chord와 리액티브 Chord의 알고리즘을 바꾸도록 한다. Chord에서 발생하는 트래픽양은 아래와 같이 계산될 수 있다.

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

3.2.1 기존 Chord의 트래픽 발생량

기존 Chord의 트래픽은 크게 주기적으로 발생하는 메시지와 데이터 요청 메시지 부분으로 나눌 수 있다.

기존 Chord의 트래픽양
= 주기적인 메시지 + 데이터 요청 메시지(x)

$$\text{주기적인 메시지} = \frac{ping + pong + stab + stab\ reply + fs + fs\ reply}{stabilization\ period}$$

Ping 메시지, Stabilization 메시지 그리고 Fs 메시지는 Stabilization 주기마다 하나씩 발생 하게 된다. 그리고 pong 메시지, Stabilization reply 그리고 Find_successor reply 메시지는 그에 따른 응답으로 발생하게 된다. Ping 메시지 및 Stabilization 메시지는 노드 하나를 대상으로 하고 메시지를 받은 노드는 그에 대해서 응답을 하는 방식이기 때문에 Stabilization 주기마다 하나의 메시지만 발생하게 된다. 그렇지만 Fs 메시지는 대상 노드로 전달이 된 후에 전달 받은 노드가 파악하여 그 메시지를 전달하거나 응답을 하거나를 선택하게 된다. 즉, Fs 메시지는 Fs메시지의 목적지 까지 Fs 메시지를 계속 발생 시키게 되는 것이다. 이런 Fs 메시지의 목적지는 그 응답을 할 수 있는 노드가 된다. 하나의 Fs메시지는 Fs 메시지의 평균 홉 수인 (log2N - 1)개의 메시지를 발생 시키는 것과 마찬가지 이다. 여기서 1을 뺀 이유는 목적지(D)로 가기 이전의 노드(preD)는 최종 목적지 노드(D)에 대해서 알고 있기 때문에, 목적지 노드(D)가 아닌 그 이전의 노드(preD)가 답을 할 수 있기 때문이다.

$$\text{기존 } Chord\text{의 트래픽양} = \frac{ping + pong + stab + stab\ reply + \log_2 N - 1 + 1}{stabilization\ period} + data$$

$$= \frac{ping + pong + stab + stab\ reply + \log_2 N}{stabilization\ period} + data$$

3.2.2 리액티브 Chord의 트래픽 발생량

리액티브 Chord는 Ping 및 Stabilization 메시지는 기존의 Chord 방식과 같이 사용하기 때문에, Stabilization 따라서 변하게 된다. 그렇지만 Fs 메시지는 요청 메시지의 개수에 따라서 변하기 때문에 메시지의 발생양은 아래와 같다.

리액티브 Chord의 트래픽양
= 주기적인 메시지 + 데이터 요청 메시지(x)에 따른 Fs 메시지 및 Fs reply 메시지 + 데이터 요청 메시지(x)

데이터 요청 메시지 역시 Chord 알고리즘에 따라서 라우팅 되기 때문에 하나의 요청 메시지는 log2N 의 홉 수를 가지게 된다. 데이터 요청 메시지에 따른 Fs 메시지 및 Fs reply 메시지는 아래와 같다. 리액티브 Chord에서 요청 메시지마다 라우팅 테이블을 업데이트 하게 되면 트래픽이 크게 커지기 때문에, 10개의 요청메시지 마다 라우팅 테이블을 업데이트하기 위한 메시지(Fs)를 보내도록 하였다. 그렇기 때문에 전체 메시지양은 1/10 으로 줄어든다. 또한 향상된 방식[12]에서 10초의 타임아웃이 되기 이전에 요청 메시지가

들어 왔을 경우에는 테이블을 업데이트 하지 않기 때문에 리액티브 Chord의 메시지양은 아래와 같다.

$$\text{리액티브 } Chord\text{의 } fs \text{ 및 } fs\ reply\text{메시지양} = \frac{x}{10\times 10}\sum_{y=0}^{\log_2 N-1}(\log_2 N-1+1-y)$$

$$= \frac{x}{10\times 10}\sum_{y=0}^{\log_2 N-1}(\log_2 N-y)$$

3.2.3 임계값 선택 (Changing Threshold)

