인과적 순서 전달을 보장하는 전염형 그룹 통신 알고리즘

김 차 영[†] · 안 진 호[††]

요 약

소·중규모 분산 시스템에서 기결정된 메시지 순서화 속성들을 만족시키기 위한 많은 신뢰성 있는 그룹통신 알고리즘들이 제안되었다. 그러나, 엄격한 신뢰성을 보장해야 하는 기존 알고리즘들은 대규모 시스템에 적합하지 않을 수 있다. 이러한 문제를 해결하기 위하여, 기존 알고리즘에 비해 합리적으로 보다 약한 신뢰성을 보장하는 동시에 확장성을 매우 향상시키기 위한 전염형 그룹통신 알고리즘들이 제안되었다. 이러한 알고리즘들은 모두 원자적 메시지 순서 전달 속성을 보장하도록 설계되었다. 그러나, 멀티미디어 시스템 및 협력 작업과 같은 분산 애플리케이션들이 보다 약한 메시지 순서 전달 속성인 인과적 순서 전달만을 요구할 수 있다. 따라서, 본 논문에서는 전염형 기법의 고유한 확장성을 유지하면서, 인과적 순서 전달을 보장하는 효율적인 전염형 그룹 통신 알고리즘을 제안한다.

Epidemic-Style Group Communication Algorithm ensuring Causal Order Delivery

Chayoung Kim[†] · Jinho Ahn[††]

ABSTRACT

Many reliable group communication algorithms were presented to satisfy predetermined message ordering properties in small or medium-scale distributed systems. However, the previous algorithms with their strong reliability properties may be unappropriate for large-scale systems. To address this issue, some epidemic-style group communication algorithms were proposed for considerably improving scalability while guaranteeing the reasonably weaker reliability property than the existing ones. The algorithms are all designed for ensuring the atomic order message delivery property. But, some distributed applications such as multimedia systems and collaborative work, may require only the weaker message ordering property, i.e., causal order delivery. This paper proposes an efficient epidemic-style group communication algorithm ensuring causal order delivery to provide the indigenous scalability of the epidemic-style approach.

키워드 : 분산시스템(distributed system), 그룹통신(group communication), 전염형 알고리즘(epidemic-style algorithm), 확장성(scalability), 메시지 전달 순서(message delivery order)

1. 서 론

비동기적 분산 시스템에서, 한 프로세스 그룹의 멤버들이 협업하여 특정된 분산 컴퓨팅을 수행하기 위해서는 그룹 통신 기법을 사용한다[5]. 특히, 신뢰성 있는 그룹 통신은 메시지 전달 순서에 대한 제약 조건과 원자성(atomicity)을 보장함으로써 메시지 전달 지연 및 손실 시 발생하는 일관성 문제를 해결할 수 있다. 신뢰성 있는 그룹 통신에서 메시지들 간의 전달 순서를 정하는 데에는 두 가지 방법이 있다. 첫 번째, 원자적 순서 브로드캐스트는 모든 메시지를 하나의 일관된 전체 순서로 전달하는 것을 보장해야 하기 때문에 투표형 프로토콜을 구현하는 데 유용하다. 두 번째, 인과적 순서 브로드캐스트는 임의의 두 메시지가 인과적 관계를 가지고 동일한 목적지로 전달되는 경우, 그 두 메시지는 송신 순서로 해당 애플리케이션에게 전달되도록 하는 것을 보장한다. 따라서 인과적 순서 브로드캐스트 기법은 최소한의 메시지 전달 지연을 발생시킴으로써 원자적 순서 기법에 비해 동기화 비용이 매우 적다[1,2].

전통적인 신뢰성 있는 그룹 통신 알고리즘들은 엄격한 신뢰성 보장 때문에, 인터넷과 같은 초대형 시스템에서 확장적이지 못하다[3,4,7,8,10,11,14,16,20]. 이러한 문제점들을 해결하기 위하여, 피투피(peer to peer) 상호 작용 모델의 몇몇 브로드캐스트 알고리즘들이 제안되었다[17]. 첫 번째, SRM (Scalable Reliable Multicast Protocol)[9] 혹은, RMTP (Reliable Message Transport Protocol)[19]는 메시지 손실 혹은 노드 고장을 극복할 수 있는 IP 네트워크 계층에서의 베스트 에포트형(best-effort)인 확장적 멀티캐스트 알고리즘이다. 그러나, 멤버쉽 추적은 네트워크 레벨에서 주요한 논

※ 이 논문은 2004년도 한국학술진흥재단의 지원에 의하여 연구되었음. (KRF-2004-003-D00281)
† 준 회 원 : 고려대학교 컴퓨터학과 박사과정
†† 정 회 원 : 경기대학교 전자계산학과 조교수 (주저자)
논문접수 : 2005년 1월 3일, 심사완료 : 2005년 4월 1일

점으로 남아있고, IP 멀티캐스트가 범용적으로 배치되어 있지 않다는 것이 적용을 용이하지 못하게 한다. 결과적으로, SCRIBE[4], CAN-multicast[21]와 같은 응용 계층 멀티캐스트 알고리즘들이 좋은 대안으로써 최근에 대두되고 있다. 그러나, 이러한 알고리즘들은 자신의 특정된 대규모 피투피 라우팅 구조를 반드시 기반 해야 하기 때문에 범용적이지 못하다[10,11].

