분산 컴퓨팅 시스템에서 에이전트를 이용한 회복 기법

이화민*O 정광식** 윤태명*** 이원규* 유헌창*
* 고려대학교 대학원 컴퓨터교육과 ** 런던대학교 컴퓨터학과 ***고려대학교 대학원 컴퓨터학과
{zelkova, lee, yuhc}@comedu.korea.ac.kr, k.chung@ucl.ac.uk

A Recovery Technique Using Agent in Distributed Computing Systems

Hwa-Min Lee*O Kwang-Sik Chung** Tae-Myung Yoon*** Won-Gyu Lee* Heon-Chang Yu*
* Dept. of Computer Science Education, Korea University ** Dept. of Computer Science, University College London

요 약

분산 컴퓨팅 시스템은 단일 시스템보다 결함에 민감하기 때문에 기존의 많은 연구들에서 분산 시스템에서 결함이 발생할 경우 이를 해결하기 위한 많은 복귀회복기법들이 연구되었다. 본 논문에서는 기존의 분산 컴퓨팅 시스템의 결함 포용 기법에 멀티 에이전트의 개념을 도입하여 운영체제에 독립적인 에이전트를 이용한 회복기법을 제안한다. 이를 위해 본 논문에서는 프로세스의 회복을 담당할 회복 에이전트, 결함 포용 규칙과 정보를 유지·관리하는 정보 에이전트, 전체 에이전트간의 통신 기능을 담당할 조정 에이전트를 정의 및 설계하고 회복 에이전트를 이용한 회복 알고리즘을 제안한다. 분산 컴퓨팅 시스템에서 회복 에이전트의 도입은 결함 발생 프로세스의 결함 회복 작업을 어플리케이션 계층과 독립적인 별도의 계층으로 계층화하여 결함 포용을 위한 메카니즘의 이식성 증대 및 확장성 증대를 가져온다.

1. 서 론

분산 컴퓨팅 시스템은 통신 네트워크에 의해서 서로 연결된 자율적인 프로세스들의 집합체로서 각 프로세스간의 통신은 메시지를 통해서만 이루어진다. 분산 컴퓨팅 시스템은 각 노드의 결함 발생 가능성과 노드를 상호 연결하는 통신 수단에서의 결함 발생 가능성까지 존재하기에 결함 발생 확률이 단일 시스템에서의 결함 발생 확률보다 월등히 크다. 또한 분산 컴퓨팅 시스템은 각 프로세스간 공유메모리가 없기 때문에 장시간 수행되는 응용프로그램의 경우, 임의의 프로세스에서 결함이 발생하면 프로세스의 의존성으로 인해 전체 응용프로그램이 원하는 결과를 산출해내지 못하는 잠재적인 위험이 존재한다[1]. 이처럼 분산 컴퓨팅 시스템은 단일 시스템보다 결함에 대해 민감하기 때문에 기존의 많은 연구들에서 분산 시스템에서 결함이 발생할 경우 이를 해결하기 위한 많은 복귀회복기법들이 연구되었다[2][3].

기존의 결함 포용 기법에 관련된 검사점기반 회복 기법들과 메시지 로그기반 회복기법들의 대부분은 고장-멈춤(fail-stop) 모델을 채용함으로써 결함이 발생한 프로세스에 대해 운영체제의 관리하에 모든 회복 작업이 수행되어 왔다. 즉, 결함이 발생한 프로세스와 의존 관계에 있는 모든 프로세스들은 각각의 운영체제의 관리하에서 결함 발생 프로세스의 회복에 관련된 동기화를 수행해야 했으며, 이러한 회복 기법은 전체 분산 컴퓨팅 시스템의 성능을 저하시키는 원인이 되었다.

