분산 해시 테이블 기반 P2P 기술 동향

Trend of Distributed Hash Tables-Based P2P

김병오 (B.O. Kim) P2P네트워킹연구팀 위촉연구원
이일우 (I.W. Lee) P2P네트워킹연구팀 선임연구원
박호진 (H.J. Park) P2P네트워킹연구팀 팀장

목 차

정보의 바다라고 널리 알려진 인터넷은 클라이언트-서버 커뮤니케이션 모델을 기반으로 하여 패킷의 송/수신자가 정해져 있는 단방향 통신을 사용하고 있다. 하지만, 실제 사람들 사이의 맨투맨 커뮤니케이션에서는 송신과 수신이 정해져 있는 경우보다 대화에 참여하는 모든 사람들이 송/수신의 역할을 동시에 수행하는 양방향의 형태가 대부분이며, 이러한 불일치는 인터넷을 통해 제공되는 서비스의 가능성을 제한하는 요소가 되고 있다. 이에 따라, 각각의 단말이 송신과 수신의 역할을 동시에 수행할 수 있는 양방향 통신이 가능한 P2P 커뮤니케이션 방법이 고안되었고, 현재, 이러한 P2P를 이용한 다양한 서비스 모델이 제안되고 있다. 본 고에서는 이러한 P2P 네트워크의 구현 기술 중 하나인 분산 해시 테이블을 바탕으로 한 연구의 동향과 그 응용 현황에 대해서 살펴 보고자 한다.

I. 서론

P2P는 기존의 인터넷에서 사용되던 클라이언트-서버 커뮤니케이션의 단방향 특성이 정보를 동시에, 그리고 유기적으로 교환하는 휴먼 커뮤니케이션에서의 상황에 적합하지 않기에 고안된 양방향 커뮤니케이션 모델이다.

이러한 P2P는 크게 unstructured P2P와 structured P2P의 두 가지로 분류할 수 있다. Unstructured P2P는 다시 3가지 형태로 나눠지고, 또한 전체 네트워크에 대한 정보들이 모든 노드들에 의해 관리되거나 한 노드에게 집중되는 반면에, structured P2P 시스템은 각각의 노드가 전체 네트워크가 아닌 부분적인 네트워크 정보를 유지, 관리하게 함으로써, unstructured P2P 시스템의 단점을 보완한 방법이다.

이러한 DHT 알고리듬을 사용하면 네트워크를 구성하고 있는 모든 노드들이 하나의 동일한 프로토콜에 따라 다양한 서비스를 제공할 수 있다. 예를 들면, 네트워크 상에 존재하는 다양한 콘텐츠들을 해싱(hashing) 함수를 이용하여 네트워크 상에 적절히 분산시키고, lookup latency를 최소화할 수 있다. 또한, 관리자가 직접 노드에 새로 참가하거나 떠나는 것을 처리할 필요가 없어 관리 비용을 줄일 수 있다는 장점이 있다. 이러한 DHT 알고리듬의 예로, Chord[1], Pastry[2], CAN[3] 등이 있다.

DHT는 주로 분산 시스템과 애플리케이션 레벨의 멀티캐스트 기술에 응용되고 있다. 예를 들면, CFS[4]는 Chord 알고리듬을 이용하여 분산된 환경에서의 읽기 전용 파일 시스템으로 구현되었고, Pastry 알고리듬은 분산 파일 시스템인 PAST[5]와 애플리케이션 레벨 멀티캐스트 시스템인 SCRIBE[6]의 구현에 적용되었다. 또한, CAN은 OceanStore[7]라는 분산 파일 관리 시스템에 응용되고 있다.

본 고의 II장에서는 P2P시스템을 unstructured와 structured로 구분하고, III장에서는 P2P를 구현하는 대표적인 DHT 알고리듬들과 DHT 기술 연구 동향에 대해서 살펴 본다. 또한, CFS, PAST, SCRIBE, 그리고 OceanStore 등 기존의 DHT 알고리듬들을 이용하여 구현된 응용 사례들을 IV장에서 분석하고, 마지막 V장에서는 본 고에서 다루어진 내용들을 요약·정리한다.

● 용 어 해 설 ●

P2P (Peer to Peer): 기존의 서버와 클라이언트 개념이나 공급자와 소비자 개념에서 벗어나 개인 단말끼리 직접 연결하고 검색함으로써 모든 참여자가 공급자인 동시에 수요자가 되는 형태이다. P2P는 크게 2가지 방식으로 구별되는데, 하나는 어느 정도 서버의 도움을 얻어서 개인간 접속을 실현하는 방식(hybrid P2P)이고, 다른 하나는 클라이언트 상호간에 서버 없이 직접 연결하는 방식(pure P2P)이다.

DHT (Distributed Hash Table): DHT란 시스템 내의 각 노드(node)들이 키 셋을 나누어 가지고 있는 분산 시스템을 말한다. DHT는 크게 keyspace partitioning, overlay network의 두 부분으로 나뉘는데, 해시의 키 집합들을 DHT 시스템 내부의 각 노드에 분산(keyspace partitioning)시키고 DHT의 엔트리 포인트에 상관없이 키의 위치를 찾아갈 수 있도록 라우팅하는 알고리듬을 이용해서 DHT 시스템 내부의 네트워크(overlay network)를 떠돌며 목적지를 찾아가게 된다.

II. P2P의 분류

P2P 시스템은 크게 unstructured와 structured의 두 가지로 분류할 수 있다.

