분산 시스템에서의 복잡한 사건/상태의 결함 허용 분산 탐지

심 영 철†

요 약

분산 시스템은 높은 성능, 결함 허용성, 정보와 자원의 공유 등을 이룰 수 있는 환경을 제공한다. 그러나 분산 시스템 내에서 발생하는 사건과 상태들을 적절히 관리하지 못하면 이러한 장점의 혜택을 받을 수 없게 된다. 이러한 사건과 상태들은 성능 저하, 동작 오류, 수상한 행위 등의 징후가 될 수 있으므로 자세히 분석되어야 한다. 사건/상태를 적절히 관리하려면 이들을 명세하고 효율적으로 탐지할 수 있어야 한다. 본 논문에서는 사건/상태의 중앙 집중 탐지 알고리즘에 대해 설명한다. 다음 계층적 구조를 갖는 분산 시스템에서 이 탐지 알고리즘을 분산화 하는 방안에 대해 설명한다. 분산 알고리즘은 사건/상태 탐지 임무를 부임무들로 분해하는 과정과 부임무들을 적절한 노드들에 할당하는 과정으로 구성된다. 또 이 분산 탐지 알고리즘이 결함 허용성을 갖도록 하는 방안에 대해서도 설명한다.

Fault-Tolerant, Distributed Detection of Complex Events and States in Distributed Systems

Young-Chul Shim†

ABSTRACT

Distributed systems offer environments for attaining high performance, fault-tolerance, information sharing, resource sharing, etc. But we cannot benefit from these potential advantages without suitable management of events/states occurring in distributed systems. These events and states can be symptoms for performance degradation, erroneous functions, suspicious activities, etc. and are subject to further analysis. To properly manage events/states, we need to be able to specify and efficiently detect these events/states. In this paper we first describe an event/state specification language and a centralized algorithm for detecting events/states specified with this language. Then we describe an algorithm for distributing an event/state detection task in a distributed system which is hierarchically organized. The algorithm consists of decomposing an event/state detection task into subtasks and allocating these subtasks to the proper nodes. We also explain a method to make the distributed detection fault-tolerant.

※이 연구는 94년도 한국과학재단 연구비 지원에 의한 결과임.
(과제 번호:KOSEF 941-0900-011-2)

† 정 회 원:홍익대학교 컴퓨터 공학과

논문접수:1997년 3월 19일, 심사완료:1997년 5월 21일

1. 서 론

분산 시스템은 높은 성능, 고장 허용, 정보와 자원의 공유 등을 이룰 수 있는 환경을 제공한다. 그러나 분산 시스템을 사용할 때는 여러 가지로 바람직하지 못한 사건이 발생하거나 바람직하지 못한 상태가 지속되어 분산 시스템이 제공할 수 있는 장점을 효율적으로 활용할 수 없는 경우가 생길 수 있다. 분산 시스템의 행태를 잘 이해하기 위해 또는 다른 사용자의 행위와 동기(synchronize)하기 위해 특정한 사건이나 상태를 관찰하기를 원할 수 있다.

사건의 예로는 컴퓨터에 침투하려는 시도나 어떠한 사용자가 컴퓨터에 로그인 하는 것 등이 있다. 상태의 예로는 네트워크 과잉밀집현상(congestion)이나 호스트 과부하(overloading) 등을 들 수 있다.

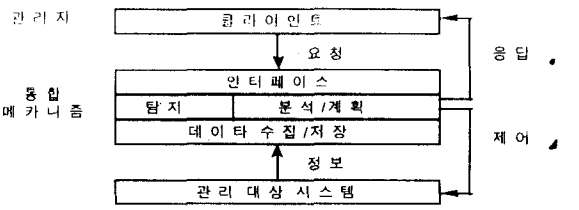
분산 시스템을 효율적으로 사용할 수 있기 위해서는, 이러한 사건/상태들을 탐지(detect)하고 관리하는 것이 매우 중요하다. 만약 탐지된 사건/상태가 컴퓨터에 침투 시도, 네트워크 과잉밀집 현상, 호스트 과부하 등과 같이 바람직하지 못한 것이라면, 이러한 사건/상태를 신속히 탐지하고 이러한 문제점들을 해결할 수 있는 대책을 세우는 것이 중요하다. 또 우리가 만약 어떠한 특정한 사용자가 특정한 컴퓨터에 로그인 하자마자 연락을 취하고 싶다면, 이 사용자가 로그인하는 사건/상태가 발생하자마자 이에 대한 보고를 받고 싶을 것이다. 이러한 목적으로 성능 관리자, 보안 관리자, 결함 관리자 등의 다양한 관리자들이 개발되고 있다. 이러한 관리자들의 서로 다른 관점에서의 관리 기능을 제공하지만 기본적으로는 같은 메카니즘을 사용하고 있다. 이들 기본 메카니즘들은 다음과 같다.

(1) 데이타의 수집과 저장: 관리 대상 분산 시스템을 계속적으로 모니터링함으로써 탐지, 분석, 관리 동작의 계획 및 수행에 필요한 데이타를 수집하고 저장한다.
(2) 사건/상태의 탐지: 바람직하지 못하거나 관심이 있는 사건/상태를 탐지한다.
(3) 분석: 저장된 데이타를 분석함으로써 탐지된 사건/상태의 원인을 규명한다.
(4) 관리 동작의 계획 및 수행: 요청된 관리 동작을 달성하기 위해 계획을 세우고 수행을 한다.

위와 같은 기본 메카니즘들은 (그림 1)과 같은 통합 메카니즘을 구성하고 모든 관리자들에 의해 공유될 수 있다. 관리 대상 시스템은 계속적으로 모니터링되고 데이타가 수집된다. 수집된 데이타는 이력 데이타베이스(historical database)에 저장된다. 관리자는 이 통합 메카니즘에 클라이언트가 되고 다음과 같은 기본 형식을 갖는 요청(request)을 보낸다.

사건/상태 → (가설 → 동작)

이러한 요청을 받으면 통합 메카니즘은 (i) 사건/상태를 탐지하고 (ii) 이력 데이타베이스내의 데이타를 분석하여 주어진 가설이 탐지된 사건/상태의 원인이라는 것을 증명한 후 (iii) 주어진 동작을 달성하기 위한 계획을 세우고 수행을 한다[1].



(그림 1) 관리 환경의 기본 구조
(Fig. 1) Architecture of management environment

본 논문에서는 위의 통합 메카니즘 중 사건/상태 탐지 메카니즘에 대해 기술한다. 사건/상태의 탐지를 위해서는 먼저 사건/상태 명세 언어가 필요하고 다음 이에 대한 탐지 알고리즘이 필요하다. 본 논문에서 제안된 명세 언어는 시간 논리(temporal logic)에 근거한 명세 언어로서 표현력이 우수하여 다양한 사건/상태를 편리하게 명세할 수 있으며 또 이 명세언어로 표현된 사건/상태의 탐지가 신속히 이루어질 수 있도록 간단한 구조를 갖고 있다.

분산 시스템에서의 사건/상태의 탐지를 위해서는 많은 양의 정보를 수집 분석함이 필요하므로 만약 한 노드가 이 일을 전부하게 되면 특정 노드에 대한 통신이나 계산의 부하가 너무 크게 되므로 사건/상태 탐지의 태스크를 여러 노드에 나누어 상호 협조하도

록 하는 분산 탐지 알고리즘을 개발하는 것이 중요하다. 분산 탐지 시에는 여러 노드가 탐지에 참여하게 되므로 만약 이중 하나의 노드라도 죽게 되더라도 탐지는 실패하게 된다. 그러므로 탐지에 참여한 노드의 일부가 죽게 되더라도 탐지 알고리즘이 계속 수행될 수 있도록, 분산 탐지 알고리즘은 결함을 허용하는 알고리즘이 되어야 한다.

사건/상태의 명세 및 탐지는 주로 분산 프로그램의 디버깅과 망관리의 분야에서 많이 연구되어 왔다. 분산 프로그램을 디버깅하기 위해서는 중단점(breakpoint)을 설정하고 프로그램이 이 중단점에 도달할 때까지 수행을 한 후 레지스터나 메모리의 내용을 관찰하는 방법을 쓰거나 또는 정확한 프로그램이 보여야 하는 행위를 명세한 후 실제의 프로그램을 수행한 후 실제 프로그램이 이 정확한 행위를 그대로 따라가고 있는가를 비교하는 방법을 쓴다. 중단점이나 정확한 프로그램이 보여야 하는 행위가 사건/상태로서 명세가 된다. 분산 디버깅에서 프로그램의 행위는 오직 사건/상태들의 인과 관계(causal relationship)에만 의존하므로 동기화된 전역적 시계(global clock)는 필요없으며 메시지의 지연 시간에 대해서는 아무런 가정도 하지 않는다. 이러한 상황에서 명세된 사건/상태는 정확히 탐지되어야 한다. 그러므로 분산 디버깅에서 제안된 사건/상태의 명세 언어는 확장된 정규식(regular expression)에 의거하고 있어 명세할 수 있는 사건/상태의 종류가 매우 한정된 반면 이 사건/상태가 발생하자마자 이를 즉시 탐지하는 알고리즘이 제시되고 있다[2-6].