이에 비해, 전염형 알고리즘들은 그러한 하부 구조를 요구하지 않으면서, 대규모 그룹에 확장적이며 배치가 쉽고, 노드 고장 혹은 메시지 손실율이 증가함에 따라 성능을 급속하게 감소시키지 않는다[16]. 이 알고리즘들은 전염형 전파 방법을 사용하여 엄격한 신뢰성 조건을 합리적으로 약화시킴으로써 메시지 전달에 대한 확장성을 매우 향상 시킨다. 이러한 확장성은 피투피 모델에 의해서도 보장이 되는데, 참여하는 모든 노드들이 부하(load)를 분산하여 실행하기 때문이다. 또한 중복된 메시지들로 높은 신뢰성도 제공한다. 전염형 알고리즘에서는 한 노드가 메시지를 발생하는 경우, 해당 메시지를 다른 노드들 중, 랜덤하게 선택한 일부의 노드들에게만 전송을 한다. 임의의 노드가 처음으로 그 메시지를 수신하였을 때, 위와 동일한 과정을 수행한다. 이 과정에서, 선택된 일부의 노드들을 가쉽 타겟(gossip target)이라 하고, 그 수를 팬-아웃(fan-out)이라고 한다. 이는 신뢰성 정도를 판단하는 중요한 요소이다[7,16]. 메시지들 간의 전체 전달 속성을 가지는 많은 전염형 알고리즘들이 제안되었고, 팬-아웃과 가쉽의 타켓을 선택하는 방법에 있어서 모두 다르다[11,15]. 그러나, 메시지들 간의 전체 순서 전달 속성 보다 적은 비용을 발생시키는 인과적 순서 전달 속성을 가지는 전염형 알고리즘은 현재까지 제안되어 있지 않다. 특히, 인과적 순서 전달 속성은 비디오-컨퍼런싱, 다중 측(multi-party) 게임과 같은 많은 분산 애플리케이션을 위해 필수적이다[5,20]. 본 논문에서는, 피투피 상호 작용 모델의 고유한 확장성을 유지하면서, 신뢰성 있는 인과적 순서 전달 조건을 보장하는 전염형 브로드캐스트 알고리즘을 제안한다.

본 논문은 다음과 같이 구성되어 있다. 2절과 3절에서는 본 논문에서의 시스템 모델과 전염형 인과적 순서 브로드캐스트 문제를 해결하는 알고리즘을 설명하고, 4절과 5절에서는 제안한 알고리즘의 성능평가와 관련연구에 대해서 논의하고, 마지막으로 6절에서 결론을 맺는다.

2. 시스템 모델

시스템은 $P=\{p_1, \ldots, p_n\}$의 유한개의 프로세스들의 집합으로 구성되어 있다. 이러한 프로세스들은 모두 점대점으로(point-to-point) 네트워크에 의해 연결되어 있으며 메시지를 교환하는 것에 의해서만 통신한다. 프로세스들은 유일하고 전체적으로 순서를 정할 수 있는 식별자를 가지고 있으며, 확률을 정할 수 있도록, 가중치가 있는 코인(weighted coin)을 전달한다. 시스템은 현재의 라운드에서 송신되고, 다음 라운드에서 전달이 될 수 있는 일련의 연속적인 라운드로 실행을 한다. 이 실행 과정에서 시스템은 확률적인 두 가지의 고장을 가정하는데, 첫 번째는 프로세스 고장이다. 이것은 유한한 실행과정 동안 독립적으로 τ의 확률로 프로세스가 고장 난다는 것이다. 두 번째는 메시지 전송 실패이고 결함이 없는 프로세스들 사이에서 메시지 전송 실패는 독립적으로 ε의 확률로 일어난다. 그리고, 메시지 전송 실패와 프로세스 고장 또한 서로 독립적으로 발생한다고 가정하며, 고의적 고장이나 위조 또는 변조된 메시지는 없고, 프로세스 복구 또한 포함하지 않는다. 여기에서 τ와 ε는 매우 작은 확률이다. 프로세스들은 send(m)과 receive(m)을 사용하여 통신하는데, 통신 링크는 다음과 같이 공평하게 손실(fair-lossy)이 일어날 수 있다.

- 만약, 프로세스 p가 q에게 메시지 m을 무한한 횟수로 보낸다면, q는 p로부터 무한한 횟수로 메시지 m을 받을 수 있다.
- 만약, 프로세스 p가 유한한 횟수로 q에게 메시지 m을 보낸다면, q는 유한한 회수로 메시지 m을 p로부터 받을 수 있다.
- 만약, 프로세스 q가 p로부터 시점 t에 메시지 m을 받는다면, p는 시점 t 이전에 메시지 m을 보낸 것이다.

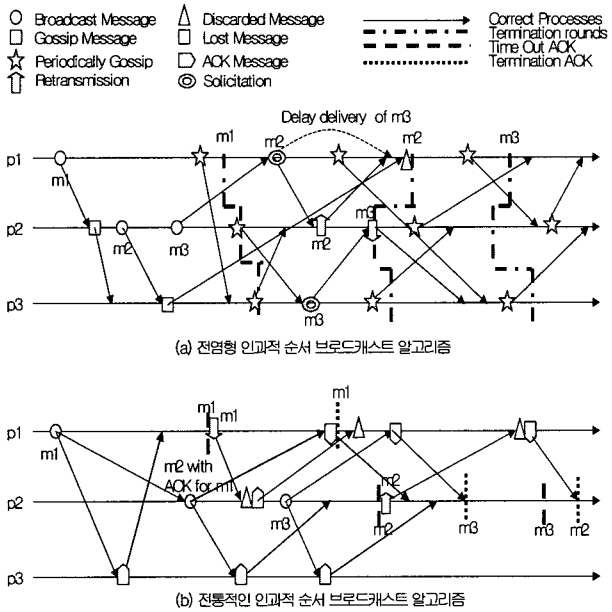
비록 손실이 많이 생기는 링크라고 하더라도, 정상적으로 수행하고 있는 프로세스들은 주기적으로 메시지를 재송신하여 손실 많은 통신 링크 위에서 신뢰성 있는 통신 링크를 만들 수 있다. 만약, 정상적으로 수행하는 프로세스 p가 q에게 메시지 m을 여러 번 송신한다면, 결국 q는 p로부터 m을 수신할 수 있다.