먼저, unstructured P2P 시스템의 종류에는 flooding을 기반으로 한 분산 P2P 방식과 서버를 기반으로 한 중앙집중형 P2P, 그리고 마지막으로 두 가지 방식의 혼합형인 혼성 P2P 방식이 존재한다. 이들의 특징은 새로운 노드의 참여를 위해 복잡한 알고리듬을 사용하여 각각의 노드가 새로운 노드의 참여를 처리해야 하는 로드를 떠맡는 대신에, 일부 노드들의 정보만 업데이트하거나 (혼성 P2P와 중앙집중형 P2P) 혹은 인접한 다른 노드의 정보를 복사해서 사용하면 되기 때문이다(분산 P2P). 또한, 소수의 노드에 의한 데이터의 검색이 가능하기 때문에 lookup 과정에서 복잡한 내용의 쿼리를 처리할 수 있다는 장점도 가지고 있다. 이러한 장점들이 있는 반면에, 정보의 아이덴티티에 대한 정보를 가지고

있지 않아 희귀한 정보의 복제가 불가능하기 때문에 희귀 정보의 손실에 대한 적응력이 낮고, 네트워크에 대한 의존도가 높기 때문에 시스템의 확장성에 문제를 가지고 있다.

반면에, structured P2P 시스템에서는 데이터의 해싱 키와 시스템에 참여한 노드들의 id를 하나의 주소 공간으로 매핑하여 데이터와 노드를 함께 하나의 주소 공간에 두고 관리하는 DHT 알고리듬을 사용한다. 콘텐츠의 lookup 성능 향상을 위해 고안된 이 방법은 노드의 참여 및 이탈을 자동화하여 관리 부담이 적고, 각종 노드 실패의 경우들에도 자동 복구가 가능하게 되어 있다. 하지만, 해싱을 통한 유일한 키 값에 의존하기 때문에 복잡한 쿼리의 사용이 힘들다.

Ⅲ. DHT 연구 동향

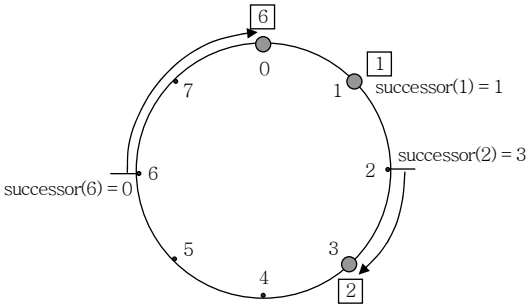
DHT는 해싱을 통하여 생성된 데이터 키 값과 id 짝들을 시스템을 구성하고 있는 모든 노드들에 균일하게 분산하기 위하여 고안된 lookup 방법이다. 데이터 키 값과 id를 이용하여 모든 노드와 데이터들을 동일한 주소 공간에 할당함으로써 데이터와 노드 정보들을 한꺼번에 관리하기가 용이하고, 또한 자동화된 시스템 업데이트가 이루어져 관리의 필요가 없는 P2P 시스템 관리 알고리듬이다. 현재까지 DHT의 연구는 DHT 자체의 알고리듬 개발, 노드들 사이의 부하를 균일하게 분산시키기 위한 load balancing, 그리고 시스템의 신뢰도 향상 기법의 세 가지 측면에서 이루어져 왔다. 따라서, 본 장에서는 먼저 기본적인 DHT 알고리듬들에 대하여 알아보고, 이들 알고리듬들에 적용되어 노드들 사이의 부하를 분산시키고 시스템의 신뢰도를 향상시키기 위해 고안된 방법들을 알아보도록 하겠다.

1. DHT 알고리듬

가. Chord

Chord는 버클리 대학에서 만들어진 방법으로, m-bit($0 \sim 2^m-1$)의 원형 주소 공간에 각각의 노드와 데이터의 키 값을 할당하는 방법이다. 각각의 노드와 데이터의 키 값으로부터 해싱 함수를 이용하여 주소 공간으로의 매핑이 이루어지며, 또한 주소 공간 내에 존재하는 데이터 키 k보다 크거나 같은 노드를 k의 successor라고 명명하여 라우팅의 효율을 높이는 데 사용한다. (그림 1)은 Chord 네트워크를 구성하고 있는 3bit 주소 공간의 예이다. 0, 1, 3의 3개의 노드가 네트워크 상에 존재하고 있으며, 데이터 키 1, 2, 6의 successor로서 1, 3, 0의 3개 노드들이 시스템에 참여하고 있다.

Chord에서는 각각의 노드들이 finger table이란 라우팅 정보를 유지하고 있다. Finger table은 주소 공간의 크기에 따라 그 크기가 달라지는데, m-bit의 경우 m개의 행으로 이루어지게 된다. 각각의 행에는 현재 노드의 위치를 기준으로 하여, 2의 n(1~m) 제곱수에 해당하는 데이터의 successor 정보들을 가지고 있다. Finger table을 바탕으로 이진 트리 기법과 비슷한 방식의 lookup 프로시저가 이뤄진다. <표 1>은 finger table의 구성 원리를 나타낸 표이다. 표에 나타난 원리를 바탕으로 만들어진 정보를



(그림 1) Chord 3bit 주소 공간

〈표 1〉 Finger Table 생성 규칙

Notation	Definition
finger[*k*].*start*	$(n+2^{k-1}) \bmod 2^m,\ 1 \le k \le m$
.*interval*	[*finger*[*k*].*start*, *finger*[*k*+1].*start*]
.*node*	first node ≥ *n*.*finger*[*k*].*start*
successor	the next node on the identifier circle; *finger*[1].*node*
predecessor	the previous node on the identifier circle

바탕으로 하여 어떤 데이터의 키 값을 포함하고 있는 범위를 찾고, 그 범위에 해당하는 데이터들을 관리하는 successor로 쿼리를 전달함으로써 라우팅이 이루어지게 되고, 그에 따른 오버헤드는 log2N이 된다.

