분산 시스템에서의 효과적인 코디네이터 선출 알고리즘

김기[1], 이동훈[2], 최은미[1]
[1]한동대학교 정보통신공학과
bart@seed.handong.edu, emchoi@handong.edu
[2]건국대학교 컴퓨터정보통신공학과
podong@konkuk.ac.kr

An Effective Coordinator Election Algorithm in a Distributed System

Ki Kim[1], Donghoon Lee[2], Eunmi Choi[1]
[1]Department of IT, Handong Global University
[2]Department of Computer Science and Engineering, Konkuk University

요 약

분산 시스템 상에서 공유되는 정보와 노드들의 동기화를 위하여 코디네이터를 선출하여 하부작업을 다른 노드에 할당하거나 분산된 노드 관리에 사용한다. 코디네이터 선출 알고리즘은, 분산된 노드들 간의 sing-point-of-failure를 피하여 동적으로 코디네이터를 선출하여 코디네이터 부재 시의 시스템의 상태와 기능을 복귀 시키게 된다. 본 논문에서는 기존의 코디네이터 선출 알고리즘의 문제점을 보안하기 위하여, fail이 존재하는 네트워크상에서 사용 가능한 안정성 있는 효과적인 코디네이터 선출 알고리즘을 제안하고 실험을 통해 코디네이터 선출과 합병에 걸리는 시간을 측정한다.

1. 서론

분산 시스템에서 독립적으로 운영이 되는 노드들 간의 상태 정보나 데이터 정보를 교환하기 위한 통신 방법과 어떻게 각 노드들을 동기화 하고 상호작용 할 것인가 하는 것들은 분산 시스템의 성능을 결정하는 중요한 Issue 들이다[1,2]. 이러한 경우 같은 일을 하는 노드들의 그룹에서 하나의 유일하고 중앙집중적인 코디네이터(Coordinator)를 두어 하부 작업을 다른 노드에 할당하거나 다른 노드들을 관리하는 등의 특별한 일을 담당하도록 한다. 코디네이터를 하나의 노드가 항상 담당하도록 하는 것은 이상상황으로부터의 신뢰성(fault tolerance)의 결여를 가져오게 되므로, 코디네이터에 문제가 발생했을 경우에 그룹의 나머지 노드들이 동적으로 선출 알고리즘(Election Algorithm)을 통하여 다른 가능성이 높은 노드를 코디네이터로 선출하여 시스템의 상태와 기능을 복귀시켜야 한다[3]. 분산 시스템 상에서의 대표적인 코디네이터 선출 알고리즘으로는 Bully Algorithm[4], Invitation Algorithm[4], Ring Algorithm[5]과 같은 알고리즘이 존재한다.

본 논문에서는 신뢰성 있는 네트워크를 바탕으로 하고 있는 Bully Algorithm과 문제가 존재하는 네트워크에서 사용 가능한 Invitation Algorithm을 결합하여 효과적인 코디네이터 선출 알고리즘을 2장에서 제안하고 실험을 통해 선출에 걸리는 시간과 여러 코디네이터들이 하나로 합해지는 시간을 측정한 것을 3장에서 보이도록 하겠다. 4장에서 결론을 맺는다.

2. 제안하는 코디네이터 선출 알고리즘

2.1. 노드의 상태 정보

모든 노드들은 코디네이터 선출을 위해 다음과 같은 상태 정보를 유지한다.

State	{Election, Confirming, Wait_Confirm, Normal_Mode, Coordinator_Mode, Coordinator_Fail} 중의 하나의 상태
Coordinator	자신이 속한 그룹의 Coordinator의 ID
Group	자신이 속한 그룹의 ID
Members	자신이 속한 그룹에 속한 노드 리스트
MyID	자신을 구별할 수 있는 유일한 ID

코디네이터는 그 그룹에 속한 노드들의 생존을 확인하기 위해 다음과 같은 데이터구조를 만들어 관리한다.

attendaceBook	(key:ID, value:failCount)의 Hash Table

2.2. 코디네이터 선출 알고리즘 Pseudo Code

코디네이터와 노드들은 주기적으로 서로에 대해 생존을 확인하게 된다(알고리즘 1~3). 노드들은 코디네이터가 일정한 횟수 이상 응답이 없다면 코디네이터가 fail 되었다고 간주를 하고, 새로운 코디네이터