"기존 Chord의 트래픽 양 = 리액티브 Chord의 트래픽"이 되는 x 값 이후로 x 값이 커지게 되면, 리액티브 Chord는 기존의 Chord에 비해서 트래픽양이 많아지게 되는 문제점이 생긴다. 따라서 본 논문에서는 평소에는 리액티브 방식으로 동작을 하다가 초당 요청 개수가 x값을 넘어가게 되면 기존의 방식으로 바뀔 수 있도록 하는 방식을 제안한다.

$$\text{기존 } Chord\text{의 트래픽양} \;=\; \text{리액티브 } Chord\text{의 트래픽양}$$

$$\frac{ping+pong+stab+stab\ reply}{stabilization\ period}+\frac{fs+fs\ reply}{stabilization\ period}+data$$

$$=\frac{ping+pong+stab+stab\ reply}{stabilization\ period}+\frac{x}{100}\sum_{y=0}^{\log_2 N-1}(\log_2 N-y)+data$$

$$\frac{\log_2 N}{stabilization\ period}=\frac{x}{100}\sum_{y=0}^{\log_2 N-1}(\log_2 N-y)$$

$$\therefore\ x=\frac{100\log_2 N}{stabilization\ period\sum_{y=0}^{\log_2 N-1}(\log_2 N-y)}$$

이 경우 Stabilization 주기가 1초 이고 네트워크의 총 노드수가 1000개 일 경우, x 값은 18개 정도가 된다. Stabilization 주기가 1초 이고 노드수가 2000개 일 경우는 x 값은 16.5개 정도가 된다.

x 값은 Stabilization 주기와 네트워크 전체의 노드 수에 의존하고 있다. 구현 관점에서 Stabilization 주기는 프로그램상의 값을 이용하면 된다. N 값은 네트워크 전체의 노드 수이기 때문에 직접적으로 하나의 노드가 알아내기는 힘들지만 자신이 가지고 있는 Finger Table을 기반으로 찾아낼 수 있다. 자신이 알고 있는 Finger Table의 노드의 수는 log2N 개 정도의 노드를 알고 있게 된다. 즉, 수식에서의 log2N 의 값은 Finger Table에서 자신이 알고 있는 노드의 개수와 거의 같다고 생각 할 수 있다. 위의 수식은 구현 관점에서 Stabilization 주기와 Finger Table에 있는 노드의 개수를 기반으로 구현이 될 수 있다.

4. 실험 및 토론

4.1 실험 환경

실험은 버클리에서 나온 어플리케이션인 I3에서 Chord 부분의 시뮬레이터를 통하여 실험을 수행하였다[14]. 실험에서 사용한 총 메시지의 양은 아래와 같이 계산이 되었다. 실험에서 사용한 변수 및 값은 <표 1>와 같다.

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

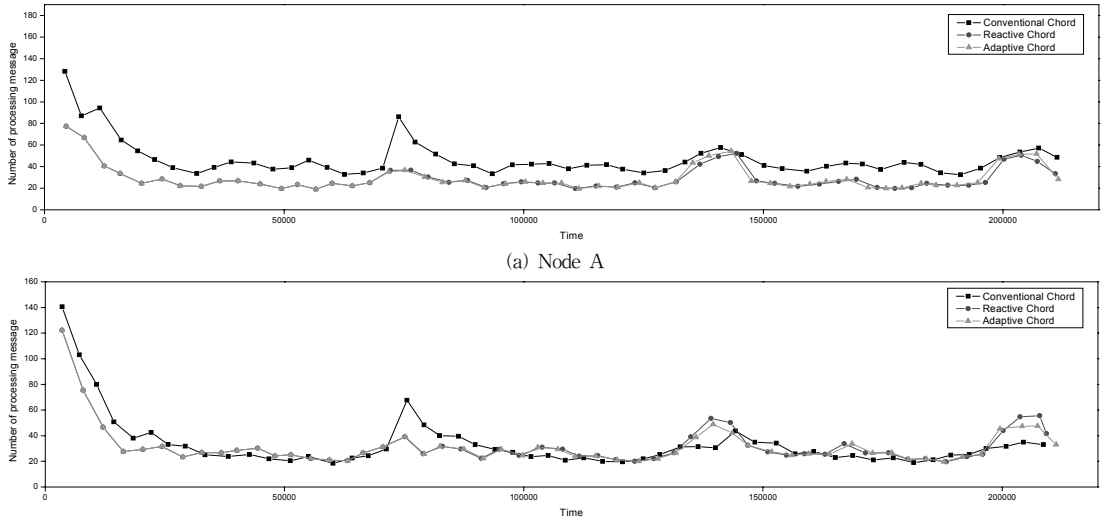
〈표 1〉 실험에 사용된 변수 및 값

변수	값
노드 수	1,000노드, 2,000노드, 4,000노드
Stabilization 주기	0.5초, 1초(Default주기)
메시지 요청 주기	Random 요청

