韓國 컴퓨터情報學會 論文誌
第10卷 第6號, 2005. 12.

2005-10-6-1-5

표준 미들웨어 기반 분산 객체 리플리카를 위한 효과적 푸쉬 결함 관리 알고리즘

김 분 희*

An Effective Push Fault Management Algorithm for Distributed Object Replica based on Standard Middleware

Boon-Hee Kim*

요 약

분산 환경 기반으로 많은 작업이 이뤄지는 상황에서 해당 자원에 대한 신뢰성 제공은 매우 중요하다. 자원 제공자는 서비스의 신뢰성을 제공하기위해 일반적으로 채택되는 유형은 서버 리플리케이션 구조 이다. 해당 서버에는 중복된 객체인 리플리카가 유지하는 구조로 이루어진다. 이러한 리플리케이션 구조에서는 안정된 리플리카의 서비스가 중요하므로, 각 리플리카의 결함 발생을 진단하기 위한 구조가 요구된다. 본 논문에서는 분산 객체 시스템에서 제공하는 리플리케이션 관리 시스템의 풀 모니터링 스타일에서 나타날 수 있는 단점을 극복한 CORBA의 푸쉬 모니터링 스타일 기반 효과적 푸쉬 결함 관리 알고리즘을 제안한다. 제안된 푸쉬 결함 모니터링 스타일의 실험결과 비교 대상 시스템에 비해 장애 검출과 관련한 작업량과 평균 타임아웃 비율 측면에서 가치 있는 결과를 확인 할 수 있었다.

Abstract

In processing many operations based on distributed environment, it is very important to support the reliability of resources. Providers of resources generally adapt the structure of server-replication to support the reliability of services. In server side, it maintains replicas, duplicated server objects. In the structure of this replication, service of stable replica is very important. Therefore the structure to diagnosis the fault of such replica is required. In this paper, we suggested an effective PUSH fault management algorithm based on PUSH monitoring style of CORBAto overcome weak points of the PULL monitoring style of the replication management system in the distributed object system. Outcomes of the suggested PUSH fault monitoring style were better than other system. We confirmed valuable result in the workloads and timeout-rates.

▸ Keyword : 리플리카(Replica), 결함 관리(Fault Management), 코바(CORBA)

• 제1저자 : 김분희
• 접수일 : 2005.10.12, 심사완료일 : 2005.12.15
* 동명정보대학교 멀티미디어공학과 전임강사

Ⅰ. 서 론

인터넷 관련 응용분야에서의 분산된 환경은 신뢰성, 안정성, 응답시간 향상성을 제공하기위해 서버 리플리케이션 (server replication) 구조[1]를 가지며, 해당 서버에 중복된 객체인 리플리카(replica: duplicated server object)를 유지하는 구조로 이루어진다. 이러한 리플리케이션 구조에서는 안정된 리플리카의 서비스가 중요하므로, 각 리플리카의 결함 발생을 진단하기 위한 구조가 요구된다. 분산 환경에 존재하는 다양성을 통합함으로써 이형질의 구성 요소들 간에 서로 소통하는데 중요한 역할을 하는 미들웨어 가운데 국제 표준화 활동의 결과인 CORBA(Common Object Request Broker Architecture, 코바)는 해당 리플리카에 대한 다양한 결함 관리(fault management) 방법을 제공해 주고 있다. CORBA의 결함 관리 방법은 다양한 구성 요소들의 복합적인 조합으로써 사용자의 요구에 따라 결함을 찾는 방법이 결정된다. 리플리케이션 관리자를 구성하는 요소 가운데, 특성 관리자(property manager)에 의해 관리되는 주요 요소로써 존재하는 결함 모니터링 스타일(fault monitoring style)은 리플리카의 결함을 감시하는 스타일을 결정한다. CORBA에서 제공하는 대표적인 결함 모니터링 스타일에 풀 모니터링 스타일(pull monitoring style)이 제시되어 있다. 이는 결함이 발생되지 않거나 서비스를 마칠 때 까지도 문제가 발생되지 않은 수많은 리플리카를 감시하기 위해 주기적으로 대상 리플리카를 감시하는 특징을 지닌다. 푸쉬 모니터링 스타일(push monitoring style)은 모니터 대상 객체가 주기적으로 결함 모니터에게 알리는 형태이다. 만약 모니터링 가능 객체가 정해진 시간 간격 내에 알림 메시지를 보내지 않는다면, 장애 모니터는 그것이 고장이 아닐까 의심하게 된다. 이 방법의 주요한 이점은 결함 모니터에 의해 모니터 대상 객체의 결함을 빨리 찾을 수 있다는 점이다[2][3].