망관리의 주목적은 관리 대상 시스템의 성능, 신뢰성, 보안을 유지하는 것이며 이를 위해 성능, 신뢰성, 보안상의 문제점이 사건/상태로 명세되고 탐지된 후 적절히 처리된다. 이를 위해서 분산 디버깅보다는 훨씬 다양한 종류의 사건/상태를 명세할 수 있어야 한다. 예를 들어 망내에서 어떠한 상태가 지속된 물리적 시간(physical time interval) 등을 명세할 수 있어야 한다. 또 이의 탐지를 위해서는 동기화된 전역적 시계가 필요하다. 또 망 관리자는 관리 대상 시스템으로부터 각자의 상태에 대한 보고를 받으며 이 상태보고 메시지의 지연 시간은 유한하다고 가정한다. 망관리에서의 사건/상태의 명세 언어는 주로 데이터베이스의 질의어에 의거하고 있어 매우 다양한 종류의 사건/상태들을 명세할 수 있다[7-10]. 반면 사건/상태가 발생하는 시각과 이를 탐지할 수 있는 시각과에는 다소 시간차가 존재하게 된다. 그러므로 사건/상태를 탐지하였을 때 이 사건/상태는 이미 끝난 후일 수도 있다. 그러나 망관리에서는 이보다 더 많은 종류의 사건/상태를 탐지할 수 있는 것이 더 중요하다.

분산 디버깅의 분야에서는 사건/상태를 명세하는 방법과 이 사건/상태의 명세가 이미 분해되어 여러 노드에 할당되었다고 가정하였을 때 이를 탐지하는 알고리즘에 대해 많이 연구하였다.

망관리 분야에서는 대부분이 데이터베이스 질의에 의거한 명세 언어를 사용하므로 사건/상태의 탐지를 위해서 분산 데이터베이스에서의 질의 처리(query processing) 기능을 사용한다고 가정하였고 사건/상태의 할당과 분해 및 탐지 기법에 대해서는 거의 언급을 하지 않고 있다.

본 논문에서 기술하는 명세 언어와 이의 탐지를 위한 알고리즘들은 망관리 분야에의 응용을 그 목적으로 한다. 그러므로 소개될 명세 언어는 분산 디버깅 분야에서의 명세 언어보다 훨씬 다양한 종류의 사건/상태를 명세할 뿐 아니라 분산 디버깅이나 망관리의 양 분야에서 거의 취급하지 않은 사건/상태 명세의 분해 및 할당을 위한 알고리즘들을 설명한다. 또 탐지 알고리즘에 결함 허용성을 추가하는 방안에 대해서도 설명한다.

본 논문은 다음과 같이 구성되어 있다. 제 2장에서는 명세 언어에 대해 설명한다. 제 3장에서는 이 언어로 표현된 사건/상태의 중앙 집중식 탐지 알고리즘에 대해 개략적으로 설명하고 제 4장에서는 탐지 알고리즘을 분산화 하는 방안에 대해 기술한다. 제 5장에서 분산 탐지에 결함 허용성을 추가하는 방안에 대해 설명하고 마지막으로 제 6장의 결론으로 끝을 맺는다.

2. 사건 및 상태의 명세

사건/상태의 탐지 알고리즘을 설계하기 위하여서는 사건/상태 명세언어의 구문(syntax) 및 의미(semantics)가 먼저 정의되어야 한다. 본 장에서는 이 명세언어의 구문 및 의미에 대해 간단히 기술한다.

사건 및 상태가 탐지될 분산 시스템은 ER(entity-relationship) 모델을 사용하여 모델링될 수 있다. 객

체(entity)는 호스트(host), 프로세스(process), 메세지(message) 등을 나타내며 각 객체는 애트리뷰트(attribute)와 오퍼레이션(operation)으로 특징지어진다. 예를 들어 프로세스 객체는 create, destroy, stop, resume 등과 같은 오퍼레이션을 갖고 process-id, host-name, owner, command, status 등과 같은 애트리뷰트를 갖는다. 객체들 사이에는 관계성(relationship)이 존재한다. 예를 들어 호스트 객체와 프로세스 객체 사이에는 process-part-of와 같은 관계성이 존재한다. (그림 2)는 분산 환경의 간단한 ER 모델의 예이다. 그림에서 사각형은 객체이다. 관계성은 1 대 1, 1 대 n, n 대 m이 될 수 있고 다이아몬드로 나타내져 있다.

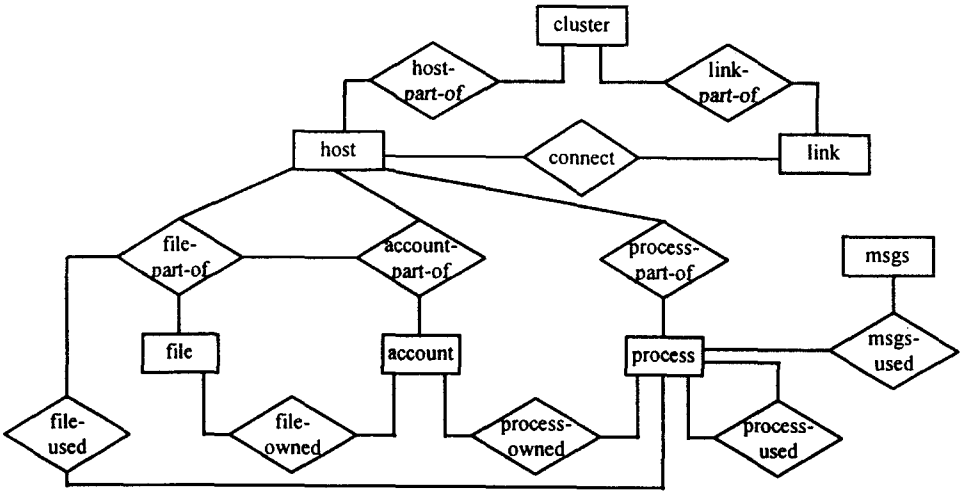
분산 환경 내에는 프로브(probe)들이 심어져 있어 프로세스의 생성과 같은 단순한 사건을 탐지할 수 있고 프로세스 상태와 같은 단순한 상태를 탐지할 수 있다. 복잡한 사건 및 상태는 이들 단순한 사건과 상태를 조합하여 명세되고 탐지된다.

본 장에서 기술하는 명세 언어는 망관리에서 사용되며 다음과 같은 다양한 종류의 사건/상태를 표현할 수 있다.

(1) 객체에서 지속되는 상태와 이의 지속 시간
(2) 여러 가지 다른 상태들의 임의의 논리적 조합
(3) 특정한 상태를 만족하는 객체들에 대한 통계치
(4) 객체들에서 지속되는 상태의 시간상 공간상의 통계치
(5) 객체에서 발생되는 사건과 이의 발생 시간
(6) 여러 가지 다른 상태들과 사건들의 시간상의 발생순서를 포함한 임의의 논리적 조합

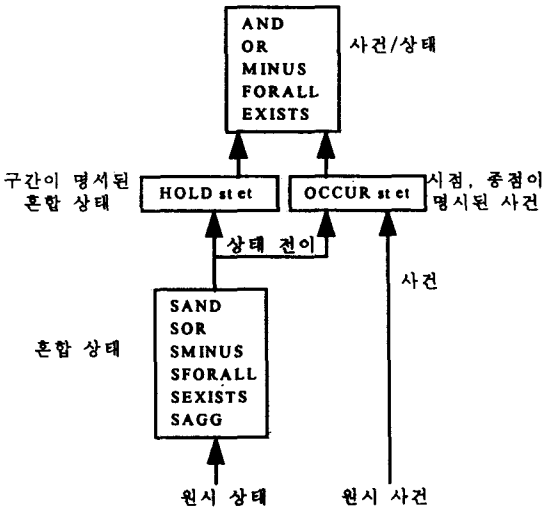
사건/상태 명세의 일반적인 구조는 (그림 3)과 같다. 사건/상태의 명세는 연산자(operator)에 의해 결합된 피연산자(operand)들의 모임으로 볼 수 있다. 두 가지 종류의 피연산자가 있다. 첫 번째 종류의 피연산자는 객체의 상태나 객체들 사이의 관계성으로 나타내지는 시스템 내에 유지되는 원시 상태(primitive state)이다. 원시 상태의 피연산자들은 SAND, SOR, SMINUS, SFORALL, SEXISTS, SAGG와 같은 연산자들에 의해 혼합상태(compound state)를 이룬다.

혼합 상태는 분산 시스템 내의 여러 객체들 사이에 동시에 유지되고 있는 상태를 나타내나 아직 이 혼합상태가 계속적으로 유지되는 구간은 명시가 되지 않는다. 두 번째 종류의 피연산자는 시스템 내에서 발생하는 원시사건으로서 언제 이 원시 사건이 시작 종료되는지에 대한 시점(start time), 종점(end time)의 정보는 명시되지 않는다. 혼합상태와 원시사건의 피연산자에 대한 시점과 종점에 대한 정보는 HOLD와 OCUR의 두 연산자에 의해 명시된다. HOLD 연산자는



(그림 2) 분산 환경의 ER 모델
(Fig. 2) An ER model of a distributed environment

혼합상태와 이 혼합상태가 지속되는 구간의 시점과 종점을 매개변수(parameter)로 받는다. OCCUR 연산자는 상태전이(state transition)를 나타내기 위해서는 혼합상태를, 사건의 발생을 나타내기 위해서는 원시 사건을 매개변수로 갖는다. 또, 상태전이나 사건이 발생한 시점과 종점을 표시한다. HOLD와 OCCUR 연산자에 의해 구성된 표현(expression)을 각각 시간한정 상태(temporally qualified state)와 시간한정사건(temporally qualified event)이라 부른다. 이 시간한정상태와 시간한정사건들은 AND, OR, MINUS, FORALL, EXISTS 등의 연산자에 의해 조합되어 탐지하고자 하는 사건/상태를 구성한다. 이 연산자를 이용하여 사건/상태를 표현할 때는 구성요소사건, 상태들 사이의 시간 관계를 명시할 수 있다.



