P2P 시스템에서 안전성과 성능을 고려한 노드 복구와 복제 기법

차봉관[O] 박성환 손영성 김경석
부산대학교 컴퓨터공학과
{bgcha[O], shpark, ysson, gim2005@asadal.cs.pusan.ac.kr

Replication and Node Recovery for Efficiency and Safety in P2P sytmem

Bonggwan Cha[O], Sunghwan park, Youngsung Son, Kyongsok Kim
Dept. of Computer Science and Engineering, Electronics and Telecommunications Research institute, Division of Computer Science and Engineering, Pusan National University

요 약

최근에 P2P(Peer-to-Peer) 시스템에서 효율적인 자원 탐색 방법에 대해 많이 연구되고 있다. 대부분의 P2P 시스템은 overlay network를 형성하므로 노드와 노드 사이의 물리적인 거리를 고려하지 않는다. 그래서 서로 이웃한 노드라도 실제 물리적인 latency가 클 수 있다는 문제점을 가지고 있다. 이런 문제점을 해결하기 위해 Topology를 고려한 계층적 시스템을 설계하였다. 이 시스템을 TB-Chord(Topology-based Chord)라 부른다. TB-Chord는 자신의 subnet에 Global network에 있는 데이터의 사본(Replica)을 저장하기 때문에 저장공간(storage)의 낭비와 노드가 떠날(leave) 때 데이터의 이동에 따른 네트워크 부하가 생기는 문제가 있다. 이 논문은 효율적인 복제 기법을 이용하여 저장공간과 네트워크의 효율성을 높이고 노드 fail에 대한 rejoin 메카니즘을 사용하여 효율적으로 시스템을 회복하는 방법을 제안한다.

1. 서 론

P2P 시스템은 P2P 시스템에 참여하는 노드들의 자원을 공유하는 분산 시스템으로 여기에 참여하는 노드들은 서버와 클라이언트의 역할을 모두 수행한다. 대부분의 P2P 시스템의 가장 핵심이 되는 기능은 데이터를 효율적으로 저장하는 일이다. 현재 가장 잘 알려진 P2P 시스템으로는 파일 공유시스템인 Napster[1]와 Gnutella[2]가 있으며, 이 시스템들은 각 노드에 파일을 저장하고 파일을 원하는 노드는 시스템에 속한 그 파일을 저장하는 노드로부터 직접 파일을 전송 받는다. 대부분의 P2P 시스템의 가장 중요한 기능은 데이터를 효율적으로 배치하고 탐색(lookup)하는 것이다. 그러나 Napster나 Gnutella의 경우 확장성(scalability) 문제를 가지고 있다.

확장성 문제를 해결하기 위해 확장 가능한 P2P overlay network(CAN[3], Chord[4], Pastry[5], Tapestry[6]) 들이 제안되었다. P2P overlay network은 데이터의 배치와 탐색 알고리즘을 제공한다. 즉, 데이터들이 저장되어 있는 위치 정보들이 overlay network상의 노드들에 분산 배치되기 때문에 새로운 데이터를 저장하기 위해서는 어떤 노드에 저장할 것인지를 결정하는 방법과 어떤 데이터가 저장된 위치를 알기 위해 데이터의 위치 정보를 가지고 있는 노드를 찾기 위한 탐색 알고리즘과 관련이 있어야 한다.

P2P 시스템은 물리적인 거리를 고려하지 않아서 P2P 시스템상에 가까워도 실제 네트워크에서는 여러 라우터를 경유하기 때문에 물리적 latency가 클 수가 있다. 이런 문제점을 해결하기 위해 Topology를 고려한 계층구조인 TB-Chord를 제안하였다. 하나의 subnet에서는 하나의 Super peer[7]를 두어 모든 노드의 join과 leave은 Super peer를 통해서 이루어 진다. TB-Chord 시스템 안에서 각 subnet는 탐색할 때나 저장할 때 자신의 subnet에 데이터가 없으면 Global network상에 있는 데이터들의 사본(replica)을 저장하므로 저장공간의 낭비가 발생하며 노드가 조인하거나 떠날 때 데이터 사본들이 그 노드의 successor로 이동하기 때문에 네트워크의 부하가 발생하는 문제가 생긴다. 또한 Super peer는 시스템에서 중요한 역할을 하기 때문에 Super peer에 fail이 발생하면 전체 시스템에 문제가 생기며 Super peer를 복구하는데 많은 비용과 시간을 소비한다. 이 논문에서는 보다 효율적인 사본을 관리하는 복제 기법, 성능을 고려하여 Super peer를 선택하는 방법과 Fault-tolerance에 대해 제안한다.