4.2 실험 결과

4.2.1 정량적 비교

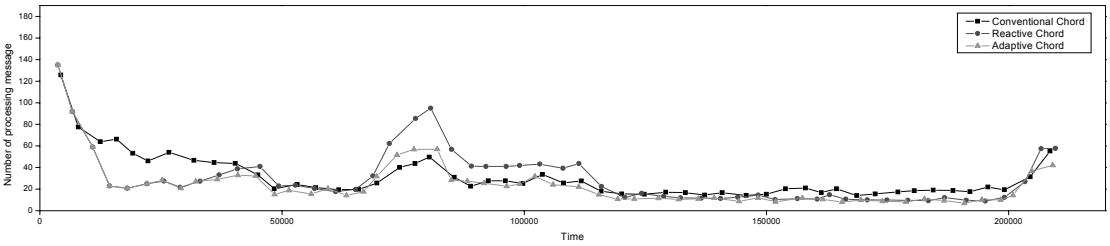
(그림 9)-(a),(b)는 총 노드수가 1000개이고 Stabilization



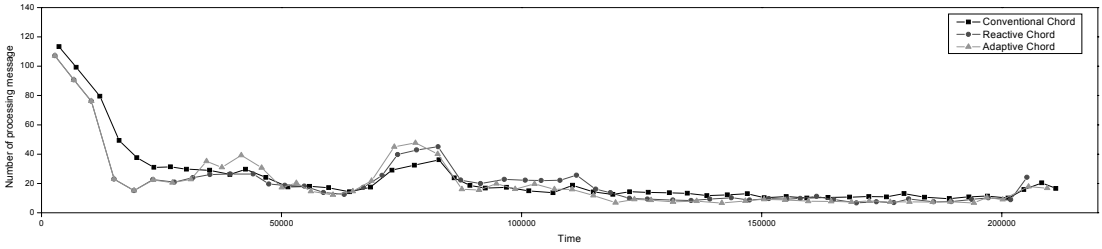
(a) Node A

(b) Node B

(그림 9) 시간에 따른 처리 메시지양 (1000node, 0.5sec)

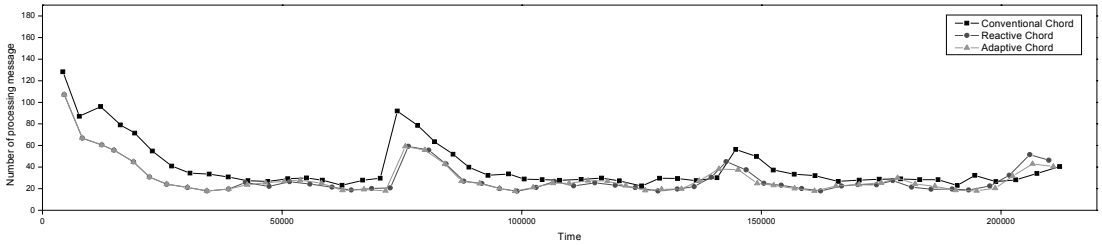


(a) Node A

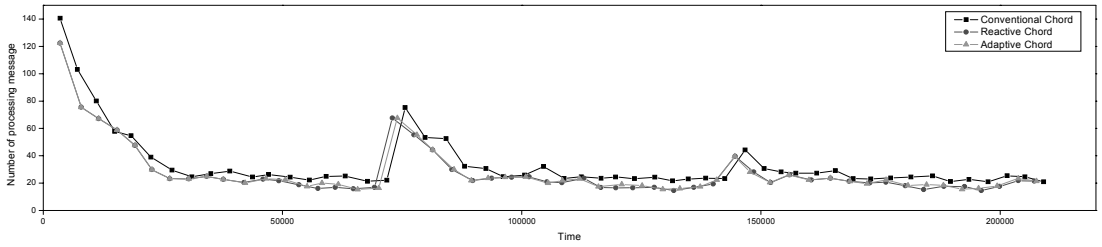


(b) Node B

(그림 10) 시간에 따른 처리 메시지양 (1000node, 1sec)