3. 전염형 인과적 순서 브로드캐스트 알고리즘

3.1 기본 개념

각각의 프로세스들은 전염형으로 인과적 순서 전달을 보장하는 브로드캐스트(ECCast) 알고리즘을 일련의 연속된 라운드로 실행을 한다. 모든 프로세스들은 고정된 최대의 라운드인 R까지 전염형으로 통신을 하는 알고리즘, 즉, 가쉽핑을 한다. 가쉽핑을 할 때마다 R은 1씩 감소되며, R이 0이 된 후, 해당 브로드캐스트 메시지는 시스템으로부터 가비지 콜렉션 된다. 프로세스 p가 현재 라운드에서 하나의 메시지를 브로드캐스트한 후, 다음 메시지를 브로드캐스트하기 원한다면, 첫 번째 메시지의 최대 가쉽 라운드가 끝날 때까지 기다리지 않고, 계속해서 다음 메시지를 브로드캐스트할 수 있다. 메시지들은 그들의 벡터 타임스탬프에 의해 목적지에서 인과적 순서로 응용 층에 수신 즉시 전달된다. 그러나, 수신된 메시지의 벡터 타임스탬프 값보다 적은 메시지들이 수신되지 않아서 해당 메시지가 응용 층으로 전달될 수 없기 때문에 버퍼에 유지되는 경우가 발생할 수 있다. 그렇지만, ECCast는 메시지의 전체 순서가 아니라 인과적 순서만

을 보장하기 때문에, 매우 적은 전달 지연을 발생시킨다. 따라서, 인과관계가 없는 동시적인 메시지들이 수신되는 경우, ECCast는 수신된 메시지들의 전달을 지연시키지 않는다. 메시지가 처음 수신된 후, 프로세스들은 그 메시지의 송신시간과 최대 가쉽핑 라운드 시간을 더하여 새로운 시간을 설정한다. 그리고, ECCast도 [3]과 같이, 각 프로세스가 브로드캐스트 그룹 내의 다른 모든 프로세스들에 대해 알고 있다고 가정한다. 한 메시지가 그룹 내의 다른 모든 멤버들에게로 브로드캐스트될 때, 그 메시지에 피기백되는 벡터 타임스탬프의 크기는 멤버의 수가 된다.

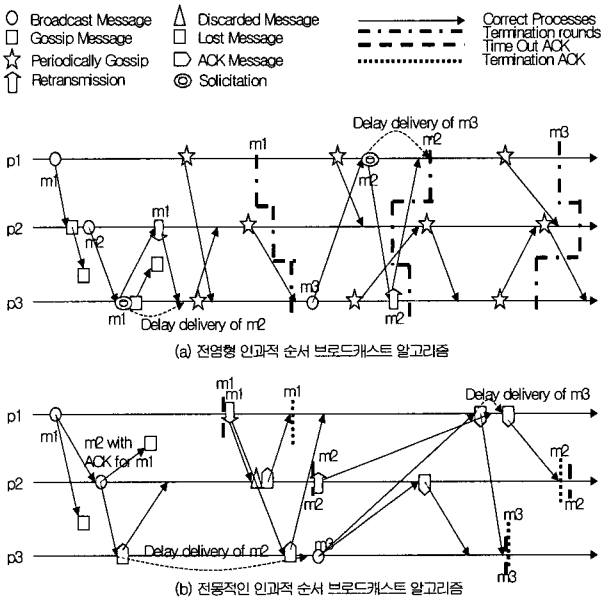


(그림 1) 프로세스 고장이나 메시지 전달 손실이 발생하지 않은 예

(그림 1)은 고장 난 프로세스나 손실된 메시지가 없는 (a) ECCast 알고리즘과 (b)전통적인 인과적 순서 알고리즘 실행의 예를 보이고 있다. (그림 1)의 (a)에서 프로세스 p1은 메시지 m1을 만들고, 랜덤하게 선택된 다른 노드들의 일부분 {p2}에게 m1을 브로드캐스트 한다. p2가 처음으로 m1을 수신하였을 때, p3에게 그 메시지를 가쉽 한다. 그리고, p2는 m2를 만들어 p3에게 브로드캐스트 한다. 메시지 m2를 받은 프로세스 p3은 랜덤하게 선택된 p1에게 그 메시지를 가쉽 한다. 프로세스 p1이 메시지 m2를 받지 못한 상태에서 p2가 다음 메시지 m3을 만들어서, 가쉽-다이제스트 정보와 함께 p1에게 브로드캐스트 한다. 이 정보에 의해, p1은 메시지 m2를 아직 받지 못했음을 알 수 있다. 이때, p1이 m2의 송신자인 p2에게 다시 보내줄 것을 요청하는 것이 아니라, 메시지 m2에 대한 다이제스트 정보를 송신해 준 프로세스 p2에게 재전송을 요청한다. 즉, 프로세스 p2는 메시지 m2에 대한 최초의 송신자로써 m2를 재전송하는 것이 아니라, 다이제스트 정보를 가쉽한 p1에 이웃하는 프로세스로써 m2를 재전송한다. 이와같이, ECCast 알고리즘은 이웃하는 노드에게 높은 가중치를 두는 전염형 알고리즘의 특성을 이용하여 최초의 메시지 송신자보다 거리의 측면에서 가까이에 있는 프로세스에게 재전송 요청을 함으로써 높은 확장성을 보장할 수 있다.

한편, (그림 1)의 (b)전통적인 인과적 순서 알고리즘의 경우, 최초의 메시지 송신자는 다른 모든 프로세스들로부터 해당 메시지의 ACK을 받은 후에 그 메시지에 대한 신뢰를 보장할 수 있다. 그래서, 프로세스 p1이 송신한 메시지 m1에 대한 p2의 ACK 전송 지연으로, 응답 타임아웃이 되었기 때문에, p1은 p2에게 메시지 m1을 재전송한다. 마찬가지로, 프로세스 p1이 수신한 m2에 대한 ACK이 m2의 응답 타임아웃을 지나서 지연되었기 때문에, 최초의 송신자였던 프로세스 p2가 재전송을 한다. 이와 같이, 최초의 메시지 생성자가 메시지 전송에 대한 신뢰성을 보장함으로, 한 노드에 네트워크 부하가 집중되는 ACK-implosion 문제가 심각해 질 수 있다.