결함 진단과 그에 따른 대응체제로써 고장 모니터링 메커니즘은 메시지 수신시간의 타임아웃을 기준으로 한다. 그러나 실제 비동기 분산 시스템에서는 리플리카 자체의 결함과 네트워크 과부하를 구별하기 어렵다. 평균 전송시간보다 낮은 타임아웃 값은 잘못된 장애 검출을 야기하고, 평균 전송시간보다 더 큰 타임아웃 값은 충돌 발생시 장애 검출의 지연 가능성이 높다[8]. 이에 메시지 평균 전송시간과 타임아웃의 상관관계에 따른 신뢰성 부여 모델의 장애 검출 시기 지연 문제를 해결한 절충형 모델이 요구된다.

본 논문에서는 분산 객체 시스템에서 제공하는 리플리케이션 관리 시스템의 풀 모니터링 스타일에서 나타날 수 있는 단점을 극복한 CORBA의 푸쉬 모니터링 스타일 기반 효과적 푸쉬 결함 관리 알고리즘을 제안한다. 제안된 푸쉬 결함 모니터링 스타일의 실험결과 비교 대상 시스템에 비해 장애 검출과 관련한 작업량과 평균 타임아웃 비율 측면에서 가치 있는 결과를 확인 할 수 있었다.

본 논문의 구성은 다음과 같다. 2장에서 관련연구, 3장에서는 본 논문의 핵심 부분인 결함 관리 알고리즘 및 평가를, 마지막으로 4장에서 결론을 맺는다.

Ⅱ. 관련 연구

2.1 CORBA의 결함 검출 기법

본 절에서는 CORBA에서 제공하는 리플리카의 결함 처리 방법을 나타낸다. (그림 1)은 CORBA에서 제공하는 리플리케이션 관리자의 구성요소를 계층화 한 것이다. 본 논문에서 다루고자 하는 부분은 리플리케이션 관리자를 구성하는 요소 가운데, 특성 관리자(property manager)에 의해 관리되는 결함 모니터링 스타일이다. 리플리카의 결함을 감시하는 스타일을 결정해 주는 결함 모니터링 스타일에는 (그림 1)에서와 같이 PULL, PUSH, NOT_ MONITORED로 구분된다. PULL 스타일은 고장 모니터 호출시 정기적으로 생존 여부를 모니터 대상 객체에게 물어보는 방법으로써 “is_alive” 메서드가 그 역할을 한다. 이 스타일은 모니터 대상 객체가 해당 시간 간격 내에서 응답이 없으면, 고장 모니터는 이를 고장으로 판단한다. PUSH 스타일은 모니터 대상 객체가 주기적으로 결함 모니터에게 생존 여부 메시지를 “i_am_alive” 메서드를 통해 호출한다. 이 스타일은 모니터 대상 객체가 정해진 시간 간격 내에 “i_am_alive” 메서드를 호출하지 않았다면, 고장 모니터는 그것이 고장이 아닐까 의심한다. 주요한 이점은 결함 모니터에 의해 모니터 대상 객체의 결함을 빨리 찾을 수 있다는 점이다[5][6].