새로운 노드의 참가 및 기존 노드의 이탈 등을 자동으로 처리하기 위해서, 각각의 노드는 predecessor와 successor 정보를 가진다. 그리고, 이들 정보들을 이용하여 노드의 참가와 이탈이벤트를 자동으로 처리해 줄 수 있게 된다. 보다 자세히 들여다보면, 새로운 노드의 참가는 참가할 노드의 id를 통해 주소 공간에서의 predecessor와 successor 정보를 새로운 노드를 위한 predecessor와 successor로 설정하고, 정보를 다 받아들인 후 새로운 노드를 기존의 predecessor와 successor의 successor, predecessor로 각각 설정한다. 또한, 노드의 이탈은 이탈하려는 노드가 사전에 이탈 메시지를 자기 주변의 노드들에게 알리고, 이탈 프로시저를 수행하고, 이탈 메시지를 받은 다른 노드들은 자신의 finger table을 업데이트함으로써, 전체 라우팅 정보의 일관성을 유지한다. 또한, 노드 실패에 의한 이탈의 경우에는, 주기적으로 stabilize() 함수를 호출함으로써, 그 파급 효과를 최소화 할 수 있게 된다.

나. Pastry

Pastry는 Chord와 비슷하게 원형의 주소 공간을 사용하고 있다. 하지만, Pastry의 경우 주소 공간의 크기가 128bit로 고정되어 있고, 2진법을 사용하는 Chord와는 달리 2b진법이라는 임의의 진법을 사용한다.

Pastry 네트워크를 구성하고 있는 모든 노드들은 leaf set, routing table, neighborhood set의 3개의 테이블을 관리한다. 이들 정보들은 라우팅을 위한 정보들로, leaf set는 주소 공간 내에서 현재 노드에 가장 근접한 L개의 노드들의 정보들을 유지하고, routing table은 Log2b(N) 행에, 행마다 2b-1개의 노드 정보를 저장하는 것이 가능하다. 각 행은 현재 노드와 다른 노드들 사이의 공통 프리픽스의 길이가 짧은 원격 노드들을 가장 먼저 배치하고, 공통 프리픽스가 긴 것들을 테이블의 아래쪽에 배치한다. 마지막으로, neighborhood set는 현재 노드와 네트워크 상으로 가장 근접한 노드들의 정보를 저장한다. Neighborhood 역시 총 L개의 노드들을 저장한다. Pastry에서의 라우팅은 위의 3가지 요소 중 가장 먼저 leaf set를 찾아서 네트워크 상에서 가장 근접한 노드를 가져오거나, 혹은 leaf set에 존재하지 않을 때 routing table을 찾아보고 가장 긴 공통 프리픽스를 지닌 노드로 쿼리를 전달하여 라우팅을 행하게 된다. 만약에 routing table에서도 목표 노드를 찾을 수 있는 링크가 존재하지 않는다면, 현재 노드가 가지고 있는 정보 중, 목표 노드의 주소에 가장 근접한 주소를 가진 노드에 쿼리를 재전송하여 라우팅을 하게 된다. 이러한 라우팅 방법을 통하여 Pastry에서는 Log2b(N)의 라우팅 성능을 가질 수 있다. (그림 2)는 Pastry 논문에 게재되어 있는 3개의 테이블에 대한 예시이다.

Pastry에서는 새로운 노드 X가 참여하기 위해서

NodeId 10233102

Leaf set

Smaller		Larger	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table

-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Neighborhood set

13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

(그림 2) Pastry Peer 노드의 Leaf Set, Routing Table, Neighborhood Set

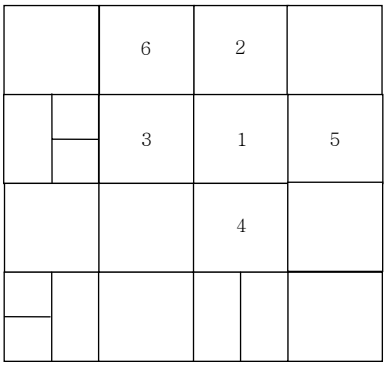
이미 오버레이에 속해 있고, X로부터 물리적으로 가까운 위치에 있는 한 노드 A를 알아야 한다. X는 A에 참여 요청을 하고, A는 참여 요청을 받은 직후에 X가 신청한 id와 가장 근접한 id를 가진 노드 Z로 참여 메시지를 보낸다. 참여 메시지가 Z로 가는 동안 지나간 모든 노드들은 자신이 가진 라우팅 정보들을 모두 X로 보내게 되고, X는 이 정보들을 조합하여 자신만의 라우팅 정보를 생성한다. Leaf set의 경우, Z에 논리적으로 가장 근접해 있기 때문에 Z의 leaf set를 이용하여 생성하고, neighborhood set의 경우에는 물리적으로 가장 근접한 A의 neighborhood set의 정보를 이용하여 생성한다. 마지막으로, X는 A의 common prefix부터 시작하여 Z까지의 중간 노드들의 라우팅 정보들을 바탕으로 자신의 라우팅 테이블을 생성한다.

다. CAN

CAN에서는 Chord나 Pastry와는 달리, d-차원의 데카르트 좌표계에 기반을 둔 주소 공간을 사용한다. 각 노드의 id와 데이터 키는 d-차원의 한 점으로 매핑이 되고, 각각의 노드는 전체 주소 공간을 최대한 균등하게 배분받아 관리하게 된다. 주소 공간을 균등하게 배분함으로써 각각의 노드가 균등한 작업량을 가질 수 있도록 한다.