(그림 2)는 프로세스 p1이 전송한 메시지 m1과 프로세스 p2가 전송한 메시지 m2의 손실이 있을 때, (a)전염형 인과적 순서 알고리즘과 (b)전통적인 인과적 순서 알고리즘의 실행을 보이고 있다. (그림 2) (a)의 ECCast에서는 모든 프로세스들이 고장 나지 않고, 정상적으로 수행하고 있을 때, 주기적으로 받은 다이제스트 정보에 의해 어떻게 손실된 메시지들이 재전송될 수 있는 지를 보여주고 있다. 프로세스 p1이 메시지 m1을 생성하고 랜덤하게 선택한 {p2}에게 m1을 브로드캐스트 한다. 그리고, p2가 m1을 수신하여 p3에게 가쉽 하였지만, 전송 중에 그 메시지가 손실되었다. 한편, p2는 메시지 m1을 수신하자마자 m2를 생성하여 프로세스 p3에게 브로드캐스트 하였다. 그러나, p3이 m2를 수신하여 p1에게 가쉽 할 때, 이 메시지 또한 전송 중에 손실되었다. 그리고, 이와 동시에 p3은 자신이 수신하지 못한 메시지 m1에



(그림 2) 프로세스 p1이 전송한 메시지 m1과 프로세스 p2가 전송한 메시지 m2의 손실이 발생한 예

```
let current_round, buffer = { }, VT = [], delivered = [], msg=(id, vt, round, gossip_count, bool_delivered)
to ECCast(msg) {                                                                    {Task 1}
   VT[P] = VT[P] + 1
   delivered[P] = VT[P]
   msg=(P, VT, current_round, 0, TRUE)
   buffer = buffer ∪ {msg}
   Unreliable_Multicast(msg)
}
to receipt_processing(msg) {                                                        {Task 2}
   buffer = buffer ∪ {msg}
   do_compare(msg)
   if(msg.current_round > 0) do_gossip(msg)
}
on receive Unreliable_Multicast(msg) {                                              {Task 3}
   receipt_processing(msg)
}
on receive Gossip(msg, msg_digest) {                                                {Task 4}
   if (there is not msg in buffer and delivered[msg.id] < msg.vt[msg.id]) then receipt_processing(msg)
   do_solicitation(msg_digest)
}
on receive Digest(msg_digest) {                                                     {Task 5}
   do_solicitation(msg_digest)
}
on receive solicit_retransmission(ID, Count){                                       {Task 6}
   if(there is msg=(ID, Count) in buffer and (current_round-msg.round) < R) then send Forward(msg) to q
}
to do_solicitation(msg_digest) {                                                    {Task 7}
   for msg in msg_digest
      if(there is not msg in buffer and delivered[msg.id] <vt[msg.id]) then
         send solicit_retransmission(msg.id, msg.vt[msg.id]) to q
}
run periodically gossiping {                                                        {Task 8}
   current_round = current_round - 1
   do_digest()
}
to do_gossip(msg) {                                                                 {Task 9}
   foreach p in processes with probability rate(randomly)
      send Gossip(msg, digest(buffer)) to p
   how_often_do_gossip()
}
to do_digest() {                                                                    {Task 10}
   foreach p in processes with probability rate(randomly)
      send Gossip(digest(buffer)) to p
   how_often_do_gossip()
}
how_often_do_gossip() {                                                             {Task 11}
   foreach msg in buffer
      if(msg.gossip_count >= MAX_Gossip_Count ) then garbage_collection(msg)
}
to do_compare(){                                                                    {Task 12}
   foreach msg in buffer
      delivered_ready = true
      for all i∈p do
         if ( i = msg.id and VT[i] < msg.vt[i] )  then delivered_ready = false
      if (delivered_ready = true) then do_deliver(msg)
}
to do_deliver(msg) {                                                                {Task 13}
      deliver(msg)
      delivered[msg.id] = msg.vt[msg.id]
      for all i∈p do
         VT[i] = max (VT[i], msg.vt[i])
}
```

(그림 3) 전염형 인과적 순서 브로드캐스트 알고리즘(ECCast Algorithm)

대한 재전송 요청을 p2에게 하였고, p2는 p3에게 메시지 m1의 최대 가쉽 라운드가 끝나기 전에 재전송을 할 수 있었다. m1이 손실된 (a)의 경우와 마찬가지로, (그림 2)의 (b)에서도 메시지 m1이 손실되어 p3가 수신하지 못했을 때, 메시지 m1에 대한 ACK 메시지의 타임아웃이 되면, 신뢰적 전송에 책임이 있는 최초의 메시지 송신자 p1이 메시지 m1을 p3에게 재전송한다. 따라서, p3에서는 메시지 m2를 응용 층으로 전달하는 시점이 m2의 인과 관계에서 바로 앞선 메시지 m1의 수신 시간이 된다. 이 경우를 ECCast와 비교해 보았을 때, (그림 1)의 경우와 마찬가지로, 거리상 가까이 있는 프로세스들이 메시지를 송신함으로써, 단위시간당 처리하는 메시지의 양이 전통적인 인과적 순서 알고리즘보다 높다.