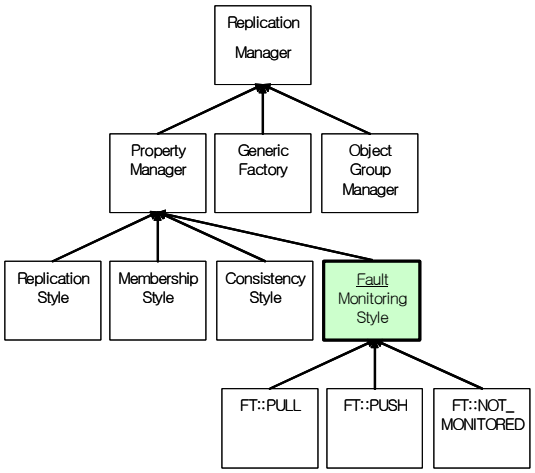


그림 1. 리플리케이션 관리자 요소
Fig 1. Elements of replication manager

2.2 CORBA 결함 모니터링 스타일

(그림 2)는 CORBA의 결함 감지 기법을 보여주는데, 상단의 RM(replication manager)은 PM(property manager), OGM(object group manager), GF(generic factory) 인터페이스를 구현하였다. 즉, RM은 PM, OGM, GF를 모듈로 하는 시스템이다. RM는 결함허용 시스템의 하부구조에서 중요한 역할을 하며 결함 허용 관리 도메인 내의 모든 호스트 상에 존재할 필요는 없다. 그러나 하나의 도메인 내에는 하나의 리플리케이션 관리자가 있어야 한다. RM을 구성하는 모듈 가운데, PM은 객체 그룹의 특징적인 면을 관리하는 역할을 한다. 이러한 관리 영역으로 객체의 리플리케이션 스타일(Replication style), 멤버쉽 스타일(Membership style), 지속성 스타일(Consistency style)을 판단하여 그에 부합하는 객체 특징을 관리한다. 리플리케이션 스타일로 "STATELESS, COLD_PASSIVE, WARM_PASSIVE, ACTIVE, ACTIVE_WITH_VOTING"가 있다. 이를 크게 분류하면 액티브 방식과 패시브 방식으로 나뉠 수 있는데, 액티브 방식은 모든 리플리카가 클라이언트의 요구 사항을 처리해야 하고, 패시브 방식은 리플리카 중 프라이머리만 클라이언트의 요구 사항을 처리하고 나머지 리플리카는 백업용으로 이용되는 방식이다. 멤버쉽 스타일에는 MEMB_APP_CTRL와 MEMB_INF_CTRL이 있다. MEMB_APP_CTRL은 객체 자체를 생성하는 타입으로 add_member()와 create_member() 메서드를 제공한다. 이는 결함 모니터 스타일을 결정하는 역할을 하며, 결함 통지를 받기 위해 결함 통지자와 함께 저장된다. MEMB_INF_CTRL은 객체 그룹에 멤버가 되기 위해 개별적인 저장소 인보크(Repository Invokation) 작업을 한다. 다른 부분은 MEMB_APP_CTRL와 동일하다. RM을 구성하는 모듈 가운데, GF는 객체 그룹을 생성하는 역할을 하며, 객체에 부여된 특성인 "Initial Number Replicas"와 "Minimum Number Replicas"에 따라서 객체를 관리한다. 여기서 "Initial Number Replicas"는 초기 객체 그룹을 생성할 때의 멤버를 나타내고, "Minimum Number Replicas"은 결함이 일어나서 멤버 중 하나를 잃었을 때를 대비하는 개념이다. RM을 구성하는 모듈 가운데, OGM은 그룹의 멤버를 관리하는 역할을 한다. 즉 그룹을 구성하는 멤버의 추가 및 삭제의 역할을 수행한다.

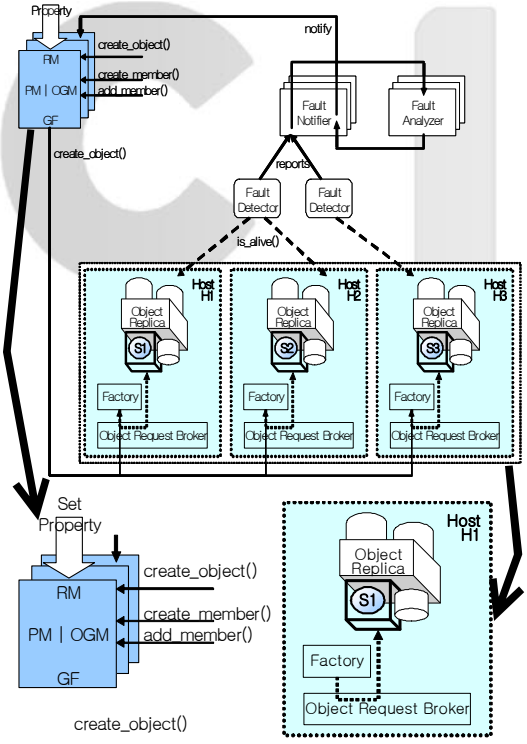


그림 2. CORBA 리플리케이션 관리자의 동작 방식
Fig 2. Operating method of CORBA replication manager