CAN에서는 서로 인접한 공간을 가진 노드들, 즉 d-1차원은 동일하고, 한 차원에서 서로 인접한 노드들을 neighbor라고 정의하고 neighbor 노드들의 정보를 테이블로써 저장한다. 따라서, d차원의 주소 공간에서 각각의 노드들은 모두 2d개의 엔트리를 가진 neighbor 테이블을 갖는다. 예를 들면 (그림 3)과 같은 2차원의 CAN 시스템에서는 전체 좌표계가 5개의 구역으로 나누어지게 되고, 각각의 공간들은 최대 4개의 인접한 neighbor 노드들이 존재하게 된다.

이러한 neighbor 테이블의 정보를 바탕으로 이루어지는 라우팅 프로시저는 인접한 neighbor들 중에 원하는 좌표에 가장 근접한 공간을 가진 neighbor로의 전송을 수행하는 greedy 알고리듬을 사용



(그림 3) 2-차원에서의 CAN 데카르트 좌표계

하며, 이 때의 라우팅 경로의 길이는 (d/4)(N1/d)이 된다. 따라서, 차원 d가 커짐에 따라 경로의 길이가 짧아지게 되고, 라우팅 성능의 향상을 가져올 수 있게 되며, 라우팅 테이블의 크기가 커지기 때문에 그에 따른 유지 관리를 위한 오버헤드가 커지게 된다. 따라서, 테이블의 유지 관리를 위한 비용과 라우팅 성능 향상의 두 가지 상호 작용을 충분히 고려하여 주소 공간의 d값을 정해야 한다.

2. 부하 분산 기법

가. 가상 서버 기술

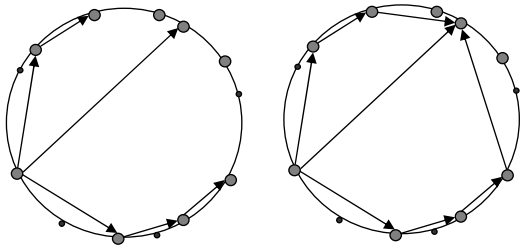
가상 서버라고는 하지만, 실제로는 가상의 노드를 하나의 노드가 관리하는 형태의 방식이다. 다시 말해서, 하나의 노드를 여러 개의 노드인 것처럼 보이게 하여 여러 개의 주소 공간을 할당 받음으로써 각각의 노드에 의해 관리되는 주소 공간을 확장시키는 것이다. 이에 따라, 각각의 노드에 저장되는 데이터량을 균일하게 분포시킬 수 있고, 이로써 각 노드들이 균일한 저장 공간을 사용하게 된다. 가상 서버 기술[8]은 노드들 사이의 저장 공간에서 부하의 불균일이 발생할 때, 가상 서버를 과부하 상태의 노드로부터 다른 노드로 이동시키는 과정을 문제가 해결될 때까지 수행하며, 이 과정을 transfer라고 한다. Transfer는 세 가지 규칙을 바탕으로 이루어지며, 그 중 첫째는 과부하 노드로부터 이동되는 가상 서버가 새로운 과부하 노드를 만들지 않아야 한다는

것이고, 둘째가 transfer 이후에 과부하 문제를 해결할 수 있는 가장 부하가 적은 가상 서버가 이동되어야 한다는 것이다. 마지막 셋째는 노드에 속해 있는 어떠한 가상 서버도 과부하를 없앨 수 없을 경우로, 이 때에는 가장 부하가 큰 가상 서버를 이동시켜 부하를 최소화한다. 이러한 transfer의 과정은 일대일, 일대다, 다대다의 세 가지 형태를 가진다. 이들 중 일대일 transfer는 시스템 상에 존재하는 두 개의 노드를 임의로 선택했을 때, 둘 중의 하나가 과부하 노드이고 하나가 아닐 때에 둘 사이의 transfer 과정을 실행할 때 사용된다. 이 때, 부하를 체크하는 과정을 과부하 상태가 아닌 노드가 수행하게 함으로써 각 노드에 걸리는 부하를 최소화할 수 있고, 또한 두 노드가 모두 과부하 상태인 경우에 transfer와 관련된 부하가 전혀 발생하지 않게 한다. 그리고, 일대다의 경우, 부하가 걸리지 않은 노드들을 하나의 그룹으로 처리하고, 과부하 상태의 노드와 그룹 내에서 가장 부하가 적은 노드 사이에 transfer를 수행하고, 다대다 transfer의 경우에는 일대다의 확장 개념으로써, 과부하 노드와 부하가 적은 노드들의 두 그룹 사이에서 transfer를 수행하게 함으로써 구현된다.

나. Power of Two Choices

Power of Two Choices[9]는 시스템을 구현할 때, 여러 개의 해시 함수를 이용하여 여러 개의 선택지들을 만든 이후에, 그 중에 가장 부하가 적은 노드에 데이터를 저장시키는 방법이다. 이를 구현하는 데에는 parallel과 redirection의 두 가지 방식이 존재하고 있고, 그 중에서 가장 간단한 parallel 방식은 처음에 데이터를 저장할 때를 포함하여, 데이터를 검색할 때마다 해시 결과값을 새로이 구하여 매번 여러 개의 쿼리를 이용하여 검색함으로써, 네트워크 사용량을 비롯하여 노드에도 부하를 주게 된다. Redirection 방식의 경우, 데이터를 저장할 때에는 parallel 방식과 똑같이 여러 개의 쿼리를 이용하여 데이터를 저장하지만, 데이터를 저장한 이후에 같은 데이터를 통해 이루어지는 라우팅의 결과가 하나의 노드로 수렴할 수 있도록 redirection 포인터를 사용하게 된다. 따라서, 새로운 정보를 저장하는 단계인 데이터 저장 단계에서는 두 가지 방식 모두 여러 개의 쿼리가 여러 개의 경로를 동시에 사용하지만, 정보의 검색 단계에서는 여러 개의 가능한 선택지들 중에서 임의의 한 선택지를 통해서 단 하나의 쿼리를 이용한 검색이 가능하게 되므로 네트워크 사용량은 물론, 하나의 검색을 위한 참여 노드의 수가 줄어들면서 시스템 전체의 부하도 줄일 수 있다. 또한, 여러 노드가 한꺼번에 데이터를 요청했을 시에 여러 개의 경로를 이용하여 검색이 가능하기 때문에 보다 좋은 검색 성능을 기대할 수 있다. (그림 4a)와 (그림 4b)는 각각 parallel 방식에서의 라우팅 경로와 redirection 방식에서의 라우팅 경로의 그림이다.