이 논문의 구조는 다음과 같다. 2절에서 TB-Chord의 구조와 기능에 대해서 간략하게 소개하고, 3절에서는 데이터의 사본에 대한 복제 기법에 대해 설명하며, 4절에서 노드의 fail에 대한 복구 방법에 대해 설명한다. 마지막으로 5절에서는 결론을 제시하고 있다.

2. TB-Chord

TB-Chord는 모든 노드로 이루어진 Global network와 물리적으로 가까운 노드들로 이루어진 여러 개의 subnet으로 이루어진 시스템이다. 각 노드는 Global network에 존재하고 그와 동시에 하나의 subnet에 속해 있다. 각 노드는 자신의 식별자를 가지기 위해 자신의 IP주소를 2개의 해쉬 함수(h1, h2)를 사용하여 global-nodeID와 local-no

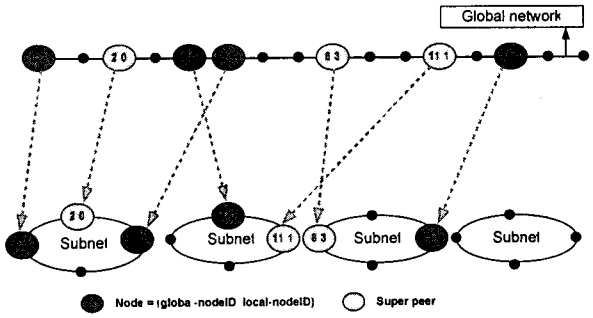

그림 1. TB-Chord의 구조

delD를 구한다. 해쉬 함수는 SHA-1과 같은 해쉬 함수를 사용하며 Global-nodeID는 Global-network에서 데이터를 찾거나 노드가 조인할 때 사용되는 식별자이고, local-nodeID는 각 노드가 속해 있는 subnet에서 데이터를 찾고 노드가 조인할 때 사용되는 식별자이다. Ping을 통해 물리적 거리를 측정하고 Global network과 subnet은 Chord lookup 알고리즘을 사용한다. 그림 1은 크기가 16인 원형 식별자 공간을 가지는 Global network에 각 크기가 4인 원형 식별자 공간을 가지는 4개의 subnet을 가지는 시스템의 구조이다. 각 노드는 global-network에 속해 있으며 또한 4개의 subnet 중 한 개의 subnet에 속해 있다. 각 subnet은 반드시 한 개의 Super peer를 가지며 각 노드는 Global network를 위한 finger table과 노드가 fail되었을 때 시스템을 복원하기 위해서 successor list의 정보를 유지한다. 또한 노드는 자신이 속해 있는 subnet에서 데이터 탐색을 위해서 필요한 finger table과 successor list의 정보를 유지한다. 2개의 finger table은 데이터를 탐색하거나 삽입 할 때 독립적으로 사용한다. Super peer는 자신과 이웃하는 다른 subnet의 Super peer의 정보를 유지한다.

시스템에 노드가 조인할 때 새로운 노드는 반드시 Super peer의 정보를 알고 있다. 시스템에 조인을 원하는 노드는 자신의 IP주소를 2개의 해쉬 함수(h1, h2)를 이용하여 global-nodeID와 local-nodeID를 구한다. 먼저 global-nodeID를 사용하여 Super peer를 통해 Global network에 global-nodeID와 일치하는 위치에 조인하다. 다음으로 Super peer와 물리적 거리를 측정한 값이 threshold 값보다 작으면 그 super peer가 속해 있는 subnet에 local-nodeID와 일치하는 위치에 조인한다. 새로운 노드가 조인되면 자신의 포함되어 있는 subnet의 super peer에서 자신의 상태정보를 보낸다.

데이터의 삽입을 원하는 노드는 데이터의 key를 2개의 해쉬 함수(h1, h2)를 이용하여 해쉬함으로써 global-nodeID와 local-nodeID를 구한다. 노드는 먼저 local-nodeID를 이용하여 자신의subnet에서 local-nodeID를 관리하는 노드에 데이터를 삽입한다. 그리고 Global-nodeID를 이용하여 Global network에서 해쉬한 global-nodeID를 관리하는 노드에게 데이터를 삽입한다.