선출을 시도하게 된다. 반면, 코디네이터는 노드로부터 일정한 횟수 이상 연락을 받지 못하면 노드 fail을 알리게 된다. 또한 fail 상태의 노드가 다시 살아나는 것을 주기적으로 확인한다.

```
coordinatorCheck() {
  if (state == Normal_Mode) {
    send (Coordinator, MyID, AreYouThere)
    wait reply of Coordinator (AYT_reply:reply), Timeout=T {
      if (reply == OK)
        failCount = 0
      else
        failCount++
    } on timeout {
      failCount++
    }
    if (failCount > failThreshold)
      changeState(Normal_Mode, Coordinator_Fail)
      election()  /* 코디네이터 fail 시 election 시작 */
  }
}
```

알고리즘 1. 노드가 코디네이터의 생존을 확인하는 알고리즘

```
respond() {
  while (state == Coordinator_Mode) {
    wait for AreYouThere message
    if (sender ∈ Member) {
      put (attendanceBook, sender, 0)
      send (sender, AYT_reply:OK)
    } else {
      send (sender, AYT_reply:ERROR)
    }
  }
}
```

알고리즘 2. 코디네이터가 생존확인에 대해 응답하는 알고리즘

```
memberCheck() {
  if (state == Coordinator_Mode) {
    for every node except myself in attendanceBook {
      failCount_i++
    }
    for every node except myself in attendanceBook {
      if (failCount_i > failThreshold) {
        alarm (NodeFailed, node_i);
        failCount_i = -1;
      }
    }
    for every node except myself in attendanceBook {
      if (failCount_i < 0)
        findLostMember(node_i);
    }
  }
}
```

알고리즘 3. 코디네이터가 노드들의 생존을 확인하는 알고리즘

코디네이터 선출을 위해 모든 노드들은 상태 정보를 유지한다. 상태정보를 바꿀 때는 가능한 조건인지 확인을 한다(알고리즘 4).

```
changeState(condition, newState) as boolean {
  if (state ∈ condition) {
    state = newState
    return true
  } else {
    return false
  }
}
```

알고리즘 4. 상태정보를 변경하는 알고리즘

코디네이터 부재 시에 노드들이 새로운 코디네이터를 선출 알고리즘은 현재 살아있는 노드들의 ID를 비교해서 가장 우선 순위가 높은 ID를 가진 노드를 새로운 코디네이터로 선출하도록 한다(알고리즘 5).

```
election() {
  if ( changeState(Coordinator_Fail, Election) ) {
    Coordinator = empty
    sortPriority (Members)
    for every node in Members {
      if (node == myself) {
        changeState(Election, Confirming)
        for every node except myself in Members
          /* 내가 선출 과정중임을 다른 노드에게 알림 */
            send (node_i, MyID, WaitConfirm)
        chageState(Confirming, Coordinator_Mode)
        Coordinator = myself
        for every node except myself in Members
          /* 내가 코디네이터임을 알림 */
          send (node_i, MyID, Confirm)
        return
      } else {
        send (ID, MyID, Ping)
        wait reply of node_i (Ping_reply:reply), Timeout=T {
          changeState(Election or Wait_Confirm, Wait_Confirm)
          return
        } on Timeout {
          continue
        }
      }
    }
  }
}
```

알고리즘 5. 새로운 코디네이터를 선출하는 알고리즘

앞에서의 코디네이터와 노드 간의 생존확인 알고리즘에서는 전송되는 메시지의 손실, 버퍼의 오버플로우, 노드의 일시적인 과부하, 네트워크의 분리 등의 문제로 인하여 여러 개의 코디네이터가 생기는 경우가 발생할 수 있다. 이러한 경우 여러 개로 나뉘어진 그룹을 하나로 합병하는 알고리즘과 fail 상태에서 회복한 노드를 다시 그룹에 포함시켜 새로운 코디네이터에게 복속되도록 하는 알고리즘이 필요하다. 여기서는 서로가 알고 있는 코디네이터의 ID를 비교하여 합병의 주도자가 결정된다(알고리즘 6).

```
findLostMember(node) {
  if (state==Coordinator_Mode) {
    send (node, MyID, WhoIsCoordinator:reply)
    wait reply of node (WIC_reply:reply), Timeout=T{
      if (Coordinator > reply) {
        send (node, MyID, MergeWithMe)
      }
    } on timeout {
        /* 상대가 여전히 fail 상태인 경우 */
      return
    }
  }
}
```