리플리케이션 관리자는 위의 동작 방식에 따라 각 객체의 결함을 검출하게 된다. 여기서 결함 검출기(fault detector)는 PullMonitorable 타입(one of the monitoring style for CORBA)의 어플리케이션 객체에 주기적으로 "is_alive" 명령을 수행하게 되고, 이에 대한 해답의 유무에 따라 결함을 감지하게 된다. 결함이 감지된 경우 결함 검출기는 결함 통보기(fault notifier)에 보고하게 되고, 결함 통보기는 최종적으로 리플리케이션 관리자에게 특정 리플리카의 결함을 기정사실화 하는 통지를 하게 되는 구조이다. 이러한 풀 결함 모니터링 스타일(Pull Fault Monitoring Style)에서 결함 검출기는 결함이 발생되지 않은 또는 서비스를 마칠 때 까지도 문제가 발생되지 않은 수많은 리플리카를 감시하기 위해 주기적으로 감시 대상 리플리카에 "is_alive" 명령을 수행해야 한다. 이러한 사항은 결함 검출기를 관리하는 시스템 측면에서 봤을 때, 시스템 자체적 결함에 대한 대처를 할 수 없고 미미하지만 시스템 성능저하 요인이 될 수 있다. 그리고 이러한 중앙 집중적인 관리보다는 리플리카가 존재하는 각 시스템에서 자신의 시스템의 상태를 자체적으로 관리하는 것이 효율성 측면에서 이점이 있다. 이때 나타날 수 있는 자체적 관리의 측면에서 각 시스템을 이루고 있는 미들웨어에 추가되어야 할 모듈이 차지하는 성능 및 용량 등의 요소에서 나타날 수 있는 부가적 요인은 거의 없음을 본 논문의 평가 결과 증명된다.

III. 제안된 알고리즘

본 논문에서 제안하는 AEffectPush(An Effective Push style) 모델은 분산된 비동기적 결함 관리모델로써 통신 장애가 없는 충돌 방지(crash-stop)형이다. AEffectPush는 이전 모델의 기본 골격을 따르며, 기존 Irineu[11]와 응답율 측면에서 향상된 성능 결과를 보일 수 있도록 알고리즘을 수정하여 효율성을 높였다. 메시지 평균 전송시간과 타임아웃의 상관관계에 따른 신뢰성 부여 모델[6]의 장애 검출 시기 지연 문제를 해결한 절충형 방식[5]에서 검출 시간이 향상된 성능 모델을 제안하고 따른다. AEffectPush 결함 모니터링 알고리즘은 타임아웃 이벤트, 응답호출 이벤트를 기반으로 동작되는데, 타임아웃 이벤트는 특정한 타임아웃 주기가 끝날 때 모니터대상 객체와 시스템 과부하에 의해서 발생되고, 응답호출 이벤트는 푸쉬 모니터가 몇몇 모니터 대상 객체에 의해 호출된 "i_am_alive" 메서드를 활성화 시킬 때에 발생한다.

3.1 모니터링 구조

제안하는 AEffectPush의 모니터링 구조를 보기 이전에 기본적인 사항을 우선적으로 설명하면 다음과 같다. 본 알고리즘을 구성하는 각종 값은 msec(milliseconds) 단위이며, 초기 설정된 타임아웃 값과 모니터링 간격은 시스템 과부하와 실제 결함의 상호 관련성을 고려하여, 실제 실행하면서 최적의 값을 찾기 위해 변경되게 된다. 초기 모니터링 간격은 많은 논문에서 최적의 값으로 판명된 3 msec으로 설정되고, 타임아웃 값은 2 msec으로 초기 설정한다. 그리고, TRS (Transparent Replica Strategy)[9] 정책을 기반으로 초기화 된다. TRS는 기존의 SAR(Semi-Active Replication) 방법의 문제점을 해결한 방법이다.

