이동 에이전트의 순환적 작업 수행을 지원하는 시간적 복제 기반 결함 포용 기법

박한석[0] 백맹순 김홍수 황종선
고려대학교 컴퓨터학과 분산시스템 연구실
{indie[0], lotieye, hera, hwang}@disys.korea.ac.kr

Temporal Replication Based Fault Tolerant Scheme Providing A Cyclic Execution of Mobile Agent

Hanseok Park[0], Maengsoon Baik, Hongsoo Kim, Chongsun Hwang
Dept. of Computer Science & Engineering, Korea University

요 약

다중 지역 이동 에이전트 컴퓨팅 시스템에서는 단일 지역으로 구성된 시스템에 비해서 호스트의 결함이나 호스트 사이의 통신 결함 발생 확률이 높아, 안정된 시스템 설계에 있어서 이동 에이전트의 결함을 검출하고 이를 복구하는 결함 포용 기법은 매우 중요한 고려 사항이다. 이동 에이전트의 안정적인 연산 수행을 보장하기 위한 기존의 결함 포용 기법들은 크게 시간적 복제 기반 기법(Temporal Replication Based Approach: TRBA)와 공간적 복제 기반 기법(Spatial Replication Based Approach: SRBA)으로 구분 지을 수 있으나, 다중 지역으로 구성된 이동 에이전트 시스템과 같은 복잡한 시스템에서는 낮은 결함 포용 비용이 요구되는 TRBA가 보다 적합하다. 그러나 기존의 TRBA에서는 이동 에이전트의 비멱등(non-idempotent) 연산 수행을 지원하기 위해 단일 수행 (exactly-once execution) 특성을 보장했지만, 이동 에이전트가 순환적 작업 경로를 가지는 연산 수행 시에는 작업을 완결하지 못하는 *복귀월산 문제(comeback-skip problem)*가 발생한다. 본 논문에서는 플래그와, 히스토리 필드, 파손 필드를 도입하여 복귀월산 문제를 해결하는 시간적 복제 기반 결함 포용 기법을 제안한다. 이 기법은 이동 에이전트의 다양한 작업 수행을 지원함으로써 이동 에이전트의 작업 수행 영역을 확대한다.

1. 서론

에이전트(agent)는 사용자나 기관을 대신해서 자율적으로 행동하는 컴퓨터 프로그램이며, 이동 에이전트(mobile agent)는 네트워크 내에서 한 시스템으로부터 다른 시스템으로 자기 자신을 전송할 수 있는 능력이 더 부여된 컴퓨터 프로그램이다.[1] 이동 에이전트가 다른 호스트로 이동할 땐 에이전트 코드와, 데이터, 실행 상태가 포착되어 다음 호스트에 전송되며, 도착 후 이동 에이전트가 시작된다. 이동 에이전트 실행은 단계들로써 진행되며, 새로운 단계는 이동 에이전트가 다음 호스트로 이주할 때마다 시작된다.[2]

이동 에이전트 컴퓨팅 시스템에서 호스트 중 어느 하나에 결함이 발생하거나 두 호스트 사이에서 통신 결함이 발생하면, 호스트를 옮겨 다니며 작업을 수행하는 이동 에이전트들이 영향을 받는다. 더욱이 호스트 수가 많아질수록 결함이 발생하는 횟수가 많기에, 단일 지역 이동 에이전트 컴퓨팅 시스템을 확장한 다중 지역 이동 에이전트 컴퓨팅 시스템에서는, 이동 에이전트의 결함을 검출하고 복구하는 결함 포용 기법이 더욱 중요하다.

이동 에이전트 컴퓨팅 시스템에서 안정적인 연산 수행을 위해 제안된 결함 포용 기법들은 검사점을 이용하는 방법과 복제를 이용하는 방법으로 나뉘며, 복제를 이용하는 방법은 TRBA와 SRBA로 분류된다.[2] 이전 단계로 롤백 복구를 하는 방식인 TRBA는, 단계마다 여러 복제 이동 에이전트들이 연산을 수행하고 결과를 합의하는 SRBA보다 결함을 포용하는 비용이 상당히 적게 드는 장점이 있다.[3] 기존의 TRBA에서는 중복 연산 취소를 통해서 비멱등 연산 수행이 가능하다. 하지만, 이동 에이전트가 연산을 반복 수행하기 위해서 이전에 방문한 호스트를 다시 방문할 경우에는 그 연산이 무시되는 복귀월산 문제가 발생한다.