알고리즘 6. 나뉘어진 그룹을 합병하거나 failure 상태에서 회복한 노드를 찾아 그룹에 포함시키는 알고리즘

새로운 코디네이터 선출과 그룹합병을 위해 전송되는 메시지를 처리하는 알고리즘은 다음과 같다(알고리즘 7).

```
receive () {
  while (true) {
    wait for message
    case Ping
      send (sender, Ping_reply:OK)
    case WaitConfirm
      changeState (Normal_Mode or Coordinator_Fail or Wait_Confirm,
                                                    Wait_Confirm)
    case Confirm
      Coordinator = sender
      changeState (Wait_Confirm, Normal_Mode)
    case WhoIsCoordinator
      if (state == Wait_Confirm)
        send (sender, WIC_reply:lowestPriority)
      else
        send (sender, WIC_reply:Coordinator)
    case MergeWithMe
      success = changeState (Normal_Mode, or
      Coordinator_Mode or Wait_Confirm, Normal_Mode)
      if (success)
        Coordinator = sender
  }
}
```

알고리즘 7. 선출과 합병을 위한 메시지 처리 알고리즘

2.3. 기존 코디네이터 선출 알고리즘들과의 비교

정상적인 네트워크 상황에서의 선출 알고리즘은 Bully Algorithm을 사용하고 있지만 fail 상태에서 회복한 노드를 포함하는 과정에서 새로운 선출 과정을 거치지 않기 때문에 시간을 줄일 수 있다. 또한 문제가 존재하는 네트워크 상황에서 여러 개로 나뉘어진 그룹을 합치는 방법은 Invitation Algorithm을 사용하지만 코디네이터가 fail되었을 때 각각의 노드가 새로운 그룹을 생성하여 합병하지 않고 통신 가능한 노드끼리 Bully Algorithm을 사용한 선출을 하여 어느 정도 합병된 그룹을 생성하게 되므로 최종적으로 하나의 그룹으로 합병되는 시간을 단축 할 수 있다[4].

3. 실험결과

실험은 코디네이터 fail 발생시 선출 알고리즘을 실행하여 새로운 코디네이터를 선출하는데 걸리는 시간과 네트워크 분리로 인해 여러 개로 나뉜 그룹이 하나로 합쳐질 때까지 걸리는 시간을 측정하였다.

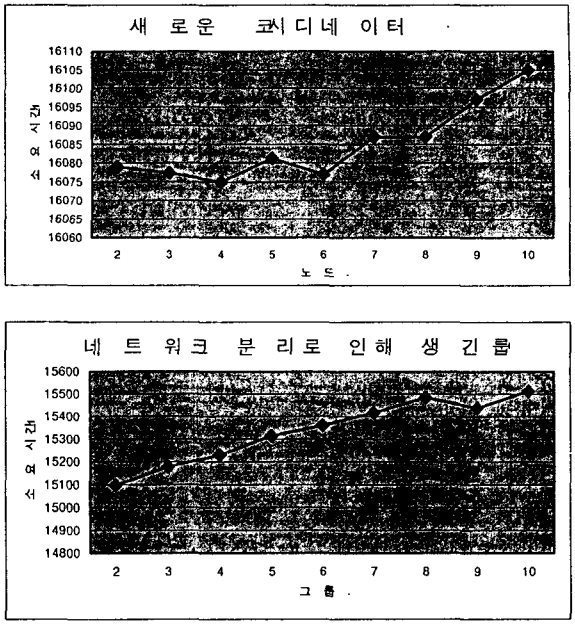

그림 1. 코디네이터 선출 알고리즘 실험결과

4. 결론

본 논문에서는 기존의 코디네이터 선출 알고리즘에서 신뢰성 있는 네트워크를 기반으로 한 Bully 알고리즘과 문제가 존재하는 네트워크에서 사용 가능한 Invitation 알고리즘을 결합하여 보다 안정성이 있는 효과적인 코디네이터 선출 알고리즘을 제안했으며 실험을 통해 선출에 걸리는 시간과 여러 코디네이터들이 하나로 합해지는 시간을 측정하였다. 본 연구의 향후 작업으로 다른 알고리즘들과의 성능 비교와 이상 상황에서의 대처 능력을 비교하도록 하겠다.

참고 문헌

[1] Andrew S. Tanenbarum, Maarten van Steen, "Distributed Systems principles and Paradigms", Prentice Hall, 2002

[2] George Coulouris, Jean Dollimore, Tim Kindberg, "Distributed Systems Concepts and Design", Addison Wesley, 2001

[3] Randy Chow, Theodore Johnson, "Distributed Operating Systems & Algorithms", Addison Wesley, 1997

[4] H. Garcia-Molina, "Elections in a distributed computer system", IEEE Trans. on Computers, C-31(2):48-59, 1982

[5] Chang, E.G. and Roberts, R. "An improved algorithm for decentralized extrema-finding in circular configurations of processors.", Comms. ACM, Vol. 22, No.5, pp.281-3, 1979