(그림 3) 사건/상태 언어의 기본 구조

(Fig. 3) The general structure of an event/state specification

먼저 기본 피연산자들의 예를 보인다. 기본 피연산자들의 구문(syntax)과 의미(semantics)는 OPS5의 좌측 표현(left-hand side expression)과 흡사하다.

```
(1)(host (name ernie) (la ?x > 5))
(2)(host-part-of (hostname mars) (clustername xcssun))
(3)(host (name ernie) ((la 5) > 10))
(4)(host (name ?x) ((la max 5) > 10))
(5)(login (hostname arpa) (acctname john))
```

예 (1)은 ernie라는 호스트의 부하 평균(load average)이 5보다 큰 상태를 나타낸다. 여기 ?x는 변수로서 만약 부하 평균이 5보다 크면 이 때의 부하 평균값이 변수 ?x의 값으로 복귀된다. 예 (2)는 호스트 mars와 xcssun이라 불리우는 클러스터(cluster) 사이에 host-part-of와 같은 관계성이 존재하는 상태를 나타낸다. 예 (3)에서는 ernie라 불리우는 호스트의 5초전의 평균부하가 10 보다 컸던 과거 상태를 나타낸다. 예 (4)에서는 ?x의 변수로 주어진 어떤 호스트의 최근 5초간의 최대 평균 부하가 10 보다 컸던 집단화된 상태를 나타낸다. 예 (5)는 john이 arpa에 로그인하는 원시 사건을 나타낸다.

명세언어를 사용하여 복잡한 사건/상태를 정의한 예는 다음과 같다.

```
(1)(SFORALL (?x)
   (host-part-of (hostname ?x) (clustername sun))
   (host (name ?x) (la > 5)))

(2)(AND
   HOLD ?t1 ?t2 (host (name ?x) (la > 5)))
   (AND
        (OCCUR ?t1, ?t3 (process-creation (hostname ?x)
                        (creator ?y) (command troff)))
        (OCCUR ?t2, ?t4 (process-creation (hostname ?x)
                        (creator ?y) (command troff)))
        (?t1 < ?t2 ≤ ?t1 + 10)))

(3)(HOLD ?t ?t + 10
   (SEXISTS (?v, ?y)
        (SAND
             (SAGG () (?y=count(?x)) (link (name ?x)))
             (SAGG () (?v=count(?u)) (link (name ?u)
                  (delay > 5)))
             (?v/?y > 0.6))))
```

예 (1)에서는 두 개의 원시 상태가 SFORALL이라는 연산자에 의해 혼합 상태를 이루고 있다. 이 혼합상태는 SUN이라는 워크스테이션 클러스터에 속하는

모든 호스트의 평균 부하가 5를 넘는 과부하 상태를 표현한다. 예 (2)는 시간 한정 상태와 시간 한정 사건이 AND에 의해 조합된 사건/상태의 예로서 평균 부하가 계속적으로 5를 넘고 있는 과부하된 호스트에 동일한 사용자가 2개의 troff라는 워드프로세싱 프로세스를 생성한 예이다. 예 (3)에서는 링크 지연 시간(delay)이 5가 넘는 링크의 수가 전체 링크 수의 60%를 넘는 상태가 10분 이상 지속되는 상태를 나타내고 이는 네트워크에서의 과잉 밀집 현상을 나타낸다.

3. 사건 및 상태의 중앙집중 탐지

본 장에서는 먼저 사건/상태의 중앙집중 탐지 알고리즘에 대해 기술한다. 탐지 알고리즘에 대해 기술하기 전에 탐지 알고리즘의 적합한 정책을 결정하는데 있어서 고려되어야 할 중요한 문제점 들에 대해 기술한다. 각각의 문제점에 대해서는 몇가지 가능한 해결책들이 있으며 이 해결책들 중에서 한가지를 선택하여야 한다. 고려될 수 있는 문제점들 중 가장 중요한 것들은 (1) 탐지 지연(detection delay)과 (2) 탐지 트리거링(detection triggring)이다.

(1) 탐지 지연 : 사건/상태를 탐지하기 위해서는 정보를 수집하고, 이 정보를 탐지 알고리즘을 수행하는 노드로 전송하고, 이 노드는 전송된 정보를 사용하여 사건/상태의 명세가 부합되는지를 계산한다. 이렇게 여러 단계의 일들이 수행되므로 어떠한 사건/상태가 실제로 발생된 시각과 이 사건/상태가 탐지되는 시각 사이에는 어느 정도의 시간 차이가 존재하게 된다. 이 시간 차를 탐지 지연이라 한다. 탐지 지연이 얼마나 긴가에 따라 탐지 알고리즘의 정책은 즉시 탐지(immediate detection)와 지연된 탐지(delayed detection)로 분류된다. 즉시 탐지의 경우에는 사건/상태가 발생하는 즉시 새로운 사건이 발생하거나 상태가 변하기 전에 사건/상태를 탐지하게 된다. 지연된 탐지의 경우에는 사건/상태가 발생한 후 이 사건/상태가 탐지되게 되는데 사건/상태의 발생과 탐지 사이에 다른 사건이 발생하거나 상태가 변할 수 있다[11].

(2) 탐지 트리거링 : 이 문제점은 언제 사건/상태의 탐지가 트리거링 되는가에 관한 것이다. 탐지는 시스템 내의 사건 발생이나 상태 변화와는 비동기적으로 단순히 주기적으로(periodically) 트리거링될 수 있다. 또는 탐지는 프로브로부터 데이타가 도착하면 트리거링될 수 있다. 처음의 경우를 시간-구동(time-driven) 탐지라 하고 두 번째의 경우를 데이타 구동(data-driven) 탐지라 한다[12].

본 연구에서는 지연된 탐지의 정책과 데이타 구동 탐지의 정책을 선택하였다. 즉시 탐지가 가능하기 위해서는 다음의 기본 조건들이 만족되어야 함이 알려졌다[11].

(1) 탐지자(detector)는 탐지될 사건/상태의 임계 구성요소(critical component)에 위치하여야 한다. 사건/상태의 임계 구성 요소는 이 구성 요소에서의 원시 사건의 발생이나 원시 상태의 변화가 사건/상태 명세의 값을 FALSE에 TRUE로 변화시키는 구성요소이다.

(2) 임계 구성 요소에서 원시 사건이 발생하거나 원시 상태의 변화가 있을 때 임계 구성 요소에는 사건/상태의 탐지에 필요한 모든 정보가 이미 존재하여야 한다.

(3) 구성 요소의 정보는 무효화되지 않아야 한다.

여러 연구자들이 즉시 탐지의 알고리즘에 대해 기술하고 있다[3, 5, 6]. 그러나 위의 세 가지 조건을 만족하여야 하므로 제안된 알고리즘에 의해 탐지될 수 있는 사건의 종류는 매우 제한되어 있고 이들의 결과는 분산 디버깅의 분야에 응용된다. 그러나 본 연구의 주 응용 분야는 망관리이므로 매우 한정된 사건/상태를 즉시 탐지하는 대신에 탐지 지연 시간을 어느 정도 허용하여 광범위한 사건/상태를 탐지하는데 초점을 두었다. 일단 지연된 탐지의 정책이 선택된 후에는 탐지에 관련된 지연 시간을 최소화하는 것이 대단히 중요하다.

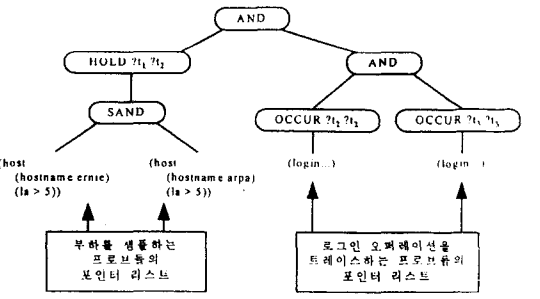
시간 구동 탐지의 정책이 사용되는 경우에는 사건 상태 탐지기에 프로브로부터에서 데이타가 도착하는 시각과 시간 인터럽트에 의해 탐지기가 트리거링되어 이 데이타를 사용하여 사건/상태 명세를 계산하는 시각 사이에는 불필요한 시간 지연이 발생하게 된다.

그러나 데이타 구동 탐지 정책의 경우에는 데이타가 도착하자마자 탐지가 시작되므로 시간 구동 탐지 정책에서와 같은 불필요한 시간 지연은 발생하지 않는다. 더구나 데이타 구동 탐지의 경우에는 이전에 받은 데이타를 처리하여 얻은 부분적인 탐지 결과를 저장할 수 있다. 그러므로 새로운 데이타가 도착하면 이 새 데이타만 처리하면 된다. 이러한 유형의 탐지 기술을 상태 저장 탐지 기술(state-preserving detection technique)이라 하고 탐지 시간을 많이 줄일 수 있다[13]. 이러한 이유에서 본 연구에서는 데이타 구동 탐지 정책을 사용하였다.