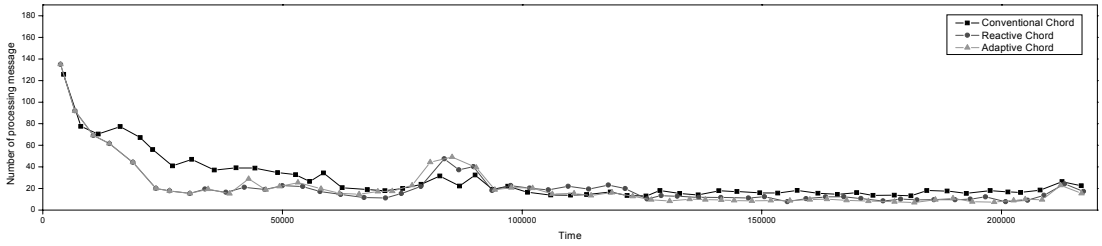


(a) Node A

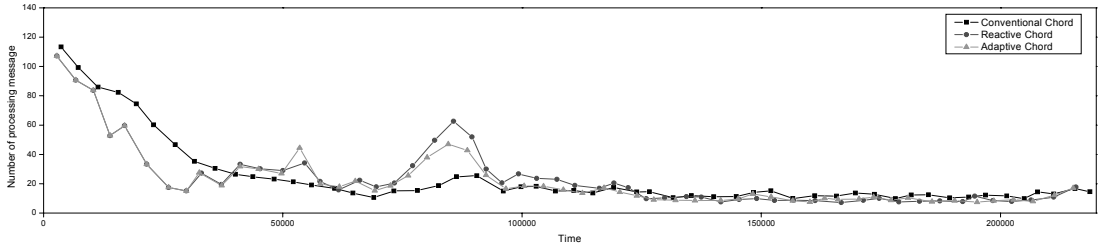


(b) Node B

(그림 11) 시간에 따른 처리 메시지양 (2000node, 0.5sec)



(a) Node A



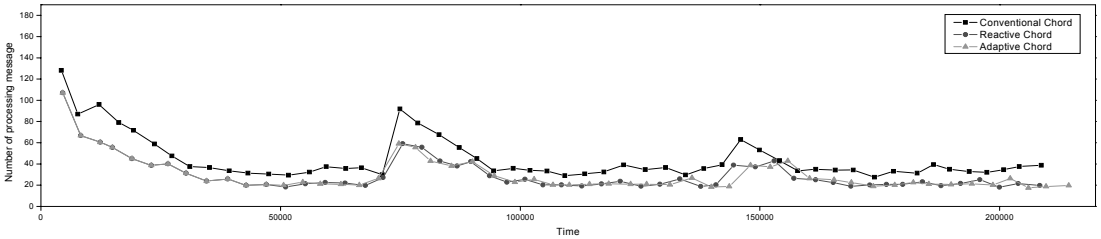
(b) Node B

(그림 12) 시간에 따른 처리 메시지양 (2000node, 0.5sec)

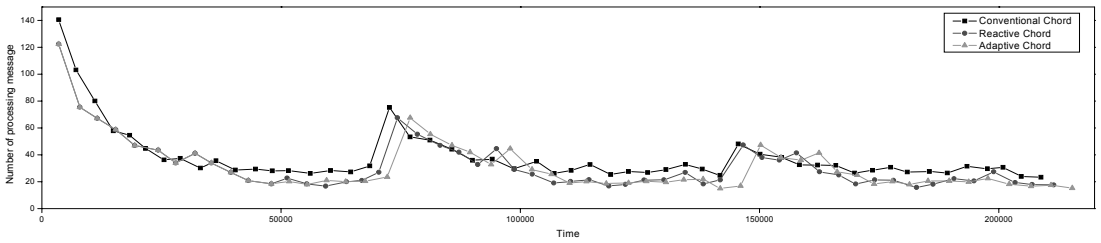
주기가 0.5초 일 경우, 전체 노드 중에서 두 개의 노드를 뽑아서 시간에 따른 각각 알고리즘의 메시지 처리량의 변화를 보여주는 그림이다. (그림 10)-(a),(b)는 총 노드수가 1000개이고 Stabilization 주기가 1초일 경우를 보여준다. 리액티브 Chord방식은 평상시에는 처리하는 메시지 양이 적지만, 요청메시지의 개수가 많아졌을 경우에는 기존의 방식에 비해서 그 양이 크게 증가하는 것을 확인할 수 있다. 그렇지만 기존의 Chord 방식은 평상시에 처리해야 하는 메시지의 양이 리액티브 코드에 비해서 크다는 것을 확인 할 수 있다. 적응적인 Chord 방식은 평상시에는 리액티브 Chord 방식을 사용하기 때문에, 메시지 처리양이 리액티브 Chord와 비슷한 것을 확인 할 수 있고, 요청 메시지의 양이 많은 경우 기존의 Chord 방식으로 변하기 때문에 그 순간에는 리액티브 코드에 비해 메시지 양이 줄어들고 기존의 Chord 방식의 메시지 처리량과 비슷하게 된다. 또한, Stabilization 주기가 빨라질 경우 에는 이전의 Chord 방식이 리액티브 Chord나, 적응적인 Chord에 비해서 메시지 양이 높은 것을 확인할 수 있다[(그림 9)와 (그림 10)을 비교]. (그림 11)과 (그림 12)는 노드 수가 2000개이고 각각 Stabilization 주기가 0.5초, 1초를 나타내며, (그림 13)과 (그림 14)는 노드 수가 4000개 이고 각각 Stabilization 주기가 0.5초, 1초 인 경우를 보여준다.

