COVERS를 이용한 선출 알고리즘의 구현 및 성능 평가

이석형[U] 최 훈
충 남 대 학 교 컴 퓨 터 공 학 과
{shlee,hchoi}@ce.cnu.ac.kr

Implementation and Evaluation of an Election Algorithm using COVERS

Seok-Hyung Lee[U], Hoon Choi
Dept. of Computer Engineering, Chungnam National University

요 약

분산 프로세스 그룹 내의 프로세스들 중에서 조정자를 결정하는 것을 선출(election)한다고 하며 분산 시스템이 가동되면서 최초로 조정자가 정해져야 할 경우, 또는 조정자 프로세스에 장애가 생겨서 새로운 조정자가 필요할 경우에 선출 알고리즘이 수행된다. 본 연구팀은 고속 조정자 선출 알고리즘을 제안한 바 있는데, 이 알고리즘의 검증 시험, 성능 평가를 수행하기 위해서 COVERS 라는 시뮬레이션 툴을 사용하여 분산 네트워크 환경을 구성한 뒤 이 위에 다수 프로세스에 선출 알고리즘을 구현하고 각종 장애 상황을 발생시켜 시험하였다. 본 논문에서는 범용 시뮬레이션 툴인 COVERS를 간단히 설명하고, 이 위에 모델링한 분산 네트워크 환경, 그리고 선출 알고리즘 구현에 대해서 설명하고, 성능 분석 결과를 제시한다.

1. 서론

분산 시스템을 효율적으로 운영하기 위해서는 여러 프로세스들 중에서 상호 배제를 보장하고 공유 자원의 할당과 회수를 관리하는 등의 역할을 수행하는 조정자 프로세스가 필요하다. 일반적으로 시스템의 장애감내성을 높이기 위해서 분산 소프트웨어 실행에 참여하는 모든 프로세스가 조정자 기능을 가지게 하고 이들 중 한 프로세스만이 조정자 역할을 수행하도록 한다. 분산 시스템이 가동되면서 최초로 조정자가 정해져야 할 경우나 조정자 프로세스에 장애가 생겨서 새로운 조정자가 필요할 경우에 선출 알고리즘이 수행된다. 선출 알고리즘의 요체는 프로세스 그룹의 살아있는 프로세스들 중에서 조정자로서 가장 높은 우선순위를 지닌 프로세스를 알아내고, 그 프로세스를 다른 프로세스들에게 알리는 것이다[1].

다양한 선출 알고리즘들이 제안되어 왔는데 그 중에서 대표적인 것이 불리 알고리즘(bully algorithm)과 링 알고리즘(ring algorithm)이다[5,6]. 그러나, 두 알고리즘은 선출에 필요한 메시지 수가 매우 많고 응답시간이 길며, 다양한 장애 상황에 대처하지 못하는 단점이 있다. 따라서, 기존의 알고리즘의 단점을 보완하고 여러 장애 상황에 대처할 수 있는 고속 불리 알고리즘을 개발하였다[2,3].

조정자 선출과 같은 분산 알고리즘을 구현하려면 알고리즘이 실행될 플랫폼, 즉 분산 시스템 환경이 필요하다. 네트워크로 연결된 다수의 호스트 컴퓨터들 각각에 조정자를 필요로 하는 분산 응용 프로세스와 선출 알고리즘 프로세스가 함께 실행되도록 플랫폼을 구성할 수 있다. 그러나 이런 실제 실행 환경은 선출 알고리즘의 정확성을 검증하는데 필요한 다양한 장애 상황을 발생시키는데 적합하지 않다. 따라서 본 논문에서는 모델링 도구인 COVERS를 이용해 네트워크, 호스트, 프로세스 등 분산 시스템 환경의 시뮬레이션 모델을 구성하고 이 모델에서 불리 알고리즘과 고속 불리 알고리즘을 구현하여 실행시킴으로써, 알고리즘이 실제 네트워크의 서로 다른 호스트에서 실행되는 것과 같은 효과를 내도록 한 연구 결과를 소개한다. 구현된 모델에서 다양한 시나리오에 따라 프로세스의 장애, 복구 상황을 시뮬레이션하고, 이 상황에 대처하여 선출 알고리즘이 정확히 동작하는지 시험하였다.