다음 제 2 장에 설명된 언어로 명세된 사건/상태를 중앙 집중 방식으로 탐지하는 알고리즘에 대해 개략적으로 기술한다. 이 알고리즘은 OPS5 규칙 기반 언어에 사용되는 RETE 알고리즘에 기초를 두고 있다[14]. 설명을 위해서 사건/상태의 명세는 (그림 4)와 같이 트리로 나타낼 수 있다.

이 트리에서 잎 노드(leaf node)는 원시 상태나 원시 사건과 같은 피연산자이고 내부 노드(inner node)는 연산자를 나타낸다. 오퍼레이션, 객체 상태, 관계성 상태를 위한 프로브의 각 유형 별로 해당하는 피연산자에로의 포인터를 저장하는 리스트가 유지된다. 그림에서 부하를 샘플링하는 프로브 유형에 한 개의 포인터 리스트가 유지되고 로그인 오퍼레이션을 트레이스하는 프로브 유형에 한 개의 포인터 리스트가 유지된다. 만약 이들 프로브로부터 받은 데이타가 특정 피연산자가 부합될 것인가 또는 안될 것인가의 결정에 영향을 미칠 수 있다면 그 피연산자에로의 포인터가 존재한다. 특정 프로브로부터 새로운 데이타가 도착하면 이 프로브 유형의 포인터 리스트에 의해 가리켜지고 있는 피연산자의 부합 여부가 이 새 데이타를 사용하여 계산된다. 만약 어떤 피연산자의 진리값(truth value)에 변화가 있게 되면 이 변화는 명세 트리를 따라 최대한 올라가게 된다. 부분 탐지 결과는 중간 노드에 저장되므로 전에 받은 데이타는 다시 처리할 필요가 없다. 포인터 리스트 메카니즘을 사용함으로써 사건/상태 명세의 계산에 영향을 줄 수 있는 데이타만을 처리하게 되고 데이타에 의해 영향을 받는 사건/상태의 명세만을 처리하게 된다.

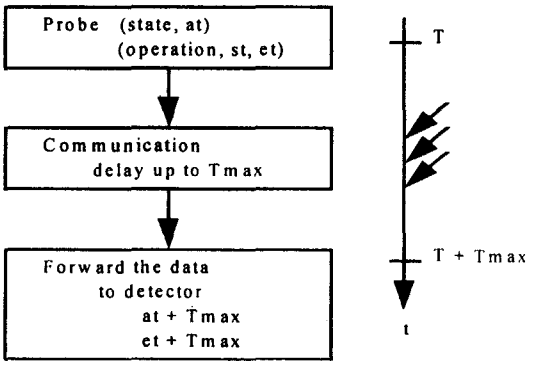
탐지 알고리즘을 개발함에 있어서는 다음의 가정들을 세웠다. 이들의 가정은 분산 디버깅의 분야에서는 가능하지 않으나 망관리의 분야에서는 적절하다.



(그림 4) 중앙 집중식 사건/상태 탐지의 전체 구성
(Fig. 4) The overall structure for centralized event/state detection

(1) 동기화가 잘 된 전역 시계(global clock)가 있어 프로브로부터 받은 데이타에 포함되어 있는 시간 정보를 사용하여 사건/상태들의 순서와 시간 구간 등을 결정할 수 있다.[15]
(2) 균일하지는 않지만 작고 또 최대 값이 Tmax로 주어진 유한의 통신 지연 시간이 존재한다.
(3) 프로브로부터의 데이타는 이 데이타가 수집된 순서에 따라 탐지기에 전달된다. 이 가정은 수집 시간이 t인 데이타는 t +Tmax에 탐지기에 전달함으로써 가능해 진다.

(그림 5)에서 이 가정들이 설명되어 있다. 그림에서 상태를 샘플 하는 프로브는 샘플된 상태 값에 at라는 샘플된 시간을 첨부하고, 원시 사건인 오퍼레이션을

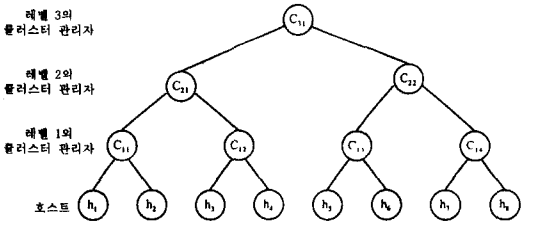


(그림 5) 탐지 알고리즘의 가정
(Fig. 5) Assumptions for the detection algorithm

트레이스하는 프로브는 이 오퍼레이션 발생의 시점과 종점인 st와 et의 값을 첨부한다. T라는 시각에 샘플된 상태의 값이나 T라는 시각에 종료되어 트레이스된 사건들은 T +Tmax 이전에 탐지기가 있는 노드에 도착하고 이 값들은 모두 T +Tmax에 탐지기에 보내져 처리된다.

4. 탐지 알고리즘의 분산화

매우 큰 분산 시스템이나 컴퓨터 네트워크를 효율적으로 관리하기 위하여 시스템을 (그림 6)과 같이 계층 구조(hierarchy)로 구성한다. 호스트 h_1과 h_2는 하나의 클러스터를 이루고 이 클러스터는 C11에 의해 관리된다. 계속적으로 C_{11}과 C_{12}는 레벨 2의 클러스터인 C_{21}에 의해 관리된다. 본 논문에서는 클러스터의 이름과 클러스터 관리자의 이름을 혼용한다. 즉 C_{11}은 레벨 1의 클러스터를 의미할 수 있고 또는 이 클러스터의 관리자를 의미할 수도 있다.



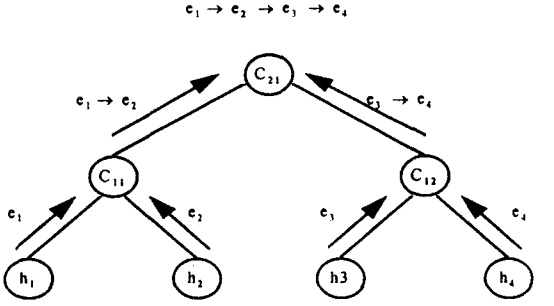
(그림 6) 계층 구조를 갖는 분산 시스템

(Fig. 6) A distributed system with a hierarchical organization

지금 만약 다음과 같은 사건/상태를 탐지한다고 하자.

$e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4$

여기서 e_i는 h_i에서 발생하는 사건/상태이고 $e_i \rightarrow e_j$는 e_i가 발생한 후 e_j가 발생함을 표현한다. 이 사건/상태는 C_{11}과 C_{12}의 두 클러스터에 걸쳐 발생하는 사건/상태이다. 이 사건/상태를 분산화된 방법으로 탐지하기 위해서 h_i는 e_i를 각각 탐지하고 C_{11}은 h_1과 h_2로부터 e_1과 e_2의 보고를 받은 후 $e_1 \rightarrow e_2$를 탐지한다. C_{12}는 h_3와 h_4로부터 e_3과 e_4의 보고를 받은 후 $e_3 \rightarrow e_4$를 탐지한다. 최종적으로 C_{21}은 C_{11}으로부터 $e_1 \rightarrow e_2$를 C_{12}로부터는 $e_3 \rightarrow e_4$의 발생에 대한 보고를 받은 후 $e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4$이 발생했음을 탐지한다. (그림 7)은 분산 탐지를 위해 필요한 호스트와 관리자들 사이에 교환되는 데이타를 보여준다.



(그림 7) 분산 탐지의 예

(Fig. 7) An example of distributed detection

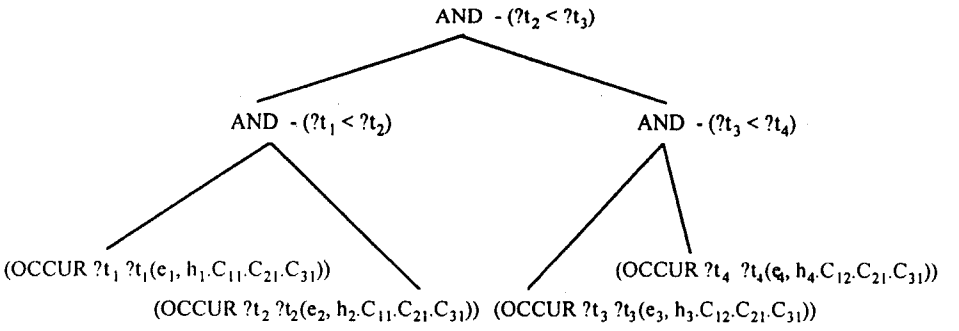
본 장에서는 $e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4$와 같이 여러 클러스터에 걸쳐 발생하는 사건/상태의 탐지를 분산화하는 알고리즘에 대해 설명한다. 일반적으로 어떠한 태스크의 분산화는 다음과 같은 세 가지 과정을 거치게 된다[16].