그리고, 프로세스 p1이 m2를 수신하지 못한 경우에도, (그림 2) (a)의 ECCast에서는 p3이 m3을 생성하여 p1으로 다이제스트 정보와 함께 m3을 브로드캐스트 했을 때, p1은 수신하지 못한 메시지 m2가 있음을 알 수 있다. 그래서, p1은 p3에게 m2에 대한 재전송을 요청 하고, p3은 즉각 재전송을 한다. 이 과정에서, 메시지 m2의 최초의 송신자인 p2가 재전송을 하는 것이 아니라, p3이 재전송을 함으로써, 거리상 가까운 프로세스가 높은 확률로 재전송을 하는 전염형 알고리즘의 특성을 이용하여, 광범위 네트워크에서도 확장적일 수 있다. 그러나, (그림 2) (b)의 전통적인 인과적 순서 알고리즘에서, 메시지 m1이 손실되었던 경우와 마찬가지로 p1이 m2를 수신하지 못했을 때, 메시지 m2의 최초 송신자 p2가 m2의 ACK 타임아웃이 되면, ACK을 보내지 않은 p1에게 재전송을 하고, p1은 p3으로부터 미리 받은 메시지 m3을 m2가 수신된 후에 응용 층에 전달한다. 따라서, 이 경우도 ECCast와 비교해 보면, 상대적으로 단위시간당 처리하는 메시지의 양이 낮게 된다.

그러므로, 두 그림의 예에서와 같이, 본 논문의 ECCast 알고리즘은 고장 없이 정상적으로 수행하는 모든 프로세스가 복제된 가쉽핑이나 재전송 요청에 의해 전송된 브로드캐스트 메시지를 언젠가는 수신할 수 있도록 함으로써 메시지 전달에 대한 높은 신뢰성을 보장하고 있다. 또한, 전염형 알고리즘의 특성인 이웃하는 프로세스가 높은 확률로 메시지를 송신함으로써, 최초의 메시지 생성자가 송신하는 것에 비해 인터넷과 같은 광범위한 네트워크에서도 높은 확률로 신뢰적일 수 있도록 확장성을 보장한다. 지금까지 설명한 ECCast 알고리즘의 정형적인 기술은 (그림 3)에 있다.

3.2 알고리즘 설명

본 논문의 알고리즘에서, 모든 프로세스가 인과적 순서로 메시지 전달하는 것을 보장하기 위하여, 모든 메시지에 벡터 타임스탬프를 피기백하여 가쉽한다. 그리고, 모든 라운드마다 각각의 프로세스는 브로드캐스트 할 메시지가 있는 경우, 해당 메시지에 첨가하여 자신의 버퍼에 수신하고 있는 메시지들에 대한 다이제스트를 가쉽한다. 브로드캐스트 할 메시지가 없는 경우에는 수신한 메시지들에 대한 다이제스트만을 주기적으로 가쉽한다. 브로드캐스트 메시지를 받았을 경우에도 수신한 메시지를 다이제스트와 함께 가쉽한다. 프로세스 p는 수신하거나 송신한 메시지를 최대 가쉽 라운드까지 자신의 버퍼에 유지한다. 버퍼에는 6개의 필드(id, vt, round, gossip_count, bool_delivered, payload)로 구성된 메시지들이 있다. 이 필드에서, id는 해당 메시지를 최초에 생성한 송신자의 식별자이고, vt는 그 송신자의 벡터 타임스탬프이며, 이것은 해당 메시지에 인과적으로 앞선 메시지의 개수가 얼마인지를 나타낸다. round는 송신 프로세스가 해당 메시지를 몇 번째 라운드에서 브로드캐스트 했는지를 나타내고, gossip_count는 해당 메시지가 몇 번 가쉽핑 되었는지를 나타내며, bool_delivered는 수신 되자마자 응용 층으로 전달되었는지 아닌지를 나타내고, payload는 데이터를 의미한다.

프로세스 p가 메시지 msg를 브로드캐스트 하였을 때, p는 자신의 버퍼에 해당 메시지에 대한 (p, vector, round, gossip_count, bool_delivered, payload)를 삽입한다. 그리고, p는 랜덤하게 일부의 목적지들을 선택하여 msg를 송신한다. 프로세스 p가 최초의 메시지 생성자 q로부터 메시지를 수신했을 때, VT(msg)[k] = VT(p)[k] + 1(만약, k=q 이면) 그리고 $\forall k$:VT(msg)[k]≤VT(p)[k](단, k≠q) 인 경우가 되면, 응용 층으로 즉각 전달한다. 메시지 msg가 응용 층으로 전달되면, VT(p)를 VT(p)[k]=max(VT(p)[k], VT(msg)[k])로 수정한다. 응용 층으로 메시지를 전달한 후, 즉각적으로 버퍼에서 해당메시지를 삭제하는 것은 아니다. 메시지가 버퍼에서 가비지 콜렉션 되는 시점은 해당 메시지의 gossip_count가 최대 가쉽 카운트가 될 때이다. 그리고, 버퍼에는 인과관계가 없는 메시지들이 수신된 시점을 기준으로 정렬되어 있다. 모든 프로세스들은 버퍼에 있는 메시지들을 다이제스트하여, 각 라운드 마다 주기적인 가쉽핑에 의해 서로 송신하고 수신하며, 그 정보들에 의해 수신하지 못한 메시지들은 재전송 요청이 되며, 같은 메시지가 여러 번 수신될 수 있음으로써 신뢰성이 매우 높은 확률로 보장될 수 있다.