본 논문은 다음과 같이 구성된다. 2장에서는 불리 알고리즘과 고속 불리 알고리즘에 대해서 간단히 설명하고, 3장에서는 모델링 도구인 COVERS에 대해 살펴보고 이를 이용하여 각각의 알고리즘이 구현된 방법을 기술한다. 4장에서는 두 알고리즘에 대한 성능을 비교 분석하고 결론을 맺는다.

2. 관련연구

불리 알고리즘은 조정자 프로세스가 죽은 것을 발견한 프로세스가 자신뿐만 아니라 다른 프로세스들에게도 똑같이 조정자 선출 과정을 시작시키는 방식이다. 살아있는 모든 프로세스가 제각기 불리 알고리즘을 수행하다가, 자신이 가장 높은 우선순위 번호의 프로세스가 아니라는 것을 알게 되면 조용히 기다리면서 새로운 조정자가 선출되어 통보되기를 기다린다[1]. 불리 알고리즘은 장애 감내형이어서 불리 알고리즘을 수행하는 프로세스가 도중에 죽게 되도 알고리즘은 이에 대처할 수 있다. 어떤 프로세스가 죽었다 살아난 경우에도 곧바로 이 알고리즘이 수행되어, 살아난 프로세스가 기존 조정자 보다 더 높은 우선순위 번호를 가진 프로세스일 경우 조정자 역할을 즉시 되찾아온다. 그러나, 시스템을 구성하는 프로세스의 수가 많아지면 선출이 수행되는 시간이 매우 길어지며, 메시지도 많이 발생된다.

이를 보완하여 고속 불리 알고리즘(fast bully algorithm)이 제안되었다[2,3]. 불리 알고리즘에서는 조정자가 죽은 것을 발견한 프로세스가 선출 알고리즘을 먼저 시작하고 election

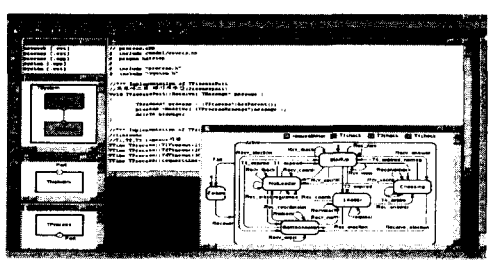
메시지를 보내 다른 프로세스들도 똑같은 과정을 수행하도록 지시한다. 그러나 고속 불리 알고리즘에서는 조정자가 죽은 것을 발견한 프로세스가 주도적인 역할을 한다. 불리 알고리즘에서와 같이 다른 프로세스들에게 선출 과정을 수행하도록 지시할 뿐 아니라, 다른 프로세스들로부터의 응답을 토대로 현재 우선순위 번호가 가장 높은 프로세스를 알아내어 그 프로세스에게 조정자 후보임을 통보한다. 조정자 후보로 통보된 프로세스는 불리 알고리즘에서와 같이 coordinator 메시지를 보내어 선출 과정을 완료한다. 또한, 죽었다 살아난 프로세스가 선출을 시작할 경우에는 IamUp 메시지를 모든 프로세스들에게 보낸 후, 그에 대한 응답 메시지로 그룹에 속한 프로세스의 목록이 담긴 view 메시지를 받음으로써, 죽어 있는 동안에 프로세스가 추가되었는지를 검사하고 새로운 조정자를 인식하게 된다.

고속 불리 알고리즘도 장애 감내형이어서 선출 과정 도중에 임의의 프로세스가 죽게 되어도 알고리즘은 이에 대처할 수 있다. 또한 불리 알고리즘에서와 같이 어떤 프로세스가 죽었다 살아난 경우에도 곧바로 수행되어, 살아난 프로세스가 기존 조정자보다 더 높은 우선순위 번호를 가진 프로세스일 경우 조정자 역할을 즉시 되찾아 온다. 또한, 그룹에 이 프로세스보다 더 높은 우선순위 번호의 프로세스가 들어와 조정자가 되었을 경우, view 메시지에 담긴 프로세스 목록을 보고 새로운 조정자가 추가되었음을 알게된다. 알고리즘이 수행되는 조건도 불리 알고리즘에서의 경우와 동일하다.