이외에도 redirection 방식을 사용하면, soft state 방식을 적용하여 보다 더 동적인 환경에 적응할 수 있는 시스템을 만들 수 있게 된다. 이는 주기적으로 업데이트 해줘야만 하는 데이터의 위치를 바꿈에 있어서 정적인 해시 기능을 이용하는 이상, 데이터가 저장될 노드는 정해져 있기 때문에 하나의 데이터에 의해 선택된 노드들끼리 기존에 저장을 수행했던 노드의 간섭없이 상호간의 데이터를 독립적으로 새로이 배치함으로써 시스템의 부하를 줄일 수 있기 때문이다.



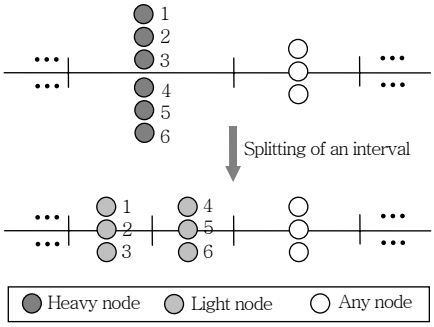
(그림 4) Power of Two Choices 방식에서의 라우팅 경로

다. 열 분산[10]

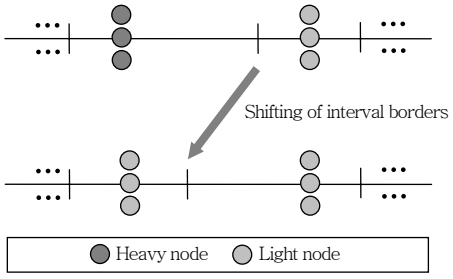
전체 주소 공간을 보다 작은 단위인 구간으로 나누어 관리하는 방법이다. 구간은 구간끼리 서로의 시스템 부하를 관리하고, 같은 구간에 속한 노드들

끼리는 같은 구간 정보를 가진다. 여기서 구간은 주소 공간을 균등하게 나누는 것이 아니라, 데이터 공간을 균등하게 나누기 위한 것으로 사용자가 정해주는 노드 수 f에 의해 결정된다. 이 방법에서는 각각의 노드가 관리하는 데이터양이 증가할 때 구간들 사이의 부하를 분산시키기 위한 방법은 구간 분할, 노드 이동, 그리고 경계 이동의 세 가지를 사용하고 있다. 구간 분할은 구간에서 과부하가 발생했을 때 그 구간에 포함된 노드의 수가 2f 이상일 경우에 일어나고, 이것으로 인해 구간에서 발생한 과부하가 줄어들게 된다. 또한 노드 이동은 과부하가 발생한 구간에 f개 이상의 노드들이 존재할 때에, 경계 이동은 f보다 적을 때 발생하게 되는 것이다. (그림 5)~(그림 7)은 각각의 구간에서 동작하는 부하 분산 방법의 예이다.

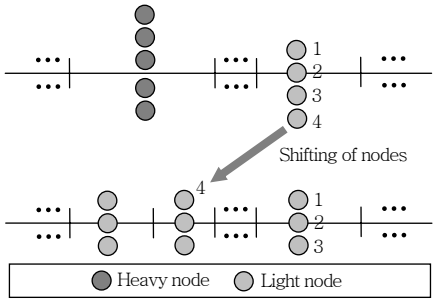
한 구간에 해당하는 모든 데이터들이 구간 내의 모든 노드들에게 복제되어 있기 때문에 데이터양이 커지게 되면 모든 노드가 동일한 과부하를 발생시키게 되고, 이 때에 구간 내에 존재하는 노드의 수에 따라 각종 부하 분산 작업을 수행하게 된다.



(그림 5) 구간 분할 예제



(그림 6) 노드 이동 예제



(그림 7) 경계 이동 예제

라. 단일 주소 공간과 데이터 분산[11]

단일 주소 공간은 가상 서버와 비슷한 개념인 가상 노드를 사용하는 것이다. 가상 서버의 경우, 한 노드에 여러 개의 가상 노드 프로세스들을 실행시키고 있는데, 이 때에 n개의 가상 서버를 위해 n개의 소켓을 열어야 하고, 이로 인해 병목 현상이 발생하게 된다. 왜냐하면 노드들의 네트워크 대역폭에 한계가 존재하기 때문이다. 많은 가상 서버들이 한 노드에서 실행되는 상황에서는 각각의 가상 서버들 모두가 그들 서버에서 관리하는 노드들의 네트워크 대역폭을 균일하게 할당받기 때문에 네트워크 성능의 한계가 오는 것이다. 따라서, 이러한 네트워크 대역폭 한계에 대한 대책으로 나온 것이 가상 노드의 개념인데, 가상 노드의 경우 여러 개의 가상 노드가 한 노드 상에서 실행된다는 건 가상 서버의 방식과 동일하지만 가상 서버처럼 항상 활성화된 상태가 아닌 현재 사용되는 가상 노드 하나만을 활성화된 상태로 유지함으로써 저장 공간의 균일화와 네트워크 최적화의 두 가지 목적을 이룰 수 있는 것이다. 그리고 데이터의 불균일 분산으로 인한 과부하의 경우, 경계 이동 방식과 유사한 방법을 사용한다. 즉, 과부하가 발생한 노드의 주소 공간의 일부를 과부하 상태가 아닌 노드의 주소 공간으로 바꾸는 것이다.