(그림 3)에는 전염형 인과적 순서 알고리즘이 있다. 태스크1에서 태스크 13은 각각 동시적으로 실행 될 수 있지만, 한 시점에는 하나의 태스크 인스턴스만이 실행될 수 있다. 모든 태스크 스케쥴러는 공평하고, 모든 태크스는 실행할 기회를 동등하게 부여받는다. 그리고, 각 실행문은 중간에 멈추지 않는다. 프로세스 p가 실행을 시작할 때, p에 대한 벡터 타임스탬프는 0으로 초기화 되고, 가쉽핑하기 위한 메시지를 보관하기 위한 버퍼가 생성된다. 또한, 각 프로세스마다 몇 번째 메시지까지 응용 층으로 전달되었는지 개수를 계산하는 delivered 벡터가 생성된다. 프로세스 p가 메시지를 브로드캐스트할 때, VT[p]는 1씩 증가한다. 그리고, VT[p]는 p가 브로드캐스트 할 메시지에 피기백된다. 이때, 자신의 버퍼에 수신하고 있던 메시지들의 다이제스트도 피기백하여 랜덤하게 선택한 일부의 프로세스들에게 해당 메시지를 송신한다{Task 1}. 각 라운드 마다 모든 프로세스는 주기적으로 자신의 버퍼에 있는 메시지들을 다이제스트하여,

최대 가쉽 카운트가 될 때까지 다이제스트 정보를 송수신한다{Task 8,10,11}. 모든 프로세스들은 주기적으로 통신 링크 채널을 체크하여, 수신할 브로드캐스트 메시지가 있다면, β *|N| 프로세스를 선택하여 가쉽하는데, β는 가쉽핑할 목적지들을 선택하는 확률이다{Task 9}. 프로세스 p가 q로부터 메시지를 받았을 때, 응용 층으로 즉각 전달할 수 있는지 비교해서 전달하고, 그 메시지를 자신의 버퍼에 삽입한다. 응용 층으로 전달하더라도 버퍼에 삽입하는 이유는 최대 가쉽 카운트까지 수신한 메시지에 대한 정보를 다이제스트하여 주기적으로 가쉽핑하기 위해서이다. 그리고, 무결성 조건을 만족하기 위하여, 이미 전달된 메시지들을 기억하는 delivered 벡터를 수정한다{Task 2,3,12,13}. 프로세스 p가 q로부터 다이제스트 정보를 수신하였을 때, 아직 수신하지 못한 메시지를 발견하면, q에게 해당 메시지에 대한 재전송을 요청한다. 그리고, 요청을 받은 프로세스 q는 최대의 가쉽 라운드 시점 내에 재전송 요청이 도착하는 경우, 해당 메시지를 재전송 한다{Task 4,5,6,7}.

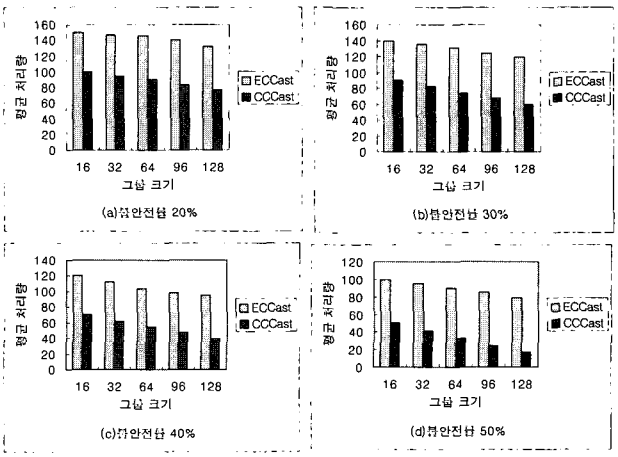
4. 성능 평가

시뮬레이션되는 분산시스템을 구성하는 프로세스들은 각각 하나의 프로세스가 할당되어 비동기적으로 수행한다. 또한 시스템을 구성하는 프로세스들 간의 통신은 UDP/IP를 사용하며, 임의의 두 프로세스 간에 FIFO 메시지 전달 순서를 보장하도록 구현되었다. 하나의 프로세스는 응용 층과 통신 층, 두개의 부분으로 구성된다. 응용 층은 프로세스가 언제 메시지를 전송할지 결정하며, 통신 층은 인과적 순서로 메시지를 전달하는 것을 보장한다. 통신 층에서 메시지를 인과적 순서로 전달하는 것을 구현하고, 응용 층에서 인과적 순서 보장 알고리즘을 실행하기 위한 메시지 패턴을 생성한다. 통신 층에서는 프로세스의 시간을 측정하는데, 프로세스 마다 자신의 시계가 있어서, 실행 초기에 0으로 초기화 한 후, 시계를 라운드 마다 1씩 증가한다. 각 프로세스는 자신의 버퍼 큐를 가지고 있어서 수신한 메시지들을 도착한 순서에 의해 정렬하고 있다. 메시지 구조에는 인과적 순서를 위한 벡터 vector와 송신자가 그 메시지를 전송한 시간 time_stamp 및 데이터 필드 payload가 있다. vector 필드는 인과적 순서 알고리즘[2]에 의해 부여되는 일련의 바이트들이다. time_stamp 필드 값은 지수분포의 평균인데, 통신 층에서 메시지의 전송시간으로 사용한다.

본 논문의 시뮬레이션에서, 어떤 프로세스에서 두개의 메시지를 전송했을 때, 그 두 메시지 사이의 평균 기간(mean inter-message time, MIMT)과 평균 전송 시간(mean transmission time, MTT)은 지수분포를 따른다. MTT는 네트워크의 속도를 측정할 수 있는 기준치이므로, 본 논문의 시뮬레이션에서는 MTT값을 다양하게 변화하여 실험하였다.

또한, 이 실험에서 사용되는 시뮬레이션 파라미터들 중에서, 가쉽핑을 위한 파라미터들은 다음과 같이 pbcast[3,14]와 동일하다. 가쉽핑할 때 일부의 목적지들을 팬 아웃이라고 하는데, 팬아웃의 개수를 b라고 하였을 때, 전송된 메시지를 위한 최대 라운드인 R의 개수는 $\log_b(N)$이다. 그리고, 메시지 전송 실패율은 0.05(5%)로써, 프로세스 고장률은 0.001(0.1%)로써 설정한다. 마지막으로, 팬아웃을 정하기 위한 확률은 랜덤하게 선택한다.