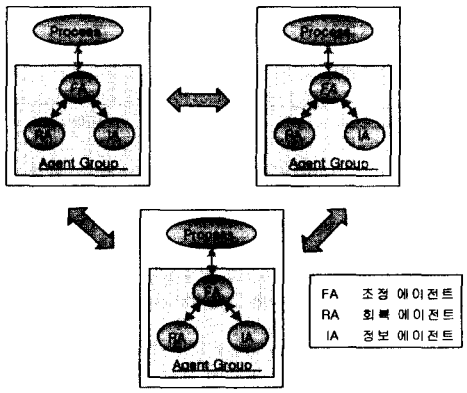
이에 본 논문에서는 분산 컴퓨팅 시스템의 결함 포용 기법에 멀티 에이전트의 개념을 도입하여 운영체제로부터 독립적인 회복기법을 제시한다. 즉, 회복 에이전트는 분산 컴퓨팅 시스템에서 운영체제의 시스템 영역에서 수행되는 결함 발생 프로세스의 결함 회복 작업을 사용자 영역에서 수행되는 서버가 담당하도록 하여 어플리케이션 계층과 독립적인 별도의 계층으로 계층화한다. 이와 같은 계층화를 통해 모듈화가 이루어질 경우 분산 컴퓨팅 시스템에서 결함 포용을 위한 메카니즘의 개방성이 증대된다. 또한 이는 분산 컴퓨팅 시스템에서 결함 포용 기능이 계층화되어 모듈화가 이루어지면 현재 상용 분산 컴퓨팅 시스템을 비롯한 여러 분산 컴퓨팅 시스템에 결함 포용 기능의 탑재가 쉬워져 이식성의 증대 및 확장성 증대를 가져온다. 본 논문에서 제안한 회복 에이전트를 이용한 회복기법은 독립 검사점(independent check pointing) 기법과 비관적 메시지 로깅(pessimistic message logging) 기법을 사용한다[2][4].

2장에서는 본 논문에서 제안하는 에이전트 기반 결함 포용 시스템의 구조를 설명하고 3장에서는 에이전트간의 통신 언어를 정의한다. 4장에서는 에이전트를 이용한 회복 알고리즘에 대해 기술하고 5장에서는 결론 및 향후 연구과제를 제시한다.

2. 멀티 에이전트 기반 결함 포용 시스템 구조

2.1 시스템 모델

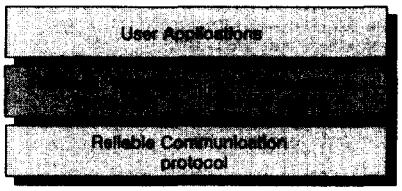
시스템 모델은 응용프로그램을 수행하는 프로세스들과 회복·조정·정보 에이전트 그리고 이들간의 통신 채널로 구성되는 분산 컴퓨팅 시스템 환경이다. 여기서 말하는 분산 컴퓨팅 시스템 환경은 공유 메모리와 같은 기능을 사용하지 않고 오직 메시지만으로 프로세스들과 에이전트들이 통신을 하는 메시지 전달 시스템을 기반으로 한다. 통신 환경은 통신 네트워크가 분할되지 않는다고 가정하며 프로세스간 통신은 신뢰성(reliability)을 보장하여 메시지를 선입선출(first-in first-out) 방식으로 전달한다고 가정한다. 또한 프로세스의 고장에 대해서는 고장-멈춤 모델에 따라 프로세스는 휘발성 메모리에 저장된 상태만을 잃어버리고, 실행을 중지한다고 가정한다. 본 논문에서 제안하는 멀티 에이전트를 이용한 결함 포용 시스템의 전체 구성도는 <그림 1>과 같다.



<그림 1> 결함 포용 시스템의 구조

기존 연구에서는 회복 기법이 사용자 애플리케이션과 신뢰성 있는 통신 기법 사이에 위치하였는데 회복기법에 회복 에이전트를 도입한 경우 회복 기법과 프로세스간 통신은 <그림 2>와 같이 변화된다. 수정된 회복기법에 따라 기존의 비관적 메시지 로깅에서는 통신 프로토콜을 통해 프로세스에게 메시지가 전달되고 프로세스가 수신한 메시지를 안정된 저장소에 로깅을 했지만 회복 에이전트를 이용한 회복기법에서

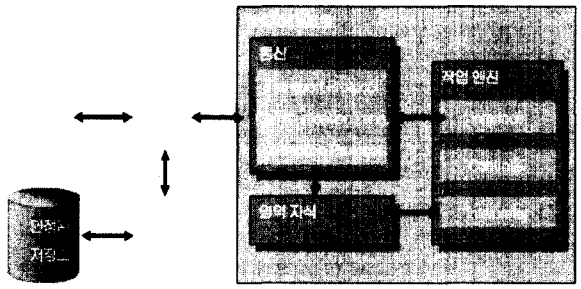
는 프로세스에게 메시지가 전달되기 전 메시지는 조정 에이전트에게 먼저 전달되고 정보 에이전트에 의해 안정된 저장소에 로그된다.