(1) 명세(specification): 수행되어야 할 태스크를 명세 언어로 표시한다.
(2) 분해(decomposition): 명세된 태스크를 여러 노드에 나누어 수행시킬 수 있도록 여러 개의 부태스크로 나눈다.
(3) 할당(allocation): 분해된 각각의 부태스크를 적절한 노드에 할당한다. 이때 부태스크를 할당받은 노드들 사이에 상호 협조(cooperation)를 위한 통신 구조가 명확히 정의되어야 한다. 통신 구조에서는 누가, 언제, 어떤 내용을, 어떤 형식으로 누구에게 전달하는가를 정의한다.

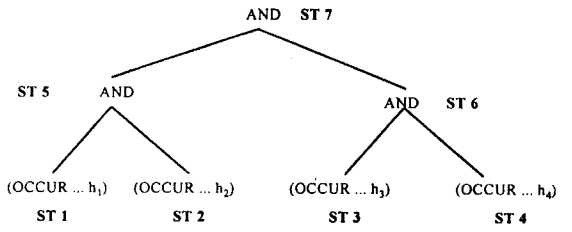
예를 들어 $e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4$는 제 2 장에서 소개된 언어를 사용하여 트리형식으로 나타내면 (그림 8)과 같이 명세 된다.

다음 이 명세는 (그림 9)와 같이 분해될 수 있다. 여기서 STi는 각각 부태스크를 표시한다.

다음 이 부태스크들은 다음과 같이 할당될 수 있



(그림 8) 사건/상태의 명세
(Fig. 8) Specification of event/state



(그림 9) 사건/상태 분해
(Fig. 9) Decomposition of an event/state

다.: (ST1 → h_1), (ST2 → h_2), (ST3 → h_3), (ST4 → h_4), (ST5 → C_{11}), (ST6 → C_{12}), (ST7 → C_{21}).

분산 탐지 알고리즘에 대해 기술하기 전에 몇 가지 규칙과 용어에 대해 설명한다. 먼저 호스트와 클러스터 관리자의 이름에 대한 규칙에 대해 설명한다. 각 이름은 '.'에 의해 분리된 구성 요소 이름들의 집합(concatenation)으로 이루어지며 예는 다음과 같다.

(1) $h_1.C_{11}.C_{21}.C_{31}$: $C_{11}.C_{21}.C_{31}$ 클러스터 내의 호스트 h_1
(2) $C_{21}.C_{31}$: C_{31} 클러스터 내의 레벨 2 클러스터 C_{21}
(3) ?h. $C_{11}.C_{21}.C_{31}$: $C_{11}.C_{21}.C_{31}$ 클러스터 내의 임의의 호스트 ?h. ?h는 변수임
(4) ?h.?C_1 : 임의의 레벨 1 클러스터 ?C_1에 속한 임의의 호스트 ?h

사건/상태 명세의 가장 기본이 되는 원시 상태와 원시 사건에는 이 상태가 유지되거나 이 사건이 발생되는 위치에 대한 정보를 나타내는 상수나 변수가 적어도 하나가 있다. 이들을 각각 위치 상수(location constant)와 위치 변수(location variable)라 한다. 이의 예는 다음과 같다.

(1) (host (hostname ernie (la > 5)) : ernie는 위치 상수
(2) (host (hostname ?x) (la > 5)) : ?x는 위치 변수
(3) (send-message (from-host ?h) (to-host venus) ...) : ?h는 위치 변수이고 venus는 위치 상수
(4) (host-part-of ?h $C_{11}.C_{21}.C_{31}$) : ?h는 위치 변수이고 $C_{11}.C_{21}.C_{31}$은 위치 상수
(5) (level1-part-of ?C $C_{21}.C_{31}$) : ?C는 위치 변수이고, $C_{22}.C_{31}$은 위치상수

각 위치 상수나 변수는 두 개의 애트리뷰트를 가진다. 이 두 애트리뷰트는 단위(unit)와 범위(range)이다. 단위란 위치 상수나 변수가 지칭하는 것이 호스트인가, 레벨 1의 클러스터인가 등을 의미하고 범위란 이 상수 또는 변수가 속하는 최소 크기의 잘 정의된 클러스터(well-defined cluster)이다. 위의 관계성으로 명세된 원시 상태를 살펴보면 (4)의 예에서 ?h의 단위는 호스트이고 이 호스트는 $C_{11}.C_{21}.C_1$에 속하므로 범위는 $C_{11}.C_{21}.C_{31}$이다. (5)의 예에서 ?C의 단위는 레벨 1 클러스터이고 ?C의 범위는 $C_{22}.C_{31}$이다.

(host (hostname ?x) (la > 5))

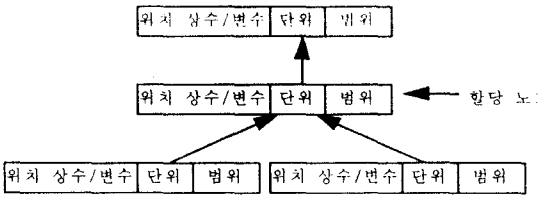
의 예에서 위치 변수 ?x의 단위는 호스트이고 이 호스트가 속하는 범위에 대해서는 아무런 제약이 없으므로 ?x의 범위는 특별한 상수인 undecided라는 값을 갖는다.

다음의 두 절에서는 사건/상태의 분할과 할당에 대해 기술한다.

4.1 사건 및 상태 명세의 분해

사건/상태의 명세를 분해하는 도중 (그림 10)과 같은 위치 정보 트리(LIT: Location Information Tree)를 생성한다. LIT의 각 노드는 위치 노드라 불리우고 위치 상수나 변수의 이름, 이의 단위, 이의 범위에 대한 정보를 포함한다.

부모 위치 노드와 자식 위치 노드는 양방향 링크(bidirectional link)에 의해 연결되어 있다. 사건/상태 명세 트리의 노드인 명세 노드에는 LIT내의 위치 노드가 한 개 대응된다. 위치 노드 내의 위치 상수/변수 이름과 단위는 이 위치 노드에 대응 하는 명세 노드가 할당될 호스트나 클러스터를 의미하며 범위는 이 호스트나 클러스터가 어떠한 클러스터에 속하는 가를 의미한다. 분해가 끝나고 할당이 행해질 때 명세 노드는 이 대응하는 위치 노드 내의 정보를 이용하여 적절한 호스트나 클러스터 관리자에 할당된다. LIT는 (그림 11)과 같은 분해 알고리즘을 통해 생성된다.



(그림 10) 위치 정보 트리
(Fig. 10) Location information tree

분해 알고리즘은 명세 트리의 루트로부터 시작하여 순환적으로 잎 노드까지 내려간 후 LIT는 잎 노드로부터 생성되기 시작하여 상위의 명세 노드로 올라가며 점점 더 큰 위치 정보 트리가 생성된다. (그림 11)의 알고리즘에 대해 자세히 설명하면 다음과 같다.

build-leaf-node-location-information-tree는 명세 트리의 잎 노드에 대한 위치 노드를 생성한다. 다음과 같은 피연산자들에 대해 생성된 위치 노드는 (그림 12)와 같다.

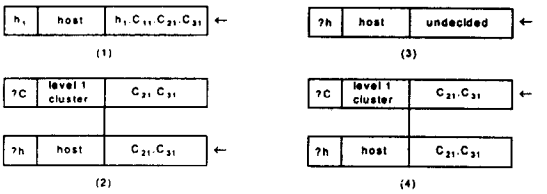
(1) (host (hostname h_1.C_{11}.C_{21}.C_{31}) ...)
(2) (login-failure (hostname ?h. ?C.C_{21}.C_{31}) ...)

```
decompose (specification node) {
    if (leaf-node)
        build-leaf-node-location-information-tree (specification node)
    else /* inner node */
        if (operand < {SAND, SMINUS, SFORALL, AND, MINUS, FORALL) {
            decompose (left child specification node)
            decompose (right child specification node)
            and-merge (left child specification node's location information tree,
                    right child specification node's location information tree)
        } else if (operand < {SOR, OR}) {
            decompose (left child specification node)
            decompose (right child specification node)
            or-merge (left child specification node's location information tree,
                    right child specification node's location information tree)
        } else /* operand < {SEXISTS, EXISTS, HOLD, OCCUR} */
            copy child specification nodes location information tree
```

(그림 11) 명세 트리 분해 알고리즘
(Fig. 11) Decomposition algorithm

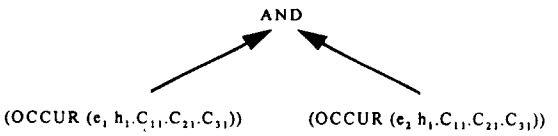
(3) (login-failure (hostname ?h) ...)

(4) (host-part-of (hostname ?h) (level1-cluster ?C.C_{21}.C_{31}))



(그림 12) 잎 노드에 대한 위치 노드의 예
(Fig. 12) Example of location nodes for leaf nodes

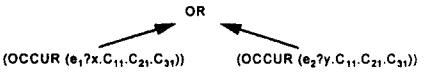
SAND, SMINUS, SFORALL, AND, MINUS, FORALL은 모두 2개의 명세 부트리를 구성 요소로 가지므로 먼저 이 명세 부트리들에 decompose 루틴이 적용되어 각각의 부트리들에 대한 위치 정보 부트리들이 생성된다. 다음 and-merge 루틴은 새로운 위치 노드를 생성하고 이 두 위치 정보 부트리들을 자식으로 함으로써 더 큰 위치 정보 트리를 생성한다. 새 위치 노드의 단위와 범위는 두 위치 부트리의 루트 내의 범위를 모두 포함하는 가장 작은 클러스터가 된다. (그림 13)과 같은 사건/상태의 명세에 대해 (그림 14)와 같은 LIT가 생성된다.