3. 신뢰도 향상 기법

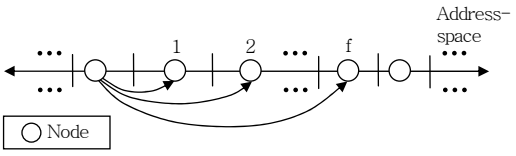
가. 조각화

P2P에서는 부분적인 데이터 손실이 전송중에 일어날 가능성이 높고, 이로 인한 시스템의 성능 저하

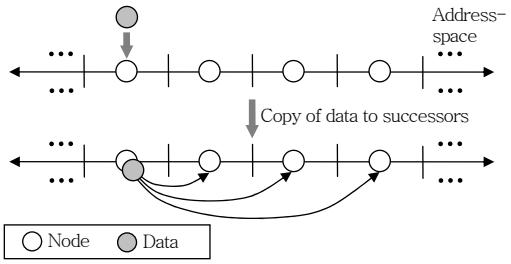
가 나타나게 된다. 이러한 성능 저하를 최소화하기 위하여 고안된 방법이 조각화(fragmentation)[12]이다. 하나의 파일을 여러 개의 데이터 조각으로 나누는 방법으로, 중복되는 부분들을 조각화하여 중복되는 부분들 중에 하나씩만 성공적으로 전송된다면 전송받은 데이터들로 기존 데이터를 복구할 수 있으므로 데이터의 전송도 성공적으로 이루어진다고 생각할 수 있다. 다시 말해서, 하나의 파일을 N개의 중복되지 않는 데이터들과 K개의 중복되는 부분으로 나누어 최소한 N개의 데이터 조각들이 성공적으로 전송된다면 이런 데이터 조각들을 전송받은 노드들이 원래의 데이터를 복구할 수 있게 하는 것이다. 하지만, 이 방법을 쓰더라도 N개의 데이터 중에 손실이 발생할 시에는 에러를 복구할 방법이 없기 때문에 추가적으로 데이터 복제를 이용하여 시스템의 신뢰도를 향상시켜야 한다.

나. Successor-List

Successor-List는 Chord에서 설명된 데이터 구조로써, r개의 successor들을 저장하는 리스트이다. 여기서 복제는 Successor-List에 존재하는 r개의 successor들에게 데이터들을 분산시켜 두는 것이다. (그림 8)과 (그림 9)는 Chord 알고리듬에서 나타나는 Successor List의 예이다. 그림에서는 r=f개의 successor들을 리스트에 연결해 놓은 모습이다.



(그림 8) Chord의 Successor-List

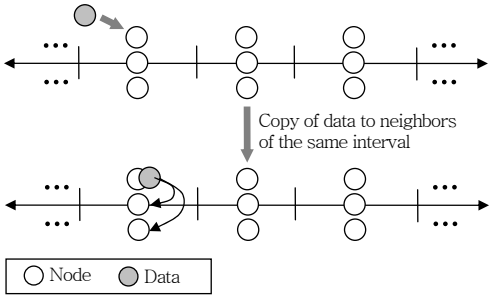


(그림 9) Successor-List를 이용한 데이터의 복제

따라서, 어떤 한 successor가 실패하더라도 f개가 동시에 실패하지 않는 이상, 리스트에 존재하는 가장 근접한 successor로부터 정보를 얻어올 수 있기 때문에 신뢰할 수 있는 시스템을 구현할 수 있다.

다. 구간 내 복제 방식

(그림 10)은 열 분산(heat dispersion) 방식에서 언급된 방법으로, 구간 내에 속한 모든 노드들이 동일한 데이터를 복제해서 관리하는 방법이다. f개의 노드로 이루어진 구간에 존재하는 모든 데이터들을 f개의 노드들에 복제시키고, 한 구간에 속한 f개의 노드들이 모두 동시에 실패하지 않는 한 데이터의 신뢰성을 보장해 준다. 또한, 가상 서버를 사용하지 않고 가상 노드의 개념을 사용함으로써 부하를 분산시키는 효율은 더 좋아지지만, 가상 노드의 활성화 및 비활성화를 위한 새로운 부하가 노드에 부여된다. 따라서, 네트워크 대역폭 제한에 의한 성능 저하와 가상 노드의 활성화/비활성화에 의한 성능 저하 중 어느 쪽이 더 큰 영향을 미치는지 사전에 알아보고, 가상 서버를 사용할지 가상 노드를 사용할지에 대한 결정이 필요하다.

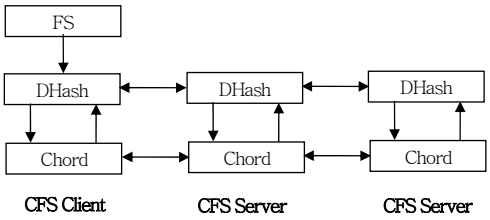


(그림 10) 구간에 속한 다수의 가상 노드로의 복제

Ⅳ. DHT 응용 분야

1. CFS

CFS는 P2P 기반의 읽기 전용 파일 시스템이다. CFS는 (그림 11)과 같이 세 계층의 중추적인 핵심



(그림 11) CFS 계층 구조

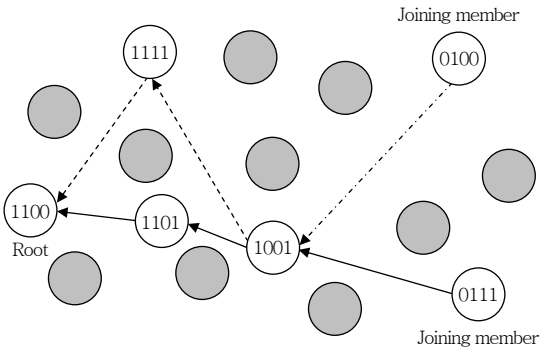
요소들로 이루어진다.