본 논문에서는 다중 지역 이동 에이전트 컴퓨팅 시스템 환경에서, 이동 에이전트의 방문 횟수를 기록한 플래그를 호스트에 남기고 이동 에이전트 지역 서버의 레지스터에 이동 에이전트의 히스토리 정보와 파손 여부를 기록하여 복귀월산 문제를 해결하는 시간적 복제 기반 결함 포용 기법을 제안한다. 제안 기법에서는 이동 에이전트가 다양한 작업 수행을 하더라도 단일 수행 특성이 위배되지 않으므로, 이 기법 사용 시 이동 에이전트 시스템 분야에서 다루는 작업 수행 영역을 확대할 수 있다.

2. 관련 연구

이동 에이전트 컴퓨팅 시스템의 결함 포용 기법 중 검사점 방식은, 결함이 발생한 호스트가 회복이 될 때까지 기다렸다가 회복이 되면 안정 저장 장치(stable storage)에 저장된 검사점을 이용하여 호스트를 복구한다.[2] 이 방식은 결함 포용 비용이 다른 방식들에 비해서 적지만, 에이전트 결함을 해결하지 못하는 단점과 이동 에이전트 실행이 차단되는 단점이 있다. SRBA는 한 단계가 이동 에이전트 복사 본을 가진 여러 호스트들로 이루어지며, 한 호스트에서 결함이 발생하면 다른 호스트에서 복제 이동 에이전트가 동일한 연산을 수행한다. 그리고 다음 단계로 넘어 갈 때에는 단계 내 이동 에이전트 복사 본들끼리 결과를 합의하여 단일 수행 특성을 보장한다.[2] 이 방식의 큰 단점은 합의 비용과 이동 에이전트 이주 비용이 너무 크다는 점이다. 따라서 SRBA는 이동 에이전트 시스템의 실질적인 결함 포용 기법의 방식으로는 적합하지 않다.

TRBA는 이전 단계의 호스트가 다음 단계의 이동 에이전트 복사 본을 저장하고 있으며, 결함이 발생했을 경우 이전 단계로 롤백 복구를 통해 이동 에이전트를 복구한다.[2] TRBA는 MAF[4]와, NetPebbles[5], JAMES[6]의 결함 포용 방식으로 적용되었다. MAF에서는 검사점 관리자가 이동 에이전트 결함을 검출하고 복구하는 결함 포용 기법이 사용되었다. 이 간단한 기법은 비멱등 연산 수행과, 네트워크 파티션, 중복된 이동 에이전트 처리, 검사점 관리자의 결함 등을 고려하지 않았다. NetPebbles에서는 분산된 결함 검출과, 비결정성을 사용한 롤백 복구, 중복된 이동 에이전트를 처리를 위한 폐영역 회수 서비스를 도입했다. 그러나 비멱등 연산 수행은 다루지 않았다. JAMES에서는 TRBA에선 처음으로 비멱등 연산을 다룬 결함 포용 기법을 사용했다. 에이전트 검색 디렉터리와 플래그를 두어 비멱등 연산 수행 시에 한번 이하 수행(at-most once execution) 특성을 보장하였지만, 복귀월산 문제가 발생하면 이동 에이전트 수행이 완결되지 못하는 결점이 있고 결함 검출과 복구가 LAN 환경에 맞게 도안이 되어서 확장성이 떨어지는 단점이 있다.

3. 이동 에이전트 컴퓨팅 시스템 모델

이동 에이전트 컴퓨팅 시스템 모델은 이동 에이전트와 실행 장소인 호스트로 구성된다.[1] 본 논문에서 가정하는 이동 에이전트는 각 호스트에서의 실행 결과에 따라 다음 호스트를 동적으로 선택하여 호스트와 호스트 사이를 옮겨 다니며 작업을 수행한다. 이동 에이전트 시스템들 중에서 같은 권한을 가진 이동 에이전트 시스템들의 집합을 지역이라고 하고, 각 지역에는 위치 관리를 담당하는 MARS(Mobile Agent Region Server)가 존재한다. 그리고 다중 지역 이동 에이전트 컴퓨팅 시스템 모델에서 MRMARMS(Multi-Region Mobile Agent Registration or Management Server)는 이동 에이전트의 전역 네이밍 서비스를 제공한다.[7] 본 논문에서는 이동 에이전트가 여러 지역으로 구성된 다중 지역 이동 에이전트 컴퓨팅 시스템에서 이동하면서 작업을 수행하는 것으로 가정한다. 그림 1은 본 논문에서 가정하는 다중

지역 이동 에이전트 컴퓨팅 시스템 모델을 나타내고 있다.