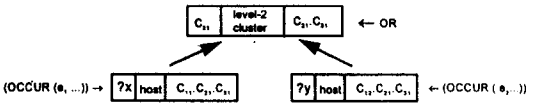
(그림 13) 한 클러스터 내의 두 호스트에서 발생하는 두 사건/상태
(Fig. 13) An event/state occurring in two hosts belonging to a single cluster

그러나 위치 변수 ?x가 각 구성 요소 위치 부트리에 공통적으로 있다면, 한 부트리 내에서의 ?x에 대한 제약은 다른 부트리로 전파되어야 한다.

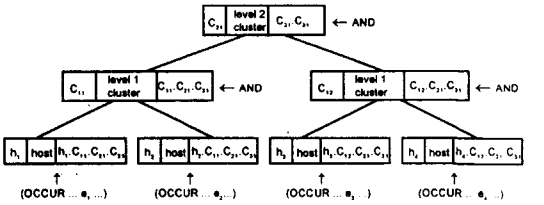
SOR와 OR도 두 개의 명세 부트리를 자식으로 가지고 decompose 루틴은 이들 부트리에 대해 각각의 LIT를 생성한다. 다음 이 두 LIT에 or-merge 루틴을 적용하여 이 두 LIT를 포함하는 더 큰 LIT를 생성한다. 이 더 큰 LIT의 루트 노드는 and-merge와 마찬가지로 형성된다. 그러나 and-merge와는 달리 공통 변수에 대한 고려가 필요 없다. (그림 16)은 (그림 15)의 명세에 대한 LIT이다.



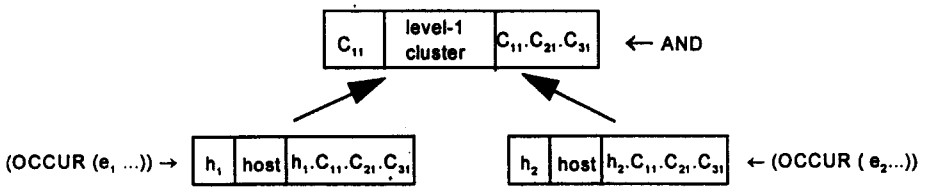
(그림 15) OR 연산자를 갖는 사건/상태
(Fig. 15) An event/state having an OR operator



(그림 16) 그림 15에 대한 LIT
(Fig. 16) An LIT for figure 15



(그림 17) 그림 8에 대한 LIT
(Fig. 17) The LIT for figure 8



(그림 14) and-merge의 예
(Fig. 14) Example of and-merge

SEXISTS, EXISTS, HOLD, OCCUR 연산자들은 한 개의 피연산자 만을 자식으로 가지므로 이 노드에 대한 위치 노드는 자식 노드의 위치 노드와 같다.

(그림 8)에 주어진 사건/상태에 대해 분해 알고리즘은 (그림 17)과 같은 LIT를 생성한다.

4.2 사건 및 상태 명세의 할당

사건/상태 명세의 분해가 끝나면 LIT를 이용하여 명세 트리의 노드들이 호스트나 클러스터 관리자들에 할당된다. 할당 알고리즘은 (그림 18)과 같다. 알고리즘은 깊이 우선(depth-first) 순서로 명세 트리의 내부의 노드를 방문한다. 명세 트리를 방문할 때 LIT내의 대응되는 위치 노드에 저장된 정보를 이용하여 명세 트리 노드를 적절한 호스트나 클러스터 관리자에 할당한다. 명세 트리 노드는 범위 애트리뷰트에 의해 주어진 클러스터 내의 단위 애트리뷰트에 명시된 유형의 모든 컴퓨터에 할당된다. 예를 들어 명세 노드가 (?x, host, $C_{11}.C_{21}.C_{31}$)이라는 위치 노드를 가진다면 이 명세 노드는 $C_{11}.C_{21}.C_{31}$ 클러스터 내의 모든 호스트들에 할당된다. 이 알고리즘은 (그림 8)의 명세 트리와 (그림 17)의 LIT에 적용하면 (그림 9)에 주어진 부태스크들이 보여진 주어진 설명과 같이 할당된다.

5. 분산 탐지의 결함 허용성

분산 탐지 알고리즘의 수행에는 여러 노드가 참여하게 되므로 이중 한 노드에 결함이 발생하여도 분산 탐지는 실패하게 된다.

결함 허용성을 위해서는 다음의 세 가지 메카니즘이 기본적으로 사용된다.

(1) 모든 사건/상태의 명세는 시스템 전체에서 유일한 식별자를 갖는다. 사건/상태 명세는 탐지기에 등록이 될 때 유일한 식별자를 할당받으며 이 명세의 분해로 생성되는 모든 부명세도 같은 식별자를 갖게된다.

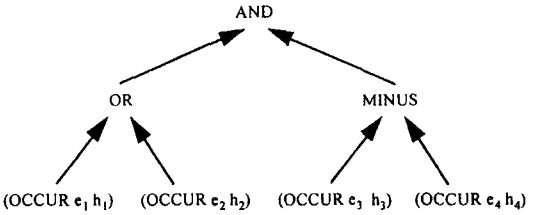
(2) 모든 레벨의 클러스터 관리자는 백업 클러스터 관리자를 갖는다. 클러스터 관리자와 백업 클러스터 관리자는 주기적으로 상대방이 정상의 상태인가를 점검한다. 만약 클러스터 관리자에 결함이 발생하면 백업 클러스터 관리자가 주 클러스터 관리자가 되며 새로운 백업 클러스터 관리자를 선택하게 된다.

(3) 호스트들이 정상 상태에 있는 가는 레벨 1의 클러스터 관리자가 주기적으로 점검한다.

위의 메카니즘을 사용하였을 때 분산 탐지 알고리즘의 과정은 다음의 세 단계로 설명될 수 있다.

(1) 초기화 : 탐지될 사건/상태의 명세가 분해 및 할당된다.

(2) 탐지 : 탐지가 수행된다.



(그림 19) 사건/상태의 명세
(Fig. 19) The specification of events/states

```
allocate (specification tree) {
    allocate the root of the specification tree;
    if (specification tree has any subtrees) {
            for all the subtrees, st, of the root
                allocate(st)
    }
}
```

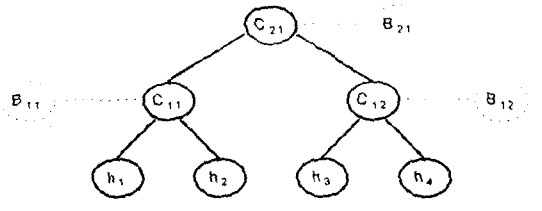
(그림 18) 할당 알고리즘
(Fig. 18) Allocation algorithm

(3) 결함 발견 및 복구 : 탐지에 참여한 노드에 결함이 발생하였음을 발견하고 이를 복구한다.

지금 (그림 19)와 같은 사건/상태를 탐지하는 경우를 고려한다. 먼저 초기화 단계에서는 각 노드에 다음과 같은 정보가 분해 및 할당 알고리즘에 의해 저장된다. 여기서 id는 사건/상태 명세의 식별자이다.

C_{21} : detect(AND (id from C_{11}) (id from C_{12}))
C_{11} : detect(OR (id from h_1) (id from h_2)) report to C_{21} with id
C_{12} : detect(MINUS (id from h_3) (id from h_4)) report to C_{21} with id
h_1 : detect(OCCUR e_1 h_1) report to C_{11} with id
h_2 : detect(OCCUR e_2 h_2) report to C_{11} with id
h_3 : detect(OCCUR e_3 h_3) report to C_{12} with id
h_4 : detect(OCCUR e_4 h_4) report to C_{12} with id

위의 정보의 의미는 다음과 같다. h_1의 경우에는 (OCCUR e_1 h_1)의 발생을 탐지하여 이를 식별자 id와 함께 C_{11}에게 알린다. C_{11}의 경우에는 h_1으로부터 id란 식별자를 가진 탐지의 통지가 오거나 h_2로부터 id란 식별자를 가진 탐지의 통보가 온 경우 이를 C_{21}에게 id와 함께 알린다. C_{21}은 C_{11}으로부터 id를 식별자로 하는 통지가 오고 또 C_{12}로부터 id를 식별자로 한 통지가 온 경우 전체적으로 사건/상태를 탐지하게 된다. 즉, 각 호스트나 클러스터 관리자에는 자신이 수행하여야 할 탐지의 부태스크와 다른 호스트나 클러스터와의 통신 구조가 저장된다. 이와 같은 내용이 (그림 20)과 같은 노드들에 저장되고 특히 C_{21}, C_{11}, C_{12}에 저장되는 내용은 이들의 백업인 B_{21}, B_{11}, B_{12}에도 저장된다.