3.알고리즘 구현

3.1 COVERS 개요

COVERS는 그래픽 사용자 인터페이스를 갖춘 소프트웨어 실행 환경으로서 객체지향개념에 기반하여 객체에 대한 명세와 데이터 처리, 수행 순서 등은 C++로 구현하는 모델링 툴이다[4]. 특히 분산된 객체간에 메시지를 송수신 할 수 있는 기능을 지원하며, 다양한 비동기적(asynchronous) 병행적(concurrent)인 이벤트들의 발생을 시뮬레이션하고, 이들의 처리 과정을 로깅(logging), 트레이싱(tracing)할 수 있기 때문에, 통신망이나 컴퓨터 시스템, 분산 시스템 등을 모델링하는데 적합하다.



[그림 1] COVERS로 구현된 고속 불리 알고리즘

COVERS는 객체의 구조와 그들 사이의 연결을 표현하는 다이어그램(diagram), 객체의 상태를 정의한 상태도(state chart), 데이터 객체와 함수를 표현하기 위한 C++ 코드로 구성되어 있으며, MS-Windows 기반의 그래픽 환경을 제공한다.

3.2 알고리즘 설계 및 구현

COVERS를 이용하여 분산 시스템을 구현하기 위해서 우선 Component를 구성한다. Component는 크게 분산 시스템을 표현한 TSystem과 시스템 구성 요소인 TNetwork, TProcess로 이루어진다.

3.2.1 TSystem

TSystem에서 분산된 호스트들에서 수행될 프로세스의 개수를 지정하며, 각 프로세스의 상태와 시스템의 상태를 파악할 수 있다. 또한 알고리즘의 정확성을 그래픽으로 증명하기 위한 루틴도 포함된다. 만약 그룹의 상태가 선출 상태에 있으면 COVERS 내부 함수 GetState()를 이용하여 프로세스들의 상태를 얻어올 수 있으며 이를 보고 선출이 시작된 시점과 끝나는 시점을 알 수 있다. 또 GetClock()함수로 선출이 시작되는 시점과 끝나는 시점의 시간을 얻어 올 수 있는데 이 두 시점의 시간차를 계산하여 알고리즘의 응답 시간을 구할 수 있다. 또한 선출 과정에서 각 프로세스가 사용하는 메시지의 수는 메시지를 보낼 때마다 카운터를 증가시키고, 선출이 끝나는 시점에서 선출에 쓰인 메시지의 카운터 값을 모두 더해 선출 동안에 사용된 메시지의 수도 알 수 있다.

3.2.2 TNetwork

TNetwork은 프로세스간에 주고 받는 메시지의 전송 방식과 속도를 나타내며 COVERS 내부에서 제공하는 포트와 타이머를 기반으로 구현되었다. [그림 2]에서와 같이 TNetworkTimer 클래스에서는 네트워크를 통해 전달되는 메시지의 전송 지연 시간을 나타냈다. 모델에서 어떤 프로세스가 다른 프로세스로 메시지를 보내기 위해서는 먼저 네트워크로 메시지를 보낸다. 네트워크에서는 메시지를 받자마자 전송 지연 시간 타이머를 작동시키고 받은 메시지를 큐에 저장한다. 그래서 지연 시간 타이머가 종료되었을 경우 메시지를 다른 프로세스들에게 전송하여 실제로 프로세스간에 메시지 전송 지연시간이 존재하는 효과를 내도록 하였다. 전송 지연 시간은 TConstDistr() 함수를 쓰는데 이번 연구에서는 지연 시간이 deterministic distribution을 따른다고 가정했다.

TNetworkTimer 클래스의 멤버 함수로 Expire()가 있는데 이 함수는 전송 지연 타이머 작동이 종료되었을 경우 호출되며 네트워크에서 프로세스로 메시지 전송을 수행하는 역할을 한다. Expire() 함수가 호출될 때 수행되는 Send(Message)는 COVERS의 Port 클래스의 멤버 함수로, 네트워크 포트와 연결된 다른 프로세스에게 메시지를 전달하는 역할을 한다.