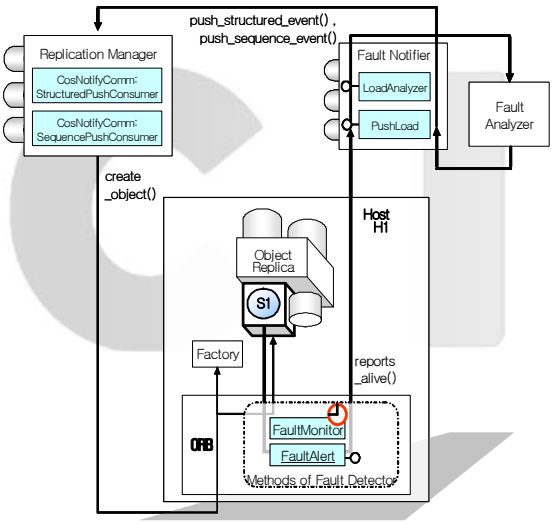


그림 3. 제안된 시스템의 동작 방식
Fig 3. Operating method of suggested system

SAR은 기존의 ART(Active Replication Technique)가 자원의 효율적인 이용 측면에서 비용 부담이 크고, 서버의 상태가 결정적일 때에만 이용될 수 있는 방법이며, PRT(Passive Replication Technique)가 응답시간 지연의 문제와 클라이언트 측면에서의 결함 마스킹의 문제가 있어, 이를 해결한 SAR이 서버의 상태에 따라 ART, PRT을 혼용 하였으나 클라이언트 측면에서의 응답시간 지연문제와

결함 마스킹의 문제가 존재한다. (그림 3)은 전체 리플리케이션 관리자의 동작 방식 중 본 논문에서 설계한 절충형 푸쉬 모델 부분만을 확대한 것이다. 리플리케이션 관리자는 결함 분석기와 결함 통보기의 상호 협조 하에 결정된 결함 정보를 받게 되는데, 이러한 정보를 받고 처리하는 기법은 CORBA 표준 규격에 정의된 바를 수정하지 않는 것이 많은 부분에서 이점을 지닌다. 그래서 본 논문에서는 기본 동작 기법을 수정 하지 않는 방향으로 푸쉬 모델을 설계하였다.

결함 허용 하부구조에서 어플리케이션 또는 하부구조에서 제어되는 그룹 스타일(membership style)로써 객체 그룹의 수를 초기화 하게 되는데, 본 구조에서는 이용자의 선택에 따라 결함 검출기를 구성하는 모듈들을 그대로 이용할 수도 있고, 결함 검출기의 기능을 일시 정지 시킬 수도 있겠다. 전자의 경우 결함 검출기의 핵심 기능인 “is_alive” 명령을 일시정지 시킨 후 서버 리플리카의 ORB(Object Request Broker)가 보고하는 내용을 받고, 이를 결함 식별기(fault identifier)에 보고해야 한다. 이는 해당 서버 측 ORB가 자체적으로 푸쉬하는 메시지를 결함 통보기에게 단지 넘겨주는 역할만 할 뿐임으로 그 유용성에서 문제가 된다. 이러한 기법보다는 풀 스타일을 선택 시 결함 검출기 자체를 일시 정지시키는 것이 알고리즘의 복잡도 측면에서 효율적이다. 결함 통보기는 결함 검출기에 의해 발견된 결함을 리플리케이션 관리자에게 보고하는 역할을 하는데, 결함 통보기와 클라이언트에 의해 선택 되어진 통보의 형태에 따라서 결함 전달이 달라진다.