<그림 2> 변화된 3-계층 구조

2.2 시스템 구성 요소

본 논문에서 제안하는 결함 포용 시스템은 회복·조정·정보 에이전트로 구성된다. 회복 에이전트는 분산 컴퓨팅 시스템에서 결함 발생시 자신의 지식 정보와 다른 회복 에이전트와의 협력을 통해 자치적으로 회복 과정을 수행하는 에이전트이다. 정보 에이전트는 회복 에이전트가 결함 발생시 회복 과정 수행을 위해 필요한 프로세스에서 발생한 정보들을 영역지식으로 구축해주는 일을 담당하는 에이전트이다. 그리고 본 논문에서는 에이전트들의 효율적인 통신을 도울 수 있도록 특별한 조정 에이전트를 정의하여 에이전트 연방 시스템을 구축한다. 본 논문에서의 회복 에이전트와 정보 에이전트 그리고 조정 에이전트를 이용한 회복기법의 기본 모델은 <그림 3>과 같다.



<그림 3> 회복 에이전트의 기본 모델

3. 에이전트의 통신 언어

3.1 회복기법을 위한 온톨로지

논문에서는 분산 컴퓨팅 시스템에서 결함포용을 위해 사용하는 어휘들을 Rollback_Recovery라 이름하여 에이전트들의 온톨로지로 새롭게 정의한다. 정의된 온톨로지는 <표 1>과 같다.

<표 1> 회복기법을 위한 Rollback_Recovery 온톨로지

어휘	의미
checkpoint	프로세스가 검사점을 취한다.
send	다른 프로세스에게 메시지를 전송한다.
receive	다른 프로세스로부터 메시지를 수신한다.
before	앞의 이벤트가 뒤의 이벤트보다 선행된다.
after	앞의 이벤트가 뒤의 이벤트보다 후행된다.
log	수신한 메시지를 로그한다.
orphan	해당 메시지가 고아메시지이다.
recovery	결함 발생 후 회복을 수행한다.
rollback	프로세스가 해당 사건까지 복귀한다.

3.2 KIF를 이용한 영역 지식 구축

본 논문에서 결함 포용을 제공하는 에이전트의 영역지식을 위해 정의한 KIF의 BNF 문법은 <표 2>와 같다.

3.3 KQML을 이용한 에이전트간 통신 언어

본 논문에서 사용하는 KQML의 BNF 문법은 <표 3>과 같다.

<표 2> KIF의 BNF 문법

```
<special> ::= " | # | ' | ( | ) | , | \ | ^ | `
<normal> ::= <stringchar>*" | #<digit><digit>*"<ascii>*
<word> ::= <normal> | <word>*
<expression> ::= <word> | (<expression>*)
<string> ::= empty | "<normal>
<indvar> ::= <?word>
<seqvar> ::= <@word>
<sentop> ::= - | /- | not | and | or | -> | <- | <-> | forall | exists
<variable> ::= <indvar> | <seqvar>
<constant> ::= <word> - <variable> - <operator>
<logterm> ::= (if <sentence> <term> [<term>]) | (cond (<sentence>
              <term>) ··· (<sentence> <term>))
<quoterm> ::= (quote <expression>)
<term> ::= <indvar> | <constant> | <string> | <funconst> | <logterm>
           | <quoterm>
<sentence> ::= <logconst> | <equation> | <inequality> | <relsent> |
               <logsent> | <quantsent>
<equation> ::= (= <term> <term>)
<inequality> ::= (/= <term> <term>)
<relsent> ::= (<relconst> <term>* [<seqvar>]) | (<funconst> <term>*
              <term>)
<logsent> ::= (not <sentence>) | (and <sentence>*) | (or <sentence>*) |
              (-> <sentence>* <sentence>) | (<- <sentence>
              <sentence>*) | (<-> <sentence> <sentence>)
<quantsent> ::= (forall <indvar> <sentence>) | (forall (<indvar>*)
                <sentence>) | (exists <indvar> <sentence>) | (exists
                (<indvar>* <sentence>)
```