가장 하위 계층인 Chord 계층에서는 기본적인 Chord의 라우팅 기법을 제공하고 있으며, 데이터의 정확성과 시스템의 내고장성 등의 추가적인 사항을 고려하여 r개의 successor들의 정보를 유지할 수 있는 successor 리스트를 생성하여 관리한다. 따라서, successor 중 하나가 실패하더라도 다른 successor를 이용하여 작업을 끝까지 수행할 수 있도록 했다. 두번째 계층인 DHash layer에서는 각각의 successor가 실패할 경우에 발생할 데이터 손실을 막기 위해 k개의 복제 데이터를 만들어 k개의 서버에 분산시키는데, 이 때 k가 r보다 작거나 같은 수라는 조건을 두고 있다. 이는 최대 r개의 successor들에게 데이터를 복제시켜 두어 r개의 successor들이 동시다발적으로 실패하는 일을 막기 위한 것이다. 또한, 캐싱 메커니즘을 사용하여 자주 사용되는 데이터들을 임시로 복제해서 임의의 successor에 과부하가 걸리지 않을 수 있도록 조절한다. 그리고, 각종 기기들의 능력에 맞게 부하를 분산시키기 위해 한 노드가 여러 개의 가상 노드들의 역할을 수행할 수 있도록 하여 각 기기들의 저장 공간의 크기에 따라 부하를 분산시킬 수 있도록 하였다. 가장 상위에 위치하는 파일 시스템에서는 전반적인 파일들의 관리와 관련된 함수들을 제공해 준다.

2. SCRIBE

애플리케이션 레벨에서의 멀티캐스트를 위해 고안된 SCRIBE는 Pastry를 기반으로 하여 만들어진 멀티캐스트 인프라스트럭처이다. 그룹 id를 할당받은 랑데부 포인트를 이용한 멀티캐스트 방식으로, 그룹 id는 그룹의 이름과 그룹의 생성자의 이름을 연결시킨 스트링을 해싱 함수를 통하여 변환시켜 얻게 된다. 멀티캐스트를 위한 그룹에 가입하기 위해서는 참여 메시지를 랑데부 포인트로 보내야 하는데, 이를 위해 각각의 노드들은 Pastry의 라우팅 알고리듬을 이용한다. 참여 메시지가 거쳐가는 경로에 속한 노드들을 forwarder 노드로 참여 요청 메시지를 받은 노드들은 그 자신이 이미 그룹에 속해 있는 경우와 아닌 경우로 나뉠 수 있다. 이미 그룹에 속해 있는 경우에는 요청을 보낸 노드들을 자신의 children 테이블에 추가하는 과정만 거치면 된다. 그룹에 참여된 노드가 아닐 경우에는 그룹에 대한 children 테이블을 생성하고, 테이블에 참여를 원하는 노드를 추가한 후 자신도 또한 참여 메시지를 송신하여 forwarder가 된다. 여기서 forwarder 노드란 멀티캐스트 트리 상에서 leaf node가 아닌 노드들을 지칭하며, 이들이 멀티캐스트 트리에서의 중간 매개자가 된다. 또한, children 테이블을 관리하는 노드들은 주기적으로 자신이 정상적으로 작동한다는 광고 메시지를 children 노드들에게 송신하고, 만약에 어떤 시간동안 children 노드들이 이러한 광고 메시지를 받지 못한다면 forwarder 노드 혹은 루트 노드가 실패했다고 생각할 수 있다.

(그림 12)는 SCRIBE에서 구축된 멀티캐스트 트리의 구조로써, 노드 1101이 실패했을 때의 실패 복구 과정을 보여 준다. 노드 1101의 child 노드인 1001이 노드 실패를 인식하고, 참여 요청 메시지를 생성하여 Pastry 알고리듬을 이용하여 그룹에 조인



(그림 12) SCRIBE 멀티캐스트 트리의 재구성

하기 위해 메시지를 보내게 되고, 이에 따라 새로운 forwarder 노드인 1111 노드가 그룹에 참여하게 된다.

3. PAST

PAST는 128bit 노드 id와 160bit 파일 id를 이용하여 주소 공간을 구축하고, Pastry 알고리듬을 적용한 lookup과 자기 조직화 메커니즘을 적용하였다. 노드 id는 스마트카드의 공개 키로부터 얻어지고, 파일 id의 경우에는 파일의 이름, 소유자의 공개 키, 그리고 랜덤 변수로부터 얻어진다. 특히, 파일의 경우 최상위의 128bit만이 라우팅 및 자기 조직화에 사용된다. 또한, 새로운 파일이 추가될 경우, k개의 노드에 중복시켜 시스템 전체의 내고장성을 향상시켰다. PAST라는 P2P 기반의 분산 파일 시스템의 인상적인 점은 각 노드에 동일한 부하를 주기보다는 각 노드의 성능과 저장 공간의 크기에 따라 그 부하를 달리 분산시키는 방법을 적용하고 있다는 점이다. 이는 각각의 노드에 해당하는 다양한 종류의 기기들의 능력을 최대한 활용할 수 있도록 하여 보다 효율적인 시스템 구축에 기여할 수 있는 방법이다. 데이터를 얻기 위한 검색 및 라우팅은 Pastry 알고리듬을 그대로 적용하였으나, 각기 다른 크기의 파일과 서로 다른 저장 용량을 가진 실제 환경을 고려하여 데이터를 분산함으로써, 시스템에 존재하는 노드들의 이질성을 충분히 고려하고 있다. 또한, 파일을 읽고 씀에 있어서 캐싱을 사용하여 자주 사용되는 데이터를 새로이 원격지로부터 로드할 때 발생하는 오버헤드를 최소화하고 있다.