(그림 4)에서는 전체 프로세스들 중 불안전한 상태의 노드들, 즉, 그룹 멤버쉽의 변화를 자주 일으키는 노드들(perturbed processes)이 20~50% 이고, 그룹의 크기가 50~100개 일 때, 평균 메시지 처리 수가 얼마나 달라지는 지 보여주고 있다. (그림 4)의 (a)불안전률 20%를 보면, 전통적인 인과적 순서 알고리즘(CCCast)이 본 논문에서 제시하는 전염형 인과적 순서 알고리즘(ECCast)보다 단위 시간당 처리하는 메시지의 수가 더 낮음을 알 수 있다. 그리고, (a)에서 (d)까지 불안전률이 증가하면서, 그룹의 크기가 커질 때, ECCast와 CCCast를 비교해 보면, 그룹의 크기가 커지더라도, ECCast 알고리즘은 불안전한 상태의 노드들이 점점 증가하는데 따른 영향을 적게 받아서, 메시지 처리율이 급격히 떨어지지 않는 반면, CCCast 알고리즘은 그룹의 크기가 커질수록 불안정한 상태의 노드들에 대한 영향을 더 많이 받아서, 메시지 처리율이 급격하게 떨어짐을 알 수 있다. 따라서, 본 논문에서 제시하는 전염형 알고리즘은 큰 규모의 그룹 크기에서 불안전한 프로세스들이 상당수가 포함되더라도 안정적인 메시지 처리율을 보장함을 알 수 있다.



(그림 4) 불안전한 상태의 프로세스들이 전체의 20~50% 일 때, 그룹의 크기 변화에 따른 단위 시간당 메시지 처리 수

5. 관련 연구

전염형 알고리즘은 전염형으로 통신을 하는 알고리즘이며, 초기에 복제된 데이터베이스의 일관성 관리를 위해 개발되었다[6]. 이후에 고장 발견 기술[22], 가비지 콜렉션[12], 대표자 선출 알고리즘[13]등의 분야에서도 성능을 높이기 위해 사용되고 있다. pbcast[3,14] 알고리즘은 대규모의 네트워크에서 신뢰성 있는 브로드캐스트를 위해 가쉽을 사용한다. pbcast에서, 메시지를 전송하는 경우, IP 멀티캐스트를 사용

하거나, 그렇지 않다면, 랜덤하게 생성된 멀티캐스트 트리를 사용하여 브로드캐스트 한다. 그 이후에, 각각의 노드는 주기적으로 랜덤하게 프로세스들의 일부를 선택하여 최근에 받은 메시지들의 다이제스트를 보낸다. 이러한 정보를 수신한 후, 프로세스는 수신하지 못한 메시지를 체크하여 필요하다면, 재전송을 요청한다.

Directional Gossip[17]은 특별히 넓은 지역의 네트워크를 대상으로 하는 알고리즘이다. 네트워크의 위상과 현재 프로세스의 상태를 고려하여, 최적의 선택이 수행된다. 자세히 말하면, 주어진 노드를 연결하는 이웃하는 노드마다 가중치가 계산된다. 노드의 가중치가 높아질수록, 다른 어떤 노드에 의해서든지 메시지를 받을 확률은 높아진다. 이 알고리즘은 높은 가중치를 갖는 노드가 낮은 가중치를 갖는 노드에 비해 보다 낮은 확률로 선택되도록 하는 간단한 휴리스틱(heuristic)을 적용함으로써, 중복된 송신 메시지 수를 줄인다. 이 알고리즘은 또한 부분적인 멤버 뷰를 기반으로 하는데, 각 LAN마다 다른 LAN에 대해 브릿지의 역할을 하는 가쉽 서버가 하나 씩 있어서, 정적인 계층 구조를 만든다. 그러나, 이러한 특징은 가쉽 서버의 고장에 의해 종속된 여러 프로세스들이 시스템의 나머지 부분과 고립되게 할 수 있다.

Felber[8]는 확률적인 안정성과 유효성을 보장하는 원자적 브로드캐스트 기법을 제시하였다. 이 알고리즘은 미리 설정된 고장난 프로세스 수만큼 포용하고, 높은 확률로써 신뢰성 있게 메시지를 전달하며, 그 순서도 보장한다.

[7]에서는 완전한 분산형 저비용 멤버쉽 알고리즘이 제시되었다. 노드들은 주기적으로 그들의 부분적인 뷰에서 랜덤하게 선택된 일부의 다른 노드들에게 마지막 기간 동안 얻은 가입자 집합 정보를 포함시켜 가쉽한다. 이 기법에서는 멤버 뷰의 크기와 가쉽 대상의 멤버수는 미리 고정되어 있다. 따라서, 본 논문의 ECCast는 이러한 멤버쉽 알고리즘을 사용하여 보다 효율적으로 실행될 수 있다.

6. 결 론

본 논문에서는 전염형 인과적 순서 브로드캐스트 알고리즘 ECCast를 제안하였다. ECCast는 피투피(peer to peer) 상호 작용 모델을 기반으로 엄격하지 않지만 합리적인 신뢰성을 보장한다. 인과적 순서를 보장하기 위하여, 프로세스들은 고정된 라운드 동안 전염형으로 메시지와 함께 벡터 타임스탬프를 전파한다. 여기서 메시지에 피기백되는 벡터 타임스탬프의 크기는 그 그룹 내의 멤버 수이다. 따라서, 제안된 알고리즘은 임의의 프로세스가 이러한 메시지를 수신하는 경우, 인과적 순서만 보장된다면 해당 메시지를 즉시 응용계층으로 전달한다. 그러므로, ECCast는 대규모의 피투피 시스템에서 기존의 엄격한 신뢰성 보장 조건을 만족시키는 인과적 순서 알고리즘에 비해 낮은 동기화 오버헤드를 발생시키고, 확장성을 매우 향상 시킨다. 향후에, 현실적인 네트워크를 기반으로 특정한 피투피 시스템 상에서 ECCast 알고리즘을 구현하고자 한다.