데이터를 탐색할 때도 데이터의 key를 2개의 해쉬 함수로 해쉬하여 global-nodeID와 local-nodeID를 구한다. 먼저 데이터 탐색을 원하는 노드는 local-nodeID를 이용

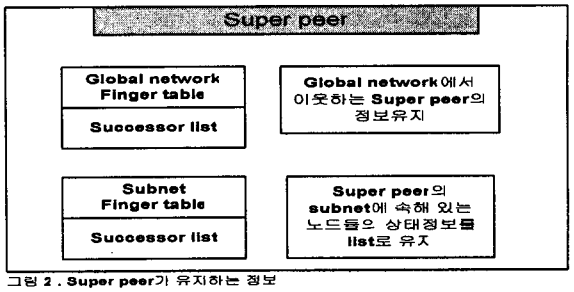

그림 2. Super peer가 유지하는 정보

하여 자신이 속해 있는 subnet에서 local-nodeID를 관리하는 노드를 찾는다. 만약 원하는 데이터가 있으면 local-nodeID를 관리하는 노드에게서 데이터를 받는다. 자신의 subnet에서 원하는 데이터가 없을 때는 global-nodeID를 이용하여 Global network에서 global-nodeID를 관리하는 노드를 찾는다. 그리고 자신의 subnet에서 local-nodeID를 관리하는 노드에게 데이터를 복사한다.

3. 복제 기법

데이터를 탐색할 때 원하는 데이터가 자신의 subnet에 없으면 Global network에서 데이터를 검색한 후 자신의 subnet으로 데이터를 복사하게 된다. Subnet에서 데이터의 인기도가 떨어지면 자신의 subnet에서 데이터 사본은 불필요한 자원이 되므로 그 데이터를 계속 유지하면 저장공간의 낭비를 가져온다. 또한 노드가 떠날 때 자신의 데이터를 자신의 successor에게 보내고 떠나기 때문에 사본이 많을수록 데이터의 전송에 대한 네트워크 부하가 생길 수 있다. 모든 노드는 데이터의 key를 해쉬하여 global-nodeID와 local-nodeID를 대한 데이터를 독립적으로 관리하고 local-nodeID에 대한 데이터들은 모두 사본이라 할 수 있다.

Local-nodeID	File name(Key)	Count
3	A.mp3	2
	T.mp3	3
	Cha.hwp	5

표 1. subnet에서 local-nodeID 3에서 데이터의 사본

저장공간(storage)과 네트워크의 효율성을 위해 데이터의 key에 count를 두어 인기도에 따라 데이터를 관리한다. 데이터가 처음 자신의 subnet에 삽입될 때 count를 Max로 두며 Max는 count의 최고 값이다. 데이터에 대한 request 요청이 있으면 count를 하나 증가 시킨다. 일정한 시간 동안 데이터에 대한 request 요청이 없으면 count를 하나 감소시킨다. Count가 0이 되면 해당 데이터를 삭제한다. 표 1는 Local-nodeID가 3인 노드가 관리하는 데이터의 사본을 나타낸 것이다.

4. 노드 복구

Super peer는 일반적인 노드가 유지하는 routing 정보와 추가적인 정보를 더 유지한다. 노드 조인을 위해 Global network상에서 서로 이웃하는 Super peer의 정보를 유지하고 자신의 subnet에 존재하는 노드들의 상태 정보(컴

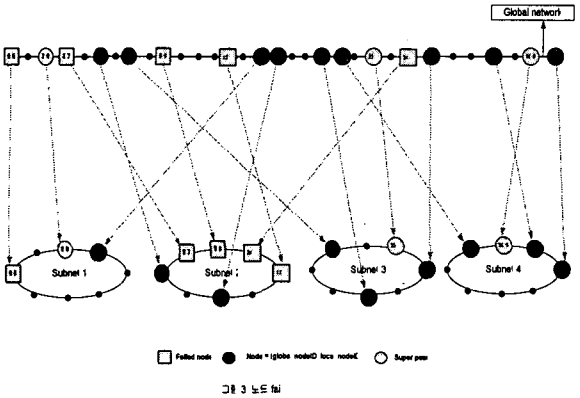

그림 3 노드 fail

퓨팅 파워, 대역폭)를 M 개 만큼 list로 유지한다. 노드가 조인 할 때 자신의 상태정보를 subnet의 Super peer에게 보낸다. Super peer가 새로 조인한 노드의 상태 정보를 받으면 자신의 list에 속해 있는 노드들의 정보와 비교하여 list를 갱신한다. List에 속해 있는 노드들은 다음 Super peer가 될 수 있는 후보들이다. 또한 list에 속해 있는 모든 노드는 Super가 유지하는 list 정보들을 유지한다. 그림 2는 Super peer가 유지하는 정보를 나타낸 것이다.