<표 3> KQML의 BNF 문법

```
<performative> ::= (<word> [<whitespace> :<word> <whitespace>
                   <expression>]*)
<expression> ::= <word> | <quotation> | <string> | (<word> [
                 <whitespace> <expression>]*)
<word> ::= <character><character>*
<character> ::= <alphabetic> | <numeric> | <special>
<special> ::= < | > | = | + | - | * | / | & | ^ | ~ | _ | @ | $ | % | : |
              . | ! | ?
<quotation> ::= '<expression> | '<comma-expression>
<comma-expression> ::= <word> | <quotation> | <string> |
                       ,<comma-expression> (<word>
                       [<whitespace> <comma-expression>]*)
<string> ::="<stringchar>*" | #<digit><digit>*"<ascii>*
<stringchar> ::= \<ascii> | <ascii> -\-<double-quote>
```

4. 에이전트를 이용한 회복 알고리즘

4.1 자료구조

N개의 프로세스 p_i (i=1, 2, ···, N)로 구성된 시스템에서 프로세스 p_i의 회복 에이전트는 대문자 RA_i로, 정보 에이전트는 IA_i로, 조정 에이전트는 FA_i로 표기한다. 회복 알고리즘에서 사용되는 함수들과 자료구조는 다음 <그림 4>와 같다.

Data Structure

- a_i : 조정 에이전트, 정보 에이전트, 회복 에이전트를 통칭
- R : 조정 에이전트에 등록된 에이전트 집합
- $ProF$: 조정 에이전트에 등록된 에이전트 프로파일 리스트
- $ProF_i$: 임의의 에이전트 a_i의 프로파일
- DK : 정보 에이전트의 영역지식
- *lastckp* : 프로세스가 결함발생 전 마지막으로 취한 검사점
- *eventlist* : 결함발생 전 마지막으로 취한 검사점 이후의 이벤트 리스트
- *rollbackpoint* : 결함발생 후 회복을 위해 복귀할 지점

<그림 4> 자료구조

4.2 정상 수행 시 알고리즘

정상 수행 시 조정 에이전트와 정보 에이전트의 작동 알고리즘은 <그림 5>와 같다. 단, 알고리즘에서 발생하는 모든 메시지는 조정 에이전트를 통하여 프로세스와 다른 에이전트에게 전달된다.

```
Facilitator Agent FAi
  Created with process pi;
    R ← ∅;   ProF ← ∅;
    if receive register message from ai then
          if (ai ∉ R) then
             R ← R ∪ {ai};        ProF ← ProF ∪ {ProFi};
          fi;
          send ack message to ai;
    fi;
  Do
    if receive system mesage from process pi then
      translate system message to KQML;
      send addDK_request message to IAi;
    fi;
  oD;
Information Agent IAi
  Created with process pi;
  Do
    send register message to FAi;
    receive ack message from FAi;
    if receive addDK_request message from FAi then
      translate addDK_request message to KIF;
      add KIF to DK;
    fi;
  oD;
```

<그림 5> 정상 수행 시 조정·정보 에이전트의 작동 알고리즘

4.3 결함 발생 시 알고리즘

결함 발생 시 조정 에이전트와 회복 에이전트, 그리고 정보 에이전트의 작동 알고리즘은 <그림 7>, <그림 8>, <그림 9>와 같다.

```
Facilitator Agent FAi
 Do
   if receive recovery_request message from process pi then
     translate message to KQML;
     send recovery_request message to RAi;
   fi;
   if receive information_request message from RAi then
     route information_request message to IAi;
       receive information_reply message from IAi;
     send information_reply message to RAi;
   fi;
   if receive rollback_request message from RAi then
     translate KQML to system message;
     send rollback_request message to process pi;
     if receive replay_request message from RAi then
       broadcast replay_request message to FAj;
       receive ack message message from FAj;
     fi;
     receive ack message message from process pi;
     send ack message message to RAi;
   fi;
 oD;
```

<그림 7> 결함 발생 시 조정 에이전트의 작동 알고리즘

```
Recovery Agent RAi
  Do
    if receive recovery_request message from FAi then
      send lastckp_request message to IAi;
      if receive lastckp_reply message from IAi then
         send loglist_request message to IAi;
         if receive loglist_reply message to RAi then
           decide rollback_point;
           send rollback_request to pi;
           send replay_request to Fj;
           receive ack message to pi;
         fi;
      fi;
    fi
  oD;
```