참 고 문 헌

[1] K. P. Birman and T. A. Joseph, "Reliable Communication in the Presence of Failures," ACM Transactions on Computer Systems, Vol.5, No.1, pp.47-76, 1987.

[2] K. P. Birman, A. Schiper and P. Stephenson, "Lightweight causal and atomic group multicast," ACM Transactions on Computer Systems, Vol.9, No.3, pp.272-314, 1991.

[3] K. P. Birman, M. Hayden, O. Ozkasap. Z. Xiao, M. Budiu and Y. Minsky, "Bimodal Multicast," ACM Transactions on Computer Systems, Vol.17, No.2, pp.41-88, 1999.

[4] M. Castro, P. Druschel, A-M. Kermarrec and A. Rowstron, "SCRIBE: A Large-Scale and Decentralized Application-Level Multicast Infrastructure," IEEE Journal of Selected Areas in Communications, Vol.20, No.8, Oct. 2002.

[5] F. J. N. Cosquer and P. Verissimo, "Survey of selected groupware applications and supporting platforms," Technical Report RT-21-94, INESC, 1994.

[6] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehar and D. Terry, "Epidemic algorithms for replicated database maintenance," In Proc. of the 6th ACM Symposium on Principles of Distributed Computing, pp.1-12, VanCouver, BC. Canada, Aug. 1987.

[7] P. Eugster, S. Handurukande, R. Guerraoui, A.-M. Kermarrec and P. Kouznetsov, "Lightweight probabilistic broadcast," In Proc. of the international Conference on Dependable Systems and Networks, pp.443-452, 2001.

[8] P. Felber and F. Pedone, "Probabilistic Atomic Broadcast," Hewlett-Packard Technical Report HPL-2002-69, 2002.

[9] S. Floyd, V. Jacobson, C-G. Liu, S. McCanne and L. Zhang, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing," IEEE/ACM Transactions on Networking, pp.784-803, Dec. 1997.

[10] A. J. Ganesh, A.-M. Kermarrec and L. Massoulie, "SCAMP: Peer-to-Peer lightweight membership service for large-scale group communication," In Proc. of the 3rd International Workshop on Networked Group Communications, London, UK., Nov. 2001.

[11] A. J. Ganesh, A.-M. Kermarrec and L. Massoulie, "HiScamp: self-organizing hierarchical membership protocol," In Proc. of the 10th European ACM SIGOPS WorkShop, Sept. 2002.

[12] K. Guo, M. Hayden, R. V. Renesse, W. Vogels and K. P. Birman, "GSGC: An Efficient Gossip-Style Garbage Collection Scheme for Scalable Reliable Multicast," Technical Report TR97-1656, Cornell University, Computer Science, Dec. 1997.

[13] I. Gupta, R van Renesse and K. P. Birman, "A probabilistically

correct leader election protocol for large groups," In Procs of the 14th International Symposium on Distributed Computing, pp.89-103, Toledo, Spain, Oct. 2000.

[14] M. Hayden and K. Birman, "Probabilistic broadcast", Technical Report TR96-1606, Cornell University, Computer Science, Sept. 1996.

[15] D. Kempe and J. Kleinberg, "Protocols and impossibility for gossip-based communication mechanisms," In Proc. of IEEE Symposium on Foundations of Computer Science, pp.417-480, Vancouver, Canada, Nov. 2002.

[16] A.-M. Kermarrec, L. Massoulie and A. J. Ganesh, "Probabilistic reliable dissemination in large-scale systems," IEEE Transactions on Parallel and Distributed Systems, Vol.14, No.3, pp.248-258, Mar. 2003.

[17] M.-J. Lin and K. Marzullo, "Directional gossip:Gossip in a wide-area network," Technical Report CS1999-0622, University of California, San Diego, Computer Science and Engineering, June 1999.

[18] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins and Z. Xu, "Peer-to-Peer Computing," Technical Report HPL-2002-57, HP Laboratories, Palo Alto, Mar. 2002.

[19] S. Paul, K. Sabnani, J. Lin and S. Bhattacharyya, "Reliable Multicast Transport Protocol(RMTP)," IEEE Journal on Selected Areas in Communications, Vol.15, No.3, pp.407-421, Apr. 2000.

[20] D. Pendarakis, S. Shi, D. Verma and M. Waldvogel, "ALMI:An application level multicast infrastructure," In Proc. of the 3rd USNIX Symposium on internet Technologies and Systems, pp.49-60, San Francisco, CA, USA. Mar. 2001.

[21] S. Ratnasamy, M. Handley, R. Karp and S. Shenker, "Application-level multicast using content-addressable networks," In Proc. of the 3rd International Workshop on Networked Group Communication, Nov. 2001.

[22] R. V. Renesse, Y. Minsky and M. Hayden, "A Gossip-Style Failure Detection Service," Technical Report TR98-1687, Cornell University, May 1998.

김 차 영

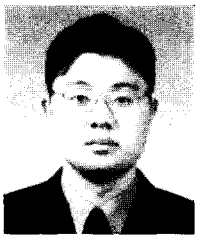
e-mail : chayoung@disys.korea.ac.kr

1996년 숙명여자대학교 전산학과 졸업(학사)

1998년 숙명여자대학교 전산학과 졸업(석사)

2001년~현재 고려대학교 컴퓨터학과(박사과정)

관심분야 : 분산시스템, 그룹통신, 이동 컴퓨팅, p2p 컴퓨팅



안 진 호

e-mail : jhahn@kyonggi.ac.kr

1997년 고려대학교 컴퓨터학과 졸업(이학학사)

1999년 고려대학교 컴퓨터학과 졸업(이학석사)

2003년 고려대학교 컴퓨터학과 졸업(이학박사)

2003년~현 재 경기대학교 정보과학부 전자계산학과 조교수

관심분야 : 결함포용 분산시스템, 이동에이전트 시스템, 그룹통신, p2p 컴퓨팅