4.1 노드 leave

일반 노드가 떠나는 방법은 Chord 알고리즘과 같고 자신의 Super peer에게 leave 메시지를 보낸다. 메시지를 받은 Super peer는 자신이 유지하는 list를 검색하여 list에 등록되어 있는 노드들 중 떠나는 노드의 정보가 있으면 해당 노드의 정보를 삭제한다. 또한 list에 속해 있는 모든 노드에게 알려 list 정보를 갱신한다.

Super peer가 떠날 때는 list에 속해 있는 노드 중 가장 성능이 좋은 노드가 다음 Super peer가 되며 자신의 subnet에 모든 노드와 global network의 서로 이웃하는 Super peer에게 메시지를 보낸다. 메시지를 받은 노드는 자신의 정보를 갱신한다.

4.2 노드 fail

어떤 노드에 fail이 발생하면 각 노드는 Global network와 subnet의 successor list를 이용해서 의해서 시스템을 회복한다. 여러 개의 노드가 동시에 fail되면 노드가 유지하는 successor list로 회복이 불가능 할 수 있다. 그림 3은 크기가 32인 원형 식별자 공간을 가지는 Global network와 크기가 8인 원형 식별자 공간을 가지는 4개의 subnet로 이루어진 시스템이다. Successor list의 크기는 log n 이므로 그림 3의 시스템에서 Golobal network의 successor list의 크기는 5이며 subnet은 3 이다. 그림 3에서 노드 5개가 동시에 fail 되면 Global network와 subnet1의 노드들은 successor list를 이용해서 정보를 복구하여 시스템이 회복 되지만 subnet2의 노드들은 successor list를 이용해도 시스템이 회복되지 않으므로 시스템 성능이 떨어지게 된다. 이러한 문제점을 해결하기 위해 rejoin 메카니즘을 사용한다. 각 노드는 subnet의 라우팅 정보와 Global network의 라우팅 정보를 유지하므로 Global network의 successor 노드에게 successor가 속해 있는 subnet의 super peer의 정보를 받고 그 super peer를 통해서 rejoin 한다. 노드가 rejoin할 때 Global network의 정보와 Global-nodeID에 대한 원본 데이터의 이동이 필요없고, local - nodeID에 대한 데이터들의 사본만 삭제한다. Rejoin 과정에서 자신의 subnet만 다시 형성하므로 데이터의 이동에 대한 비용을 줄이면서 시스템을 회복할 수 있다.

5. 결 론

현재 잘 알려진 P2P 시스템은 topology를 고려하지 않으므로 데이터를 삽입하거나 탐색할 때 물리적으로 여러 라우터를 경유하므로 물리적인 latency가 매우 클 수 있다. 그것은 데이터를 lookup할 때 성능 저하의 요인이 될 수 있다. 기존의 Topology를 고려한 시스템들은 계층적 구조를 형성하므로 super-network leader들의 부담이 매우 크며 leader에 fail이 발생했을 때 leader의 교체 비용이 매우 크다. 우리가 제안하는 TB-Chord는 peer의 능력을 고려하여 super peer를 결절하고 교체하여 super peer가 fail되거나 leave될 확률을 줄인다. 노드는 데이터 탐색과 삽입할 때 자신의 subnet에 데이터의 사본을 저장한다. Global network에서 탐색 횟수가 많을수록 사본의 양도 증가하므로 저장공간의 낭비를 초래하게 된다. 또한 노드가 시스템을 떠날 때 떠나는 노드의 successor가 그 데이터를 관리하기 때문에 데이터 이동에 대한 네트워크의 부하를 증가 시키는 원인이 된다. 우리는 데이터의 인기도에 따라 count를 두어 인기가 없으면 자신의 subnet에서 데이터를 사본을 삭제하게 하여 저장공간의 낭비를 줄일 수 있는 방법을 제안하였다. 그리고 동시에 많은 노드가 fail 되였을 때 rejoin 메카니즘을 사용하여 시스템 복구에 대한 큰 비용이 없이 시스템을 회복 할 수 있게 하였다.

참 고 문 헌

[1] Napster. http://www.napster.com

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

[3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A Scalable Content-Addressable Network," In Proceedings of SIGCOMM (August 2001), ACM.

[4] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek and H.Balakrkshnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," In Proceedings of SIGCOMM (August 2001)

[5] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peerto-peer Systems," In Proceedings of IFIP/ACM Middleware 2001 (November 2001).

[6] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing," Tech. Rep. UCB/CSD-01-1141, UC Berkeley, EECS,2001.

[7] B. Yang and H. Garcia-Molina, " Designing a super-peer network," Tech. Rep. 2002-13, Stanford University, 2002.