4.2.2 평균 및 표준 편차

<표 2>는 총 노드수가 1000개이고 Stabilization 주기가 0.5초, 1초일 경우, 전체 노드 중 2개 노드의 이전 Chord, 리액티브 Chord, 적응적인 Chord의 평균 및 표준편차를 수치로 나타낸 표이다. 전체적으로 보았을 경우 이전의 Chord 방식의 평균값이 높은 것을 볼 수 있다. <표 2>에서 Stabilization 주기가 1초일 경우 Node B의 경우에는 리액티브 Chord가 처리 메시지 양이 높지만 이 상황은 네트워크에 요청 메시지의 양이 많아졌을 경우에 이런 수치가 나왔을 것이라고 해석할 수 있다. 제안된 방식은 적응적인 Chord를

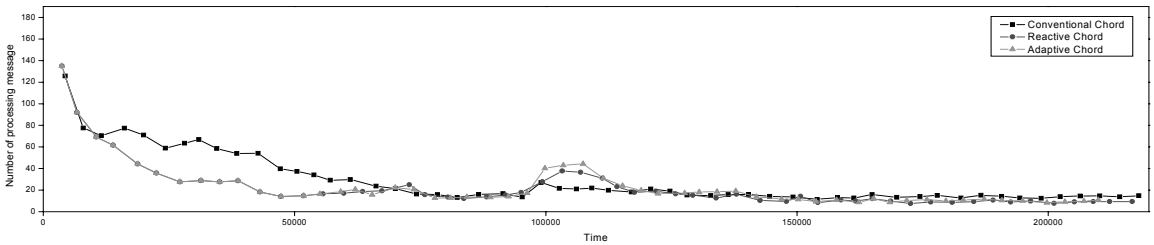


(a) Node A

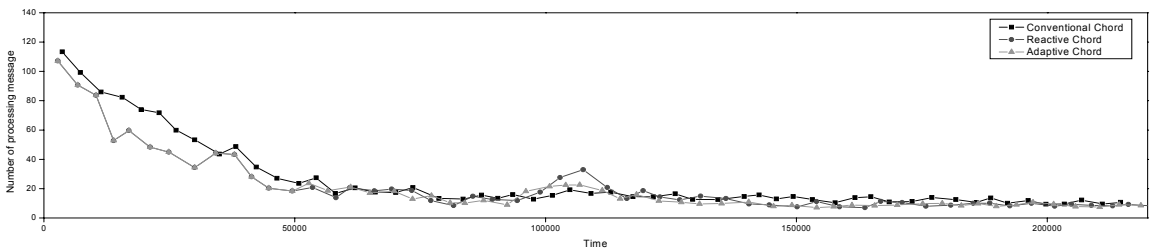


(b) Node B

(그림 13) 시간에 따른 처리 메시지양 (4000node, 0.5sec)



(a) Node A



(b) Node B

(그림 14) 시간에 따른 처리 메시지양 (4000node, 1sec)

〈표 2〉 초당 처리 메시지 양의 평균 및 표준편차 (1000 노드)