4. OceanStore

OceanStore는 CAN의 방식이 적용된 P2P 기반의 글로벌 파일 시스템으로 GUID를 이용하여 노드를 분별하고, 파일의 변경자와 사용자들에 대한 액세스 컨트롤을 제공한다. GUID는 유저가 의미를 파악할 수 있는 이름값과 파일의 소유자의 키 값을 해싱 함수를 통해 변화시킨 값을 사용하고 있다. 또한, 파일의 사용자와 변경자에 대한 액세스 제한을 두고 있는데, 이는 작업 그룹과 같은 정책을 통해 정해지게 된다. 파일의 사용자에 대한 액세스 컨트롤은 사용 권한을 가진 노드들에게만 공개된 암호화 키를 이용하여 이루어지게 되고, 파일의 변경자에 대한 액세스 컨트롤은 액세스 컨트롤 리스트에 의해서 관리된다. OceanStore에서 사용하는 CAN 알고리듬은 Plaxton의 랜덤화된 계층적 분산 데이터 구조를 사용하고 있다. 노드 id를 표현할 때, n개의 니블을 사용하고, 노드 간의 링크들을 각각의 니블 위치에 해당하는 레벨로 분류하게 된다. 따라서, 링크의 레벨에 따라서 계층적인 lookup이 수행된다. 여기서 n은 CAN에서 말하는 차원의 복잡도, 즉 d에 해당된다. 따라서, OceanStore의 라우팅 성능은 $(n/4)(N^{1/n})$이 된다. 또한, 지역성을 고려하기 위해 자료의 복사가 수행될 때 그 위치를 현재 노드에서 라우팅 트리의 루트 사이에 존재하는 모든 노드에 링크시키고, 이러한 링크 포인터를 이용하여 라우팅을 수행한다.

Ⅴ. 결론

지금까지 P2P 시스템의 분류 특성을 알아보고, 그 중에서도 현재 각광받고 있는 structured P2P, 즉 DHT 알고리듬들의 연구 현황과 그 응용 사례들에 대해 살펴 보았다. 현재, DHT가 응용되고 있는 영역은 DHT가 가지고 있는 쿼리 처리 능력의 한계 때문에 주로 파일 공유의 목적으로 주로 쓰이고 있으며, id를 이용한 라우팅의 특성 덕분에 멀티캐스팅으로도 응용되고 있다. 하지만, 그 이외의 목적으로 사용되는 데에는 id만을 사용한 lookup 메커니즘의 한계를 뛰어넘어 퍼지 쿼리와 같은 기능을 사용 가능하게 하는 방법이 고안되어야 할 것이다. 또한, PDA나 휴대폰 같은 모바일 기기들이 PC와 같은 기기들과 함께 네트워크를 구성하는 유무선 통합 환경에서는 부하의 균등한 분산보다는 각각의 기기의 성능에 맞는 부하의 분배가 요구되므로 이에 따른 고려가 필요하다. 이는 이미 CFS, PAST 등에서 고려

되고 있으나 이들 방법들은 저장 공간과 프로세싱 능력에 따른 분배가 가능할 뿐이고, 각 노드의 네트워킹 능력을 고려하지는 못한다. 따라서, 이러한 자원의 비균등 분배에 대한 알고리듬도 추가적으로 개발되어야 한다.

약 어 정 리

CAN	Content Addressable Network
CFS	Cooperative File System
DHT	Distributed Hash Tables
P2P	Peer-to-Peer

참 고 문 헌

[1] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. of the 2001 ACM SIGCOMM Conf.*, 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 Distr. Systems Platforms (Middleware)*, 2001, pp.329-350.

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

[4] F. Dabek, F. Kaashoek, R. Morris, D. Karger, and I. Stoica, "WideArea Cooperative Storage with CFS," *in Proc. of ACM SOSP'01*, Banff, Canada, Oct. 2001.

[5] P. Druschel and A. Rowstron, "PAST: A Large-scale Persistent Peer-to-Peer Storage Utility," *in Proc. HOTOS Conf.*, 2001.

[6] M. Castro, P. Druschel, A.M. Kermarrec, and A. Rowstron, "Scribe: A Large-scale and Decentralized Application-level Multicast Infrastructure," *IEEE Journal on Selected Areas in Comm. (JSAC)*, Vol.20, No.8, Oct. 2002.

[7] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: An Architecture for Global-scale Persistent Storage," *in Proc. of the 9th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. ACM*, Nov. 2000.

[8] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load Balancing in Structured P2P Systems," *in Proc. of 2nd Int'l Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, USA, 2003.

[9] J. Byers, J. Considine, and M. Mitzenmacher, "Simple Load Balancing for Distributed Hash Tables," *in Proc. of 2nd Int'l Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, USA, IEEE, 2003,

[10] S. Rieche, L. Petrak, and K. Wehrle, "A Thermal-Dissipation-based Approach for Balancing Data Load in Distributed Hash Tables," *in Proc. of IEEE Conf. on Local Computer Networks (LCN 2004)*, Tampa, USA, 2004.

[11] D. Karger and M. Ruhl, "Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems," *in Proc. of 4th Int'l Workshop on Peer-to-Peer Systems (IPTPS '04)*, San Diego, USA, 2004.

[12] Simon Rieche, Heiko Niedermayer, Stefan Götz, and Klaus Wehrle, "9. Reliability and Load Balancing in Distributed Hash Tables," *Lecture Notes in Computer Science*, Vol.3485, Nov. 2005, pp.119-135, DOI 10.1007/11530657_9, http://dx.doi.org/10.1007/11530657_9