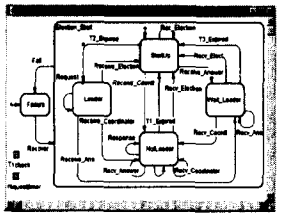
```
class TNetworkTimer: public TDynamicTimer
{
  public:

  TNetworkTimer( TActiveObject* network, TMessage* message )
   :TDynamicTimer( network, TConstDistr( TNetworkPort::MeanDelay) ),
  Message( message )
  {}

 ~TNetworkTimer()
  {
    delete Message;
  }

  virtual void Expiry()
  {
    ( (TNetwork*)Get_Object() )->Port->Send( Message );
    Message = 0;
  }

private:
    TMessage* Message;
};
```

[그림 2] TNetworkTimer 클래스

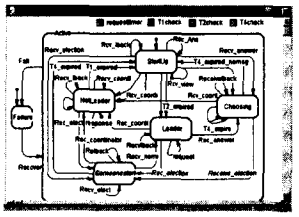
3.2.3 TProcess

TProcess는 시스템을 구성하는 프로세스의 속성을 정의한다. 프로세스가 주고 받는 메시지의 종류, 선출 과정에서 request,election,coordinator 메시지를 기다리는데 필요한 T1,T2,T3 타이머,다른 프로세스의 정보를 저장할 메모리 등을 정의한다. 또한 프로세스가 메시지를 주고 받거나, 시간의 흐름에 따라 상태가 바뀌는데 이를 behavior를 통해 나타낼 수 있다. [그림 3]은 불리 알고리즘의 behavior, [그림 4]는 고속 불리 알고리즘의 behavior를 나타낸 상태도이다.

두 그림에서 공통적으로 시스템 내의 프로세스가 Failure 상태에 있거나 Leader,NotLeader 상태에만 있으면 시스템은 정상적으로 동작을 하고 있다. 그러나 만약 프로세스가 그 이외의 상태에 있을 경우에는 시스템은 선출이 진행중인 상태이다. 상태간 변경은 메시지를 받았거나 타이머가 경과되었을 경우에 활성화되어 프로세스의 상태가 변하게 된다.



[그림 3] 불리 알고리즘에서 Tprocess의 behavior

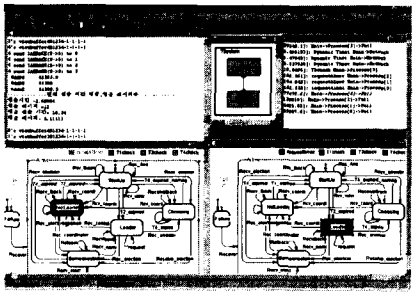


[그림 4] 고속 불리 알고리즘에서 Tprocess의 behavior

4. 성능 측정 및 분석

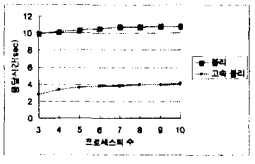
3장에서 기술된 모델을 수행하여 알고리즘이 정확히 동작하는지 정확성을 증명하였으며 불리 알고리즘과 고속 불리 알고리즘의 성능을 비교 측정하였다.선출 알고리즘의 성능을 평가하는 가장 중요한 척도는 선출 지연시간과 소요 메시지 수이다. COVERS의 Logging 기능과 Tuning 기능을 이용하여 수행중 파라메터의 변화를 주어 다양한 상황에서의 선출 알고리즘의 성능을 측정하였다.

[그림 5] 모델 실행 환경

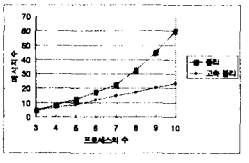


[그림 5]은 모델 실행 화면을 보인 것이다. Log 화면으로 응답 시간과 소요 메시지 수가 나타나며 각 프로세스마다 상태 변화를 그림으로 관찰할 수 있다.

[그림 6]과 [그림 7]은 구현 모델에 프로세스의 수를 변화시키면서 각 알고리즘에서 선출 지연 시간과 메시지 수를 비교한 그래프이다. 평균 선출 응답 시간은 프로세스 개수에 관계없이 비교적 일정하며 고속 불리 알고리즘이 불리 알고리즘과 비교해서 약 35%~80% 정도 단축되었다. 메시지 수도 불리 알고리즘에서는 조정자가 죽은 것을 알아낸 프로세스가 election 메시지를 보내면 더 높은 우선순위 번호의 프로세스들이 연쇄적으로 election, answer 메시지들을 생성시키므로 기하 급수적으로 메시지가 발생하지만, 고속 불리 알고리즘에서는 하나의 프로세스만이 election 메시지를 보내고 answer 메시지도 하나의 프로세스에게만 보내기 때문에 프로세스 수에 비례해 선형적(linear)인 수의 메시지로 선출이 이루어진다.