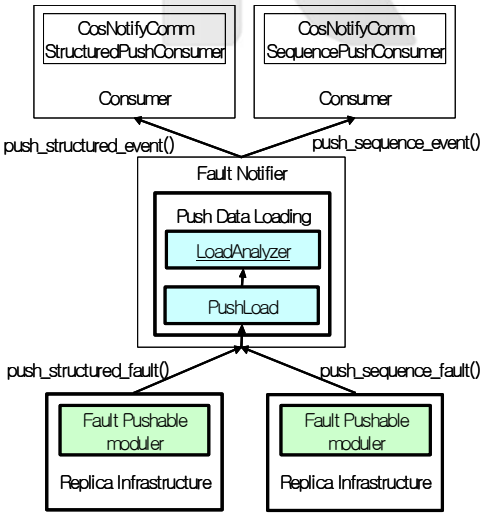


그림 4. 결함 통보기를 통한 결함 전파
Fig 4. Fault propagation through fault notifier

(그림 4)는 제안된 시스템의 결함 통보기를 통해 결함을 보고하기 위한 명령의 파생 과정을 보여준다. 리플리케이션 관리기 구현에 있어서 선택된 타입이 제안한 기법일 경우에 나타나는 결함 전파의 진행을 보여주고 있는데, 선택된 리플리카는 구현 시 PushMonitorable 인터페이스로부터 상속받게 된다. PushMonitorable 인터페이스를 구성하는 “i_am_alive” 오퍼레이션이 백그라운드 스레드로 작동하여 사용자가 정한 시간 간격 (Fault Monitoring Interval And -Timeout‘s TimeBase::TimeT)에 따라 주기적으로 메시지를 통보하고, 이 메시지는 초기 클라이언트가 선정한 통보 타입에 따라 해당 메서드에 파라메터로써 전해지게 된다. 이러한 형태는 사용자 내의 각종 하부구조를 이루는 모듈들을 개별적으로 다른 단위로 인식될 때 가능한 방법이다. 제안된 PushMonitorable 인터페이스는 다음과 같다.

```
module FT{
  interface PushMonitorable{
    typedef unsigned long long ConsumerId;
    ConsumerId i_am_alive();
    void push_structured_Fault(
      in CosNotification::StructuredEvent event);
    void push_sequence_fault(
      in CosNotification::EventBatch events);
  };
};
```

그림 5. FT 모듈
Fig 5. FT module

(그림 5)는 기존의 FT(Fault Tolerance) 모듈의 푸쉬 모니터링 인터페이스이다. 결함이 발견되지 않았을 경우에 결함 없이 본 객체가 살아있다는 메시지를 보내게 되는데, 이러한 역할을 하는 i_am_alive() 메서드와 결함이 발생시 통보하는 역할의 push_sequence_event() 또는 push_structured_event() 메서드는 결함 발생 시 둘 중 하나를 해당 리플리케이션 메니저에 통보하는 구조이다. 제안된 푸쉬 모니터링 스타일(Push Monitoring Style)은 기존의 FaultNotifier 인터페이스에서 확장하였다. FaultNotifier 인터페이스의 기본 프레임웍에 푸쉬 모니터링 스타일과 풀 모니터링 스타일이 있는데, 제안된 푸쉬모니터링 스타일은 FaultNotifier 인터페이스에서 정의한 푸쉬 모니터링 스타일을 확장한 것이다. 그러나 이 푸쉬 모델의 경우 사용자에 의해 선택되어진 결

함 모니터링 스타일이 결함 통보자(Fault Notifier)의 효율적인 동작을 위해 몇가지 추가되는 오퍼레이션이 존재한다. 제안된 푸쉬 결함 모니터링 스타일에 추가되는 오퍼레이션은 이 아래의 (그림 6) 확장된 FT(Fault Tolerance) 모듈과 같이 인터페이스의 모듈로써 한 부분을 차지하게 된다.

```
module FT{
  interface FaultNotifier{
    typedef unsigned long long ConsumerID;
    void push_structured_event(
      in CosNotification::StructuredEvent event);
    void push_sequence_event(
      in CosNotification::EventBatch events);
    ConsumerId connect_structured_fault_consumer(
      in CosNotifyComm::StructurePushConsumer
        push_consumer);
    ConsumerId connect_sequence_fault_consumer(
      in CosNotifyComm::StructurePushConsumer
        push_consumer);
    void disconnect_consumer(in ConsumerId
      connection) raises(CosEventComm::Disconnected);
    void replace_constraint(in ConsumerId connection,
      in CosNotification::EventTypeSeq event_types,
      in string constr_expr);
    void Load_Analyzer(in ConsumerId fault_analyze);
    void Push_Load(in ConsumerId push_id)
      raises(TimeIntervalFault);
  };
};
```