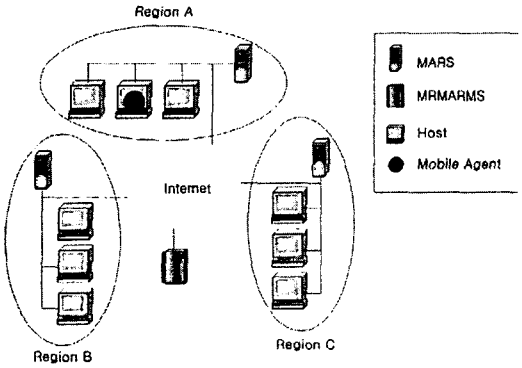

그림 1. 다중 지역 이동 에이전트 컴퓨팅 시스템 모델

4. 복귀월산 문제

이동 에이전트 실행은 단계들로 수행되며 호스트를 이주할 때마다 새로운 단계가 시작된다. 각 단계마다 이동 에이전트가 수행할 연산이 있으며, 그 연산이 비멱등 연산일 경우에는 단일 수행 특성이 보장되어야 한다.

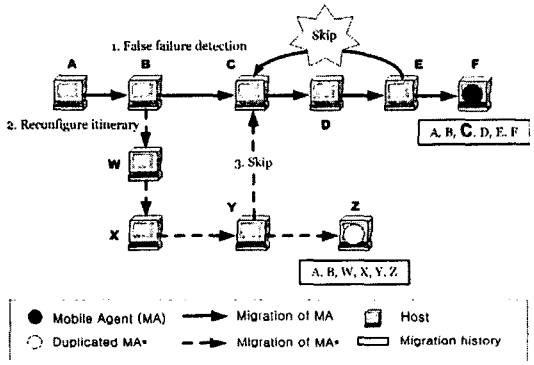

그림 2. 단일 수행 특성을 보장하는 TRBA에서 복귀월산 문제

TRBA에서는 롤백 복구를 사용한다. 그로인하여 그릇된 결함 검출에 의해 생긴 중복된 이동 에이전트들이 같은 호스트를 순서 없이 방문하여 동일한 연산을 중복 수행할 수가 있다. 이 경우에는 중복 연산들을 취소하더라도 단일 수행 특성이 보장되지 않는다. 이 문제를 방문 사실을 나타내는 플래그를 사용하여 해결했다.[3]

하지만, 중복된 이동 에이전트가 아닌 원래의 이동 에이전트가 새로운 연산을 수행하려고 이전에 방문했던 호스트를 다시 방문했을 때에는 문제가 발생한다. 그림 2의 실선 화살표에서처럼 이동 에이전트는 중간에 실행했어야 할 연산을 건너뛰었기에, 단일 수행 특성을 보장하지 못하여 이동 에이전트의 작업이 완결되지 않는다. 이 문제에 대한 정의는 다음과 같다.

정의 1. (인디 수행) 이동 에이전트 ma_i와 호스트 H_j가 다음과 같은 연산 수행을 보일 때 *인디 수행*이라 한다.

- $\alpha < \beta : \sum_{mc=\alpha}^{\beta} \left[\frac{ma_i^{mc}}{H_j}\right] = \left[\frac{ma_i^{\alpha}}{H_j}\right] \oplus \left[\frac{ma_i^{\alpha+1}}{H_j}\right] \oplus \cdots \oplus \left[\frac{ma_i^{\beta}}{H_j}\right]$
- $\forall l, k : l \neq k : \left[\frac{ma_i^{l}}{H_j}\right] \neq \left[\frac{ma_i^{k}}{H_j}\right]$

$\left[\frac{ma_i^{l}}{H_j}\right]$은 H_j에서 ma_i의 l 번째 방문을 통한 수행을 의미한다. $\oplus$은 비반사적(irreflexive), 비대칭적(asymmetric), 추이적(transitive) 특성을 가지는 연산 수행 순서를 의미한다.