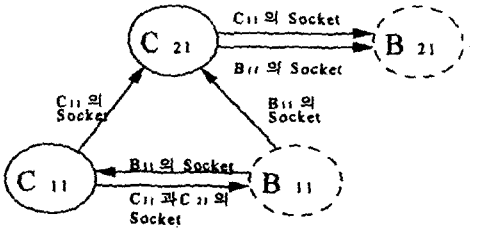
(그림 20) 결함 허용성을 부여하는 계층 구조
(Fig. 20) Hierarchical organization with backup cluster managers

제 2 단계에서 일단 탐지가 시작되면 h_1, h_2, h_3, h_4, C_{11}, C_{12}, C_{21}은 탐지를 시작하고 탐지의 부분 결과가 이들 노드에 저장된다. 백업 노드들인 B_{11}, B_{12}, B_{21}은 탐지에는 참여하지 않는다. 만약 C_{21}의 관리자에 결함이 발생한 것이 B_{21}에 의해 발견되면 B_{21}이 C_{21}의 관리자가 된 후 C_{21}이 탐지 중이던 사건/상태가 있었는지를 살펴본다. 본 장의 예에서는 id를 식별자로 하는 사건/상태의 탐지가 진행 중이었으므로 B_{21}은 이 사건/상태의 탐지에 참여했던 C_{11}, C_{12} 관리자와 h_1, h_2, h_3, h_4에게 지금까지의 id를 식별자로 한 사건/상태 탐지의 부분 결과를 모두 취소하고 새로 탐지를 시작하도록 전달한다.

만일 C_{11} 관리자에 결함이 발생하면 B_{11}은 이를 C_{11} 관리자가 탐지 중이던 id를 식별자로 하는 사건/상태의 루트를 담당하는 C_{21} 관리자에게 알리고 C_{21} 관리자는 B_{11}, h_1, h_2, C_{12} 관리자, h_3, h_4에게 지금까지의 id를 식별자로 한 사건/상태 탐지의 부분 결과를 모두 취소하고 탐지를 새로 시작하도록 명령한다.

다음에는 결함 허용성을 위해 관리자와 호스트들 사이에 어떠한 메세지가 교환되는가에 대해 설명한다.

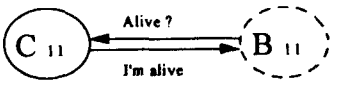
(그림 21)은 계층 구조를 만들기 위해 교환되는 메세지를 보여준다. 먼저 관리자 C_{21}과 이의 백업 B_{21}은 이미 존재한다고 가정한다. 관리자 C_{11}은 C_{21}의 소켓을 매개 변수로 받아 생성된다. C_{21}의 소켓은 C_{21}의 호스트 이름과 포트 번호로 구성되며 이 소켓을 사용하여 C_{11}은 C_{21}에게 메시지를 보낼 수 있다. C_{11}은 자신의 소켓을 만들어 이를 C_{21}에게 알리고 C_{21}은 이를 B_{21}에게 알린다. 다음 C_{11}은 자신의 백업 B_{11}을 생성한다. 이 때 C_{11}은 자신의 소켓과 C_{21}의 소켓을 매개 변수의 형태로 알려준다. B_{11}은 자신의 소켓을 생성한 후 이를 C_{11}과 C_{21}에게 알린다. C_{21}은 다시 이를



(그림 21) 계층 구조의 생성
(Fig. 21) Formation of a hierarchy

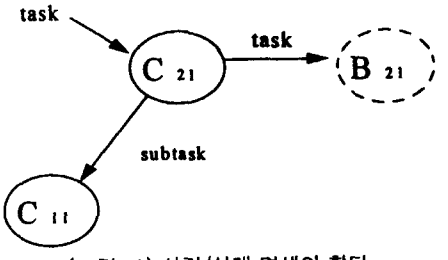
B_{21}에게 알린다. 즉 관리자와 백업은 서로 상대방의 소켓을 아는 것 외에도 하위 레벨 관리자와 백업의 소켓과 상위 레벨 주 관리자의 소켓을 알고 있다.

백업 관리자는 (그림 22)와 같이 주기적으로 주 관리자에게 폴링 메세지를 보내고 주관리자는 이에 대해 응답한다. 백업 관리자는 폴링 메세지를 보낸 후 타이머를 동작시킨다. 만약 정해진 시간 내에 응답 메시지가 없으면 다시 폴링 메시지를 보낸다. 이와 같은 시도가 정해진 횟수로 연속 실패하면 주 관리자에 결함이 발생했다고 판단하다. 주 관리자는 백업 관리자를 생성한 후 또는 폴링 메세지를 받은 후 타이머를 동작시킨다. 만약 정해진 시간 내에 폴링 메세지가 없으면 타이머를 다시 동작시킨다. 이와 같은 시도가 정해진 횟수로 연속 실패하면 백업 관리자에 결함이 발생했다고 판단한다.



(그림 22) 결함의 탐지
(Fig. 22) Detection of failure

관리자가 탐지하여야 할 사건/상태의 명세를 받으면 (그림 23)과 같이 자신의 백업에게 알리고 또 이중 일부 부명세가 하위의 관리자에게 할당되어야 하면 이를 하위 관리자에게 전달한다. 주 관리자와 백업 관리자는 각각의 할당된 명세에 대해 다음과 같은 동일한 정보를 갖는다. 사건/상태 명세와 이의 식별자, 어느 하위 관리자나 호스트로부터 탐지 결과가 오는가와 어느 상위 관리자로 탐지 결과를 보고하여야 하는 가에 대한 정보이다. 주 관리자와 백업 관리자에 저장되는 정보를 비교하면 〈표 1〉과 같다.



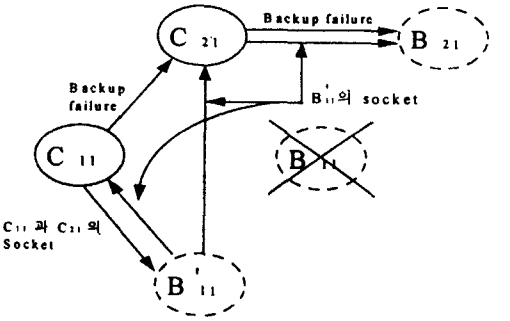
(그림 23) 사건/상태 명세의 할당
(Fig. 23) Allocation of an event/state specification

〈표 1〉 주 관리자와 백업 관리자에 저장된 정보의 비교
〈Table 1〉 Comparison of information stored in a primary manager and backup manager

정보의 종류		주 관리자	백업 관리자
탐지를 위한 정보	할당된 사건/상태의 명세와 이의 식별자	○	○
	할당된 사건/상태의 탐지를 위해 통신해야 하는 상·하위 관리자	○	○
	탐지의 부분 결과	○	×
계층 구조 관련 정보	상위 주 관리자의 주소	○	○
	상위 백업 관리자의 주소	×	×
	모든 하위 주 관리자의 주소	○	○
	모든 하위 백업 관리자의 주소	○	○

일단 탐지가 시작되면 탐지에 대한 부분 결과는 주 관리자 사이에서만 교환이 된다. 만약 주어진 사건/상태의 탐지가 완료되면 이를 탐지한 주 관리자가 이 사건/상태의 탐지를 종결할 것을 자신의 백업과 자신의 하위 관리자에게 알리고 이 종결 통지는 순환적으로(recursively)전달 된다.

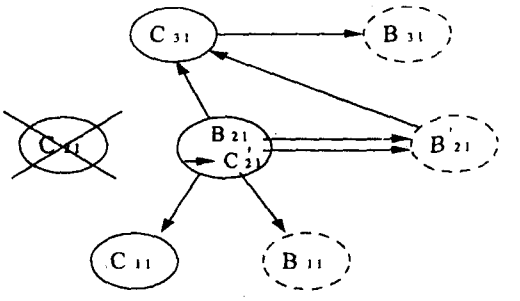
(그림 24)는 백업 관리자에 결함이 발생한 경우를 설명한다. B_{11}의 고장을 탐지한 C_{11}은 이를 상위 관리자 C_{21}에 알리고 C_{21}은 이를 B_{21}에 알린다. C_{21}과 B_{21}은 B_{11}의 소켓 정보를 무효화시킨다. C_{11}은 새 백업 관리자 B_{11}'을 생성하고 C_{11}과 C_{21}의 소켓을 알려준다. B_{11}'은 자신의 소켓을 C_{11}과 C_{21}에 알리고 C_{21}은 이를 다시 B_{21}에 알린다. C_{11}은 B_{11}'의 소켓을 통하여 C_{11}이 관리하고 있는 모든 주관리자, 백업 관리



(그림 24) 백업 관리자의 고장
(Fig. 24) Failure of a backup manager

자나 호스트들의 소켓과 현재 C_{11}이 탐지하고 있는 사건/상태의 리스트를 B_{11}'에 알린다.