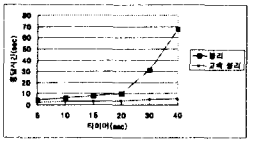


[그림 6] 프로세스의 수에 따른 각 알고리즘의 응답 시간의 비교

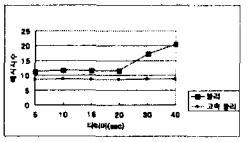


[그림 7] 프로세스의 수에 따른 각 알고리즘의 메시지 수의 비교

[그림 8]과 [그림 9]는 T2 타이머 값이 변화될 때 각 선출 알고리즘의 응답 시간과 메시지 수를 비교한 것이다.불리 알고리즘에서 T2 값이 커질수록 선출 과정 도중에 다른 프로세스에 의해 선출이 반복적으로 일어나므로 선출 시간이 길어지며 메시지 수도 증가한다. 그러나 고속 불리 알고리즘에서는 받아야 할 메시지를 다 받으면 T2 동안 기다려야 할 필요가 없으므로 선출 시간과 메시지 수가 일정하다.



[그림 8] T2 타이머 변화에 따른 각 알고리즘의 응답시간의 비교



[그림 9] T2 타이머 변화에 따른 각 알고리즘의 메시지 수의 비교

[표 1]은 30,000,000초 동안 정상적으로 선출이 일어났을 때와 선출 중 장애가 발생했을 때 모두 고려한 평균 응답시간과 메시지 수이다. 선출이 발생하였을 때 정상적으로 선출이 완료되는 경우도 있으나 선출 도중에 임의의 프로세스가 죽어 선출이 다시 일어나는 경우도 있는데 아래 표는 시뮬레이션동안에 발생한 모든 선출 과정에서의 평균 응답시간과 메시지 수이다. 고속 불리 알고리즘의 응답시간과 메시지 수가 불리 알고리즘보다 경제적임을 알 수 있다.

평균 응답 시간(sec)		평균 메시지 수(개)	
불리	고속 불리	불리	고속 불리
22.6992	3.92457	11.9206	8.60701

[표 1] 실험 결과

5. 결 론

COVERS의 다양한 기능을 통해 분산 네트워크 환경과 두 알고리즘을 구현하고 시험 및 성능을 다각도로 측정 분석하였다. 메시지를 기다리는 타이머 값에 변화를 주거나, 프로세스의 수를 변화시키면서 성능을 측정하였고, 선출 도중의 장애 상황을 발생시켜서 응답 시간과 소요 메시지 수를 측정하였다. 그 결과 고속 불리 알고리즘이 기존의 알고리즘보다 우수한 성능을 가짐을 보였다. 또한 COVERS 툴이 분산 시스템 모델링 및 분석에 효과적인 툴 임을 알 수 있었다.

[참고 문헌]

[1] S.Mullender,*Distributed Systems(2nd Ed.)*, Addison-Wesley, 1993.

[2] Seok-Hyung Lee, Hoon Choi, "The Fast Bully Algorithm : For Electing a Coordinator Process in Distributed Systems," IEEE TPDS, 심사 중.

[3] 최 훈, "분산 시스템 조정자 프로세스의 효율적인 선출 알고리즘", 정보과학회논문지, vol.25, no 9, Sep. 1998.

[4]Andrei V. Borshchev, Yuri G. Karpov and Victor V. Rouadkov,"System Modeling,Simulation and Analysis Using COVERS",http://www.xjtek.com/reference/articles/monterey/index.html, 1997.

[5]E. G. Chang and R. Robert, "An improved algorithm for decentralized extreme-finding in circular configurations of processors," Communications of ACM, vol. 22, no. 9, pp.281-283, 1979.

[6]H. Garcia-Molina, "Elections in a distributed computing system," IEEE Trans. on Computers, vol. 31, pp.48-59, Jan. 1982.