즉, 이동 에이전트가 동일 호스트 내에서 방문 순서로써만 구분되는 복수 개의 연산들을 수행할 때, 이 수행을 인디 수행이라 한다.

정의 2. (복귀월산 문제) 임의의 호스트 H_j가 단일 수행 특성을 보장하기 위해 자신에게서 수행되는 모든 이동 에이전트의 연산에 아래와 같은 특성을 요구하는 경우,

$$\exists j : \forall l, k : l \neq k : \left[\frac{ma_i^{l}}{H_j}\right] = \left[\frac{ma_i^{k}}{H_j}\right]$$

ma_i는 H_j에서 인디 수행을 할 수 없게 되고, 이를 이동 에이전트 수행에 있어서의 *복귀월산 문제*라 한다.

5. 제안 기법

본 논문에서 제안하는 기법은 TRBA 중에서 조정자를 쓰는 중앙집중식 방식이 아니고, 각 호스트들이 결함을 검출하고 개개의 호스트들이 에이전트 복사 본을 유지하는 분산 방식이다. 또한, 중복된 이동 에이전트를 검출하기 위해서는 MARS와 홈 MARS 레지스터의 이동 에이전트 위치 정보에 히스토리필드와 파손 필드를 두었고, 복귀월산 문제를 해결하기 위해서 각 호스트에 이동 에이전트의 방문 횟수를 기록한 플래그를 두었다. 제안 기법은 아래와 같이 결함 검출과 에이전트 복제, 결함 복구, 중복된 이동 에이전트 검출로 나뉜다.

5.1 결함 검출

모든 호스트들과 이동 에이전트는 모든 이전 호스트들에게 생존 메시지를 간격이 점차 감소하는 주기로 보낸다. 즉, 이동 에이전트가 지금까지 거쳐 간 이주 경로 상의 호스트가 $H_1, \ldots, H_n$과 같이 n개 있을 때, H_i는 확률 r^{j-i}을 가지는 주기로 H_j에게 생존 메시지를 보낸다. (단, $1 \leq j < i$, 그리고 $0 \leq r \leq 1$) 또한 이동 에이전트는 r^{n-i}의 확률을 가지는 주기로 각 H_i $(1 \leq i \leq n)$ 호스트에게 생존 메시지를 보낸다. 한 호스트가 미리 계산되어 주어진 특정 기간동안 어떠한 생존 메시지도 받지 못하면, 그 호스트는 결함 복구를 시작한다.

5.2 에이전트 복제

각 호스트는 이동 에이전트 별로, 이동 에이전트 식별자와, 실행 상태 정보, 히스토리(그 이동 에이전트가 거쳐 간 이주 경로 정보), 이동 에이전트가 이주한 호스트를 복구 표(recovery table)에 유지한다. 그리고 이동 에이전트가 처음 호스트를 방문했으면 이동 에이전트 별로 플래그를 호스트에 남기며, 플래그에는 이동 에이전트 식별자와 그 이동 에이전트가 이 호스트를 방문한 횟수를 기록한다. 후에 그 이동 에이전트가 다시 그 호스트를 방문했을 때는 플래그의 방문 횟수를 1 증가 시킨다.

```
MA a;
URL nextHost;  /* The host where MA migrated */
boolean success = FALSE;

if (isFailureDetected())
then
    a.id = getCrashedMAid();
    a = RecoveryTable.getMA(a.id);
    while (!success)
    do
        /* Reconfigure itinerary */
        nextHost = findNextAvailableHost(a.history);
        if (nextHost != a.nextHost)
        then
            migrate(a, nextHost);
            a.nextHost = nextHost;
            RecoveryTable.setMA(a);  /* MA Replication */
            success = TRUE;
        fi
    done
fi
```

그림 3. 결함 복구 과정 알고리즘

5.3 결함 복구

결함 검출 후 호스트가 결함이 발생한 이동 에이전트의 복사 본을 안정 저장 장치에서부터 활성화시켜서 이전과는 다른 경로로 이주 시

키고, 복구 표에서 이동 에이전트가 이주한 호스트 정보를 갱신한다. 그림 3은 결함 복구 과정에 대한 알고리즘이다.