1000 NODE	NODE A			NODE B		
	Conventional Chord	Reactive Chord	Adaptive Chord	Conventional Chord	Reactive Chord	Adaptive Chord
Stabilization Period : 0.5 sec						
Average	41.59	28.26	28.17	29.17	27.40	25.94
Standard Deviation	22.98	20.26	20.83	20.17	20.15	19.28
Stabilization Period : 1 sec						
Average	31.06	29.46	23.39	23.44	26.47	22.34
Standard Deviation	23.11	20.55	20.52	22.45	18.56	19.65

〈표 3〉 초당 처리 메시지 양의 평균 및 표준편차 (2000 노드)

2000 NODE	NODE A			NODE B		
	Conventional Chord	Reactive Chord	Adaptive Chord	Conventional Chord	Reactive Chord	Adaptive Chord
Stabilization Period : 0.5 sec						
Average	34.14	26.80	26.33	30.10	23.57	23.49
Standard Deviation	24.15	19.94	19.53	20.30	18.95	19.04
Stabilization Period : 1 sec						
Average	27.36	21.48	20.91	24.01	23.01	21.50
Standard Deviation	23.58	20.76	21.83	24.58	20.91	19.83

〈표 4〉 초당 처리 메시지 양의 평균 및 표준편차 (4000 노드)

4000 NODE	NODE A			NODE B		
	Conventional Chord	Reactive Chord	Adaptive Chord	Conventional Chord	Reactive Chord	Adaptive Chord
Stabilization Period : 0.5 sec						
Average	36.00	25.60	25.60	29.62	24.38	24.68
Standard Deviation	22.99	18.52	18.82	19.61	19.16	19.03
Stabilization Period : 1 sec						
Average	27.25	21.04	21.21	23.82	21.03	19.78
Standard Deviation	24.73	20.55	20.54	24.32	19.84	19.76

보게 되면 전체적으로 낮은 메시지 처리량을 보이고 있다. 어느 경우에는 리액티브 Chord가 처리 메시지 양이 적은 것을 볼 수 있는데, 그 때의 적응적인 Chord의 메시지 처리량을 보면, 거의 차이가 없는 것을 확인 할 수 있다. 이 경우는 네트워크에서의 메시지 요청 수가 Changing Threshold 값을 넘지 않았기 때문에, 알고리즘이 변하지 않고 그대로 리액티브 Chord로 계속 사용되었다는 것으로 해석할 수 있다. <표 3, 4>는 네트워크의 노드수가 각각 2000, 4000개 일 경우의 표이다.

4.2.3 임계값

<표 5>는 Stabilization 주기와 노드 수에 따른 임계값을 나타낸다. 표를 보면 Stabilization 주기가 클수록 임계값이 높게 설정되고, 노드수가 클수록 임계값이 낮게 설정됨을 알 수 있다.

〈표 5〉 Stabilization 주기와 노드 수에 따른 임계값

Node (Log2N) / Stabilization Period (sec)	1000 node (9.9658)	2000 node (10.9658)	4000node (11.9658)
0.5	36.36	33.33	30.77
1	18.18	16.67	15.38

4.3 토론

리액티브 Chord 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. 그렇지만 리액티브 Chord는 근본적인 문제점을 가지고 있다. 어떻게 변할지 모르는 네트워크 상황에 대해서 기존의 Chord 방식보다 성능이 나빠질 수도 있기 때문이다. 갑작스럽게 증가 할 수 있는 요청에 대해서 기존Chord 방식보다 네트워크 트래픽이 더 많아질 가능성이 있기 때문이다.

이런 문제점을 해결하기 위하여 본 논문에서는 상황에 맞춰 변할 수 있는 적응적인 Chord를 제안하였다. 이 방식은 평소에는 리액티브 Chord를 사용하다가 초당 요청 개수가 임계값을 넘어서게 되면 기존의 Chord방식을 사용하는 방식이다. 갑작스런 상황 변화에는 기존의 방식과 같다는 단점이 있을 수 있지만, 평소에는 기존의 방식보다 적은 트래픽을 가지게 됨으로써 기존 방식 보다 노드의 트래픽 처리에 따른 부하를 줄일 수 있는 것이라 생각한다.