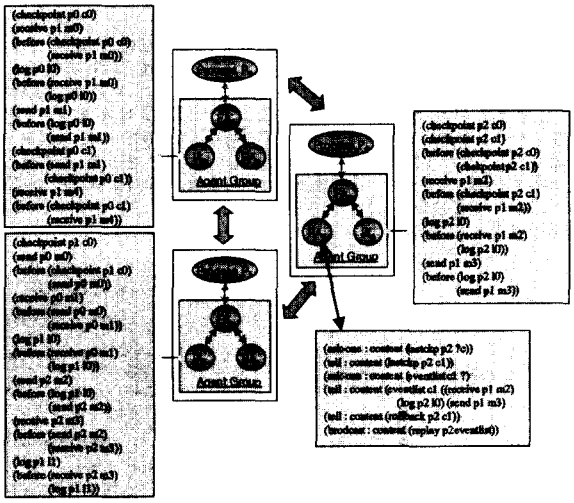
<그림 8> 결함 발생 시 회복 에이전트의 작동 알고리즘

4. 4 제안된 알고리즘의 적용 예

<그림 10>은 멀티 에이전트를 이용한 회복 알고리즘의 적용 예이다. <그림 10>의 전체 시스템은 프로세스 p_0, p_1, p_2로 구성되고 프로세스 p_2에서 메시지 m_3를 프로세스 p_1에게 전송 후 결함이 발생한 경우이다.

```
Information Agent IAi
 Do
   if receive lastckp_request message to IAi then
     find last_checkpoint from DK;
     translate last_checkpoint KIF to KQML;
     send lastckp_reply message to RAi;
   fi;
   if receive loglist_request message to IAi then
     find loglist after last_checkpoint from DK;
     translate loglist KIF to KQML;
     send loglist_reply message to RAi;
   fi;
 oD;
```

<그림 9> 결함 발생 시 정보 에이전트의 작동 알고리즘



<그림 9> 알고리즘 적용 예

5. 결론 및 향후 연구과제

본 논문에서는 기존의 분산 컴퓨팅 시스템의 결함 포용 기법에 멀티 에이전트의 개념을 도입하여 회복·정보·조정 에이전트를 정의 및 설계하여 운영체제에 독립적인 에이전트를 이용한 회복기법을 제안하였다. 에이전트를 이용한 회복기법은 결함 발생 프로세스의 결함 회복 작업을 사용자 영역에서 수행되는 서버가 담당하도록 하여 어플리케이션 계층과 독립적인 별도의 계층으로 계층화한다. 이와 같은 계층화는 분산 운영 체제의 핵심 기술이라 할 수 있는 마이크로커널의 구현을 위해 필수적인 사용자 영역에서의 서버의 시스템 기능성 제공에 에이전트를 도입하는 의미있는 시도가 된다. 또한 이는 현재 상용 분산 컴퓨팅 시스템을 비롯한 여러 분산 컴퓨팅 시스템에 결함 포용 기능의 탑재가 쉬어지는 이식성의 증대 및 확장성 증대를 가져온다.

향후 연구과제로 에이전트들간의 통신 환경으로 코바를 이용하고, 각각의 에이전트는 쓰레드로 설계하여 본 논문에서는 설계 및 제안한 에이전트를 이용한 회복기법의 수행을 모의실험하고자 한다.

참고문헌

[1] L. Alvisi, "Understanding the message logging paradigm for masking process crashes," *PhD. Thesis, Department of Computer Science, Cornell University*, Jan. 1996.

[2] E. N. Elnozahy, D. B. Johnson and Y. M. Wang, "A Survey of Rollback-Recovery Protocols in Message Passing Systems," *CMU Technical Report CMU-CS-99-148*, June 1999.

[3] E. L. Elnozahy, W. Zwanepoel. "Manetho: Transparent rollback-recovery with low overhead, limited rollback and fast output commit," *IEEE Transactions on Computers*, 41(5):526-531, March 1992.

[4] L. Alvisi and K. Marzullo, "Message Logging: Pessimistic, Optimistic, Causal and Optimal," *IEEE Trans. on Software Engineering*, Vol. 24, pp. 149-159, Feb. 1998.