5.4 중복된 이동 에이전트 검출

복귀월산 문제를 해결하기 위해서는 제일 먼저, 중복된 이동 에이전트를 검출하고 그 이동 에이전트들이 수행한 연산을 취소해야한다. 그래야만 각 호스트에 있는 플래그의 방문 횟수 정보가 원래의 이동 에이전트가 그 호스트를 방문한 횟수 정보와 일치하게 되고 이동 에이전트의 단일 수행 특성을 보장할 수 있기 때문이다.

```
MA a;

/* when MARS receives update message related to a */
record[] = Register.getRecord(a.id);
for (i =0; i < record.length; i++)
do
    if (record[i].crash == FALSE &&
          compare(Update.history, record[i].history) != 0)
    then
        if (isSameRegion(record[i].location))
        then
            sendMsg(record[i], a.HOME_MARS);
            Register.addRecord(a.id, Update);
        else
            Update.location = NULL;
            sendMsg(Update, record[i].location);
        fi
    else
        if (record[i].crash == TRUE)
        then
            record[i].crash = FALSE;
            record[i].OK = TRUE;
            sendMsg(record[i], a.HOME_MARS);
        fi
    fi
done
```

그림 4. 중복된 이동 에이전트 검출 과정 알고리즘

생존 메시지가 링크 상에서 늦게 전송되거나 손실되어서, 호스트가 그릇된 결함 검출을 했을 경우엔, 이동 에이전트 중복 실행이 발생한다. 그림 4는 중복 이동 에이전트 검출 과정 알고리즘을 나타내며, 두 히스토리 비교 시에 히스토리의 호스트 수가 다를 땐, 앞에서부터 같은 수만큼의 호스트를 취해서 비교한다. 홈 MARS는 수신한 레코드들의 정보와 위치 관리 정보를 이용하여 중복된 이동 에이전트 목록을 중복 표(duplication table)로 유지한다.

요컨대, 중복된 이동 에이전트는 이주 시에 검출되며, 홈 MARS의 중복 표를 통해서 중복된 이동 에이전트들의 위치를 알 수 있다. 제안된 기법에서는 그 중복된 이동 에이전트들에게 각자가 결함 복구 후부터 실행한 연산들을 취소하도록 한다.

6. 정확성 증명

이 절에서는 본 논문에서 제안한 기법을 사용할 때에는 복귀월산 문제가 발생하지 않고 이동 에이전트의 단일 수행 특성이 보장됨을 증명한다.

가정 1. 모든 이동 에이전트는 유한한 값인 mtwo(총 연산을 수행하는 최대 시간) 내에 이주를 한다.

가정 2. 한 호스트는 한 가지 서비스만을 제공한다. 즉, 모든 이동 에이전트는 한 호스트에서 한 가지 연산만 실행한다.

보조 정리 1. 모든 중복된 이동 에이전트는 검출된다.

증명. 가정 1에 의하면 모든 이동 에이전트는 이주하므로 모든 중복된 이동 에이전트도 이주하고, 그 이동 에이전트들은 위치 관리 시 모두 검출된다.

보조 정리 2. 한 호스트에서는 각 이동 에이전트의 한 복사 본에 의해서만 연산이 수행된다.

증명. a가 수행했던 호스트로 a′이 이주 시, a′은 그 호스트의 a에 관련된 플래그의 값 v를 검사한다. a′은 자신의 히스토리를 참조하여 이 호스트 방문 횟수를 계산해 내고, v와 비교하여 다르면 그 호스트에서 실행을 하지 않고 이용 가능한 다음 번 호스트로 이주한다. 호스트를 방문하여 연산을 실행하지 않으면 히스토리에 이 호스트가 추가되지 않으므로, 이 호스트를 방문하여 연산을 실행한 단 하나의 이동 에이전트 복사 본만이 이 호스트를 다시 방문할 수 있다.

보조 정리 3. 이동 에이전트는 자신이 방문했던 호스트를 다시 방문하여 연산을 수행할 수 있다.