5. 결론 및 향후 연구 방향

본 논문에서는, DHT를 위해 제안된 Chord의 문제점 및 이의 문제점을 해결하기 위해 제안 되었던 리액티브 Chord의 문제점을 지적하고, 이를 개선한 적응적인 Chord를 제안하였다. 리액티브 Chord는 평상시의 기존 Chord에서 발생할 수 있는 트래픽 문제에 대해서 해결하였지만, 어떻게 변할지 모르는 네트워크 상황에서 갑작스런 요청 메시지 증가에 따른 트래픽 증가의 문제가 있었다.

적응적인 Chord는 현재 자신에게 들어오는 요청 메시지의 개수를 확인하고, 그에 따라서 리액티브 Chord를 사용할 것인지, 기존의 Chord방식을 사용할 것인지를 정하게 된다. 변화를 위한 임계값은 수식을 통하여 알아보았고, 그 값을 통하여 구현을 하였다. 이를 통해 갑작스럽게 메시지가 많아졌을 경우 리액티브 Chord에서의 문제였던 요청 메시지 증가에 따른 트래픽 증가 문제를 해소 하였다. 제안된 방식은 실험을 통하여 초당 요청 개수가 많아졌을 경우 리액티브 Chord에서 기존의 Chord로 변하는 것을 확인 하였고 전체적인 네트워크의 트래픽양이 줄어들게 되었음을 확인 하였다.

향후 연구 방향을 요약하면 다음과 같다. 적응적인 Chord가 제대로 동작하기 위해서는 정확한 Changing Threshold 값을 필요로 하게 된다. 본 논문에서는 Changing Threshold 값을 알아내기 위하여 네트워크에서 발생할 수 있는 패킷들을 대략적으로 계산하여 만들었다. 이 값을 좀 더 명확히 하면, 제안된 방법보다 더 좋은 성능을 낼 수 있을 것이다. 또한 제안된 Changing Threshold 값에 네트워크 전체 노드

수가 관련이 되어 있다는 것은 수식을 통하여 증명이 되었다. 그렇지만 네트워크의 전체 노드 수를 알아내는 것은 어려운 일이다. 제안된 방법에서는 Finger Table을 이용하여 간접적으로 네트워크의 노드수를 파악하였지만, 좀 더 정확한 네트워크 노드 수 판단 방법이 있다면, 정확한 Changing Threshold 값을 만들어 낼 수 있을 것이다. 또한 임계값을 선택하는 과정에서 파라미터 추정 등의 오차가 발생하였을 경우 어떻게 대처할 수 있는지에 대한 연구가 필요하다.

참 고 문 헌

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

[2] 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.

[3] 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.

[4] 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.

[5] 윤영효, 곽후근, 김정길, 정규식, "트래픽 및 요청 지연을 최소화한 향상된 리액티브 Chord," 정보처리학회논문지C:정보통신, 제16-C권, 제1호, pp.73-82, Feb., 2009.

[6] 윤영효, 곽후근, 김정길, 정규식, "모바일 P2P 환경에서 효율적인 네트워크 자원 활용을 위한 반응적인 코드," 정보과학회논문지:정보통신, 제36권, 제2호, pp.80-89, Apr., 2009.

[7] 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.

[8] 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.

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

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

[11] 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.

[12] 이세연, 장주욱, "Backtracking을 이용한 모바일 에드혹 네트워크에서 Chord 검색 방법," 한국정보과학회 춘계학술대회, pp.517-519, 2004.

[13] 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.

[14] 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.

윤 영 효

e-mail : yyhpower@q.ssu.ac.kr
2006년 숭실대학교 정보통신전자공학부(학사)
2006년 3월~현 재 숭실대학교 정보통신전자공학부 석사과정
관심분야 : 네트워크 컴퓨팅 및 보안

곽 후 근

e-mail : gobarian@q.ssu.ac.kr
1996년 호서대학교 전자공학과(학사)
1998년 숭실대학교 전자공학과(석사)
1998년~2006년 숭실대학교 전자공학과(박사)
1998년 8월~2000년 7월 (주)3R 부설연구소 주임연구원
2006년 3월~현 재 숭실대학교 정보통신전자공학과 postdoc
관심분야 : 네트워크 컴퓨팅 및 보안

김 정 길

e-mail : cgkim@nsu.ac.kr
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~현 재 남서울대학교 컴퓨터학과 전임강사
연구분야 : 멀티미디어 임베디드 시스템, 컴퓨터구조

정 규 식

e-mail : kchung@q.ssu.ac.kr
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~현 재 숭실대학교 정보통신전자공학부 교수
관심분야 : 네트워크 컴퓨팅 및 보안