(그림 25)는 주 관리자 C_{21}이 고장난 경우이다. C_{21}의 결함을 탐지한 B_{21}은 이를 상위 관리자 C_{31}과 C_{21}의 모든 하위 관리자와 이의 백업인 C_{11}, C_{12}, ...와 B_{11}, B_{12}, ...에 알린다. 이때 C_{21}이 탐지에 관여했던 모든 사건/상태의 식별자도 알려 잠정적으로 이의 탐지를 중단할 것을 알린다. B_{21}은 사건/상태 명세의 할당시 이미 C_{21}과 같은 명세와 이의 식별자를 전달 받았으므로 C_{21}이 어떠한 사건/상태를 탐지하다가 고장이 났는지 자동적으로 알게된다. 이 탐지의 잠정 중단에 관한 통지는 순환적으로 전달된다. 다음 B_{21}은 새 주 관리자 C_{21}'이 되고 새 백업 관리자 B_{21}'을 생성한다. C_{21}'은 B_{21}'에게 상위 관리자, 모든 하위 관리자와 백업의 소켓과 C_{21}'이 현재 탐지하여야 할 사건/상태의 리스트를 전달한다. B_{21}'은 자신의 소켓을 C_{21}'과 C_{31}에 알리고 C_{31}은 이를 B_{31}에 알린다. B_{21}'



(그림 25) 주 관리자 고장
(Fig. 25) Failure of a primary manager

```
backup_failure() {
    notify backup failure to parent primary manager;
    elect a new backup manager;
    send to new backup manager the socket numbers of myself, parent primary manager
        and all the primary/backup manager or hosts managed by me;
    send my subtask list to new backup manager;
}
```

(그림 26) 고장난 백업 관리자를 복구하는 알고리즘
(Fig. 26) An algorithm for recovering a failed backup manager

```
primary_failure() {
    notify primary failure to parent primary manager, all my child primary managers, and
        all my child backup managers;
    pause the detection of all the events/states in whose detection the failed primary
        manager was participated;
    become a new primary manager;
    elect a new backup manager;
    send to new backup manager the socket numbers of myself, parent primary manager
        and all the primary/backup managers or hosts managed by me;
    send my subtask list to new backup manager;
    restart the paused detection;
}
```

(그림 27) 고장난 주관리자를 복구하는 알고리즘
(Fig. 27) An algorithm for recovering a failed primary manager

의 생성이 완료되면 C_{21}'은 잠정적으로 중단되었던 사건/상태의 탐지를 재개할 것을 상위 주 관리자와 하위 주 관리자들에게 알린다. 이 탐지 재개에 대한 통지는 순환적으로 전달된다.

이들 과정 중 백업 관리자의 고장을 발견한 주 관리자가 수행할 복구 알고리즘과 주관리자의 고장을 발견한 백업 관리자가 수행할 복구 알고리즘은 각각 (그림 26) 및 (그림 27)과 같다.

6. 결 론

통신망과 분산 시스템의 효율적인 사용을 위해서는 사건/상태의 적절한 관리가 필요하다. 어떠한 사건/상태들을 성능 저하, 고장, 보안 위반 등의 징후로서 나타날 수 있다. 이들 사건/상태는 가능한 한 빨리 탐지되고 처리되어야 한다.

이 논문은 사건/상태의 명세와 탐지 알고리즘에 대해 설명하였다. 사건/상태는 고전적 시간 논리에 의거한 명세 언어로 표현된다. 이 명세 언어로 표현된 사건/상태에 대해서는 먼저 중앙 집중식 탐지 알고리즘에 대해 설명하였다. 중앙 집중식 탐지 알고리즘의 분산화는 분해와 할당의 두 과정으로 구성된다. 분해 과정에서는 사건/상태의 명세를 부태스크로 나누고 위치 정보 트리를 생성한다. 할당 과정에서는 위치 정보 트리의 정보를 사용하여 부태스크들을 적절한 컴퓨터에 할당한다. 또 제안된 분산 탐지 알고리즘에 결함 허용성을 추가하는 방안에 대해서도 기술하였다. 이 방안은 주관리자와 부관리자를 사용하는 것과 사건/상태에 유일한 식별자를 부여하는 방안에 의거한다.

사건/상태의 탐지는 분산 디버깅과 망관리의 두 분야에서 많이 연구되었다. 분산 디버깅에서는 주로 확장된 정규식에 의해 사건/상태를 명세하고 이에 대해 즉시 탐지 알고리즘을 제공한다. 그러나 즉시 탐지 알고리즘이 가능할 수 있도록 탐지될 수 있는 사건/상태의 종류가 매우 한정되어 있다[2, 3, 5, 6, 11]. 또 분산 탐지를 위해 이미 사건/상태의 명세가 분해 및 할당되어 있다고 가정하고 결함 허용성의 문제가 언급되고 있지 않다. 본 논문에서 소개된 명세 언어와 탐지 알고리즘은 망관리 분야에 응용하는 것을 목적으로 하므로 분산 디버깅보다 다양한 종류의 사건/상태를 표현할 수 있는 명세 언어를 제공하고 있으며 따라서 이의 탐지는 즉시 탐지가 아닌 지연 탐지 알고리즘이 제공된다. 그러나 지연 시간을 최소화하기 위해 데이터 구동 메카니즘과 상태 저장 탐지 기술을 사용하고 있다. 또 사건/상태의 분해 및 할당 알고리즘과 결함 허용성을 위한 알고리즘이 포함되어 있다.

망관리 분야에서는 사건/상태의 명세를 위한 대부분 데이터베이스 질의어를 명세 언어로 사용하였다[8, 9, 10]. 그리고 이의 분산 탐지를 위해서 단지 분산 데이터베이스 시스템의 질의 처리 기능을 사용한다고 가정하였기 때문에 탐지 기술에 대해서는 구체적인 기술이 없다. 본 논문에서 제시한 시간 논리에 의거한 명세 언어는 데이터베이스 질의어보다 훨씬 명세가 간편하다. 또 사건/상태 명세의 할당 및 분해와 탐지는 명세 언어에 따라 달라지므로 본 논문의 명세 언어에 적절한 할당 및 분해와 탐지 알고리즘이 소개되었고 또 결함허용성을 위한 알고리즘도 포함되었다.

본 논문에서 기술한 명세 언어와 탐지 기술은 1장 서론에서 소개한 분산 환경의 관리를 통합 메카니즘 내에서 사용되어 분산 환경이나 통신망에서 발생할 수 있는 성능저하, 고장, 보안 위반 등의 문제점들을 신속히 발견하고 해결하는 목적에 유용히 사용될 수 있다.

참 고 문 헌

[1] Y. C. Shim, "Integrated Mechanism for the Knowledge-Based Management of Communication Networks," KT International Symposium, 1993.

[2] P. Bates and J. C. Wileden, "High-Level Debugging of Distributed Systems: The Behavioral Abstraction Approach," The Journal of Systems and Software, vol. 3, pp. 255-264, 1983.

[3] B.P. Miller and J.D. Choi, "Breakpoints and Halting in Distributed Programs," Int. Conf. on Distributed Computing Systems, pp. 316-323 1988.

[4] W. Hseush and G.E. Kaiser, "Data Path Debugging: Data-Oriented Debugging for a Concurrent Programming Language," ACM SIGPLAN Notices, vol. 24, no. 1, January 1989.

[5] V.K. Garg and B. Waldecker, "Detection of Unstable Predicates in Distribute Programs," Univ-

ersity of Texas at Austin, TR-92-07-82, March 1992.

[6] R. Schwarz and F. Mattern, "Detecting Causal Relationships in Distributed Computations in Search of the Holy Grail," Dept. of Computer Science, University of Kaiserslautern, SFB i24-15/92, Dec. 1992.

[7] A. Di Maio, S. Ceri, and S.C. Reghizzi, "Execution Monitoring and Debugging Tool for ADA Using Relational Algebra," Proceedings of the Ada International Conference, ACM Ada Letters, vol. 5, no. 2, pp. 109, Sep. 1985.

[8] L.M.L. Delcambre and J.N. Etheredge, "The Relational Production Lnaguage = A Production Language for Relational Database," 2nd International Conference on Expert Database Systems, pp. 333-351, 1989.

[9] R. Snodgrass, "A Relational Approach to Monitoring Complex Systems," ACM Transactions on Computer Systems, vol. 6, no. 2, pp. 157-196, May 1988.

[10] O. Wolfson, S. Sengupta, and Y. Yemini, "Managing Communication Networks by Monitoring Database," IEEE Transactions on Software Engineering, vol. 17, no. 9, September 1991.

[11] M. Spezialetti and J.P. Kearns, "A General Approach to Recognizing Event Occurrences in Distributed Computations," Int. Conf. on Distributed Computing Systems, pp. 300-307, 1988.

[12] Y. Chen, Event Management in Computer Network, Ph.D. Thesis, University of California, Berkeley, November 1987.

[13] A. Gupta et al, "Parallel Algorithms and Architectures for Rule-Based Systems," IEEE Int. Conf. on Parallel Processing, pp. 28-37, 1986.

[14] T.A. Coope and N. Wogrin, Rule-Based Programming with OPS5, Morgan Kaufmann, 1988.

[15] F. Cristian, "A Probabilistic Approach to Distribute Clock Synchronization." IEEE Int. Conf. on Distributed Computing Systems, pp. 288-296, 1989.

[16] A.H. Bond & L. Gasser, Distributed Artificial Intelligence, Morgan Kaufmann Publishers Inc., 1988.

심 영 철

1979년 서울대학교 전자공학과 (학사)

1982년 한국과학기술원 전기 및 전자공학과(석사)

1991년 미국 캘리포니아주립대 (버클리) 전산과(박사)

1981년~1984년 삼성전자 대리

1992년~1993년 미국 캘리포니아주립대(버클리) 연구원

1993년~현재 홍익대학교 컴퓨터공학과 조교수

관심분야:분산컴퓨팅, 망관리, 컴퓨터와 망의 보안, 소프트웨어공학