증명. 히스토리가 $H_1 \rightarrow H_2 \rightarrow \cdots \rightarrow H_n$ 인 이동 에이전트 a가 호스트 H_i를 (단, $1 \leq i < n$) 다시 방문하게 되면, H_i에 있는 a에 관련된 플래그의 방문 횟수와 a의 히스토리 상의 H_i 등장 횟수가 같으므로 a는 H_i에서 작업을 수행할 수 있다.

보조 정리 4. 중복된 이동 에이전트들이 수행한 연산들은 어떤 순서로 취소되어도 임의의 단계에서의 연산 결과는 같다.

증명. 식별자로 구분이 되는 이동 에이전트들이 하나의 호스트에서 수행한 같은 연산들은 역시 구분이 되므로, 호스트에서 연산 취소 시에 실제로 그 이동 에이전트가 실행한 연산들을 취소할 수 있다. 그리고 보조 정리 2에 의해서, 식별자로 구분이 되지 않는 이동 에이전트들이 하나의 호스트에서 같은 연산을 수행할 수 없고, 가정 2에 의해서 한 이동 에이전트가 호스트에서 수행한 연산들은 동일하므로, 한 호스트에서 하나의 이동 에이전트가 수행한 연산들의 결과는 연산 순서와 상관없다.

정리 1. 이동 에이전트는 단일 수행 특성을 보장하면서 순환적 작업을 완결할 수 있다.

증명. 첫 번째로 이동 에이전트의 작업을 이루는 연산들이 모두 다를 때, 보조 정리 1과 보조 정리 4에 따라 이동 에이전트가 수행한 연산들 중에서, 한 이동 에이전트 복사 본이 수행한 연산들을 제외한 모든 중복 연산을 취소할 수 있다. 그러므로 모든 연산들은 정확히 한번씩 수행된다. 두 번째로 연산들이 모두 같을 때, 보조 정리 3에 따라 그 연산들은 인디 수행되므로 복귀월산 문제가 발생하지 않는다. 따라서, 같은 연산들과 다른 연산들의 모임으로 이루어진 일반적인 연산들은 모두 단일 수행된다.

7. 결론 및 향후 연구

비멱등 연산 수행을 지원하는 기존의 TRBA는 이주 경로에 순환이 있을 경우 연산을 무시하는 복귀월산 문제가 발생하여, 이동 에이전트의 작업을 완결하지 못하였다. 본 논문에서 제안한 결함 포용 기법은 MARS와 홈 MARS 레지스터의 이동 에이전트 위치 정보에 히스토리 필드와 파손 필드를 두고, 방문 횟수를 기록한 플래그를 각 호스트에 둠으로써 복귀월산 문제를 해결하여 이동 에이전트의 단일 수행 특성을 보장한다.

향후 연구 과제로서, 제안 기법과 현재 연구 중인 이동 에이전트의 이주 자율성을 위한 메시지 전달 프로토콜을 결합시켜서 ODDUGI 이동 에이전트 시스템에 구현하고자 한다.

참고 문헌

[1] D. Milojicic et al., "MASIF: the OMG Mobile Agent System Interoperability Facility," *Proc. Mobile Agents*, September 1998, pp. 50-67.

[2] S. Pleisch and A. Schiper, "Approaches to Fault-Tolerant Mobile Agent Execution," Technical Report rz 3333, IBM Research, January 2001.

[3] L.M. Silva, V. Batista, and J.G. Silva, "Fault-Tolerant Execution of Mobile Agents," *Proc. Int. Conf. Dependable Systems and Networks*, June 2000, pp. 135-143.

[4] M. Dalmeijer, E. Rietjens, M. Soede, D.K. Hammer, and A.T.M Aerts, "A Reliable Mobile Agents Architecture," *Proc. 1st IEEE Int. Symp. on Object-oriented Real-time distributed Computing*, IEEE Computer Society Press, 1998, pp. 64-72.

[5] A. Mohindra, A. Purakayastha, and P. Thati, "Exploiting Non-determinism for Reliability of Mobile Agent Systems," *Proc. Int. Conf. Dependable Systems and Networks*, June 2000, pp. 144-153.

[6] L.M Silva et al., "JAMES: A Platform of Mobile Agents for the Management of Telecommunication Networks," *Proc. 3rd Int. Workshop on Intelligent Agents for Telecommunication Applications*, August 1999, pp. 76-95.

[7] ODDUGI Mobile Agent System, http://oddugi.korea.ac.kr, 2004.