그림 6. 확장된 FT 모듈
Fig 6. Extended FT module

(그림 6)은 푸쉬 결함 모니터링 스타일(Push Fault Monitoring Style)를 위해 재 설계된 FaultNotifier 인터페이스이다. 추가된 부분은 Load_Analyzer()와 Push_Load() 오퍼레이션 이다. Push_Load() 오퍼레이션은 입력으로 받은 "unsigned long" 타입의 사용자 ID 수로써 해당 리플리카가 살아 있음을 감지하게 되고, 사용자가 정해놓은 시간 간격(time interval) 내에 이런 값이 들어오지 않으면 TimeIntervalFault를 발생시킨다. 발생된 결함은 Load_Analyzer() 오퍼레이션에 의해 분석되며, 이에 따라 push_sequence_event() 또는 push_structured_event() 중에 하나를 리플리케이션 메니저에 통보하게 된다. 이는 호스트 내의 모듈들을 개별적인 객체로 선택했을 때이고, 호스트를 하나의 단위로 선택했을 때는 FaultDetector가 존재할 때와 마찬가지의 메커니즘을 따르게 된다. 본 논문에서 이렇게 단위 모듈의 범위에 따라 구분을 지어 따로 설계한 이유는 CORBA 설계 지침 상 모든 방법론이 이를 따르기 때문이다. CORBA 설계 지침을 전폭적으로 따르는 것이 표준의 의미를 높여줄 것이며, 본 논문에서 설계된 푸쉬 모니터링 스타일 또한 CORBA의 청사진에 부합한 범위에서 진행되었다.

```
algorithm FaultPushable { // example of replica1
  call Fault_Monitor(replica1); boolean b=true;
  do {
    check Fault_Monitor(
          b=timeout(replica1));
  } while(b|TimeT);
  if (b == false) {
   call Fault_Alert(
        Push_Load);
   analyze Load_Analyzer(replica1);
  } else if (TimeT) {
    switch(b) {
      case true:
        call Normal(Push_Load);
      case false:
        call Fault_Alert(Push_Load);
        analyze Load_Analyzer(replica1);
    }
  }
}
```

그림 7. FaultPushable 알고리즘
Fig 7. FaultPushable algorithm

위의 (그림 7)의 FaultPushable 알고리즘은 하나의 리플리카를 예로 AEffectPush 결함 관리 시스템의 동작 과정을 표현하였다. 하나의 리플리카가 해당 서버에 생성되면, 그와 더불어 해당 리플리카의 결함을 감시하는 Fault_Monitor 모듈이 동작된다. 주된 동작은 해당 리플리카의 응답 여부를 상시 감시하는 것으로 리플리카가 살아있다는 응답이 정해진 시간이내에 없다면, 타임아웃과 관련된 timeput 모듈이 동작되면서 해당 결함 관리자에게 해당 리플리카의 결함을 알리는 과정이 이어진다. 여기서 결함 관리자의 중앙 관리의 의미로 TimeT라는 시간을 두는데, 정해진 시간내에 해당 리플리카가 살아있다면, 살아있다는 알림을 통보를 받고, 그렇지 않다면 결함에 대한 통보를 받게 된다. 따라서

해당 예제의 리플리카는 결함 모니터 과정에서 결함이 나타났을때 만을 감지하기 위해 동작되는 부분과 결함이 없더라도 글로벌 주기에 따라 주기적으로 통보하는 부분이 함께 동작 된다. 실제 해당 예제의 리플리카에서 글로벌 TimeT가 아닌 자체 결함이 감지된 후에 글로벌 결함 관리자의 리플리카들에 대한 결함 유무를 받아들이는 창구로써의 역할을 하는 Push_Load 모듈로 통보하는 부분과 글로벌 TimeT에 의해 Push_Load 모듈로 통보되는 부분을 구분하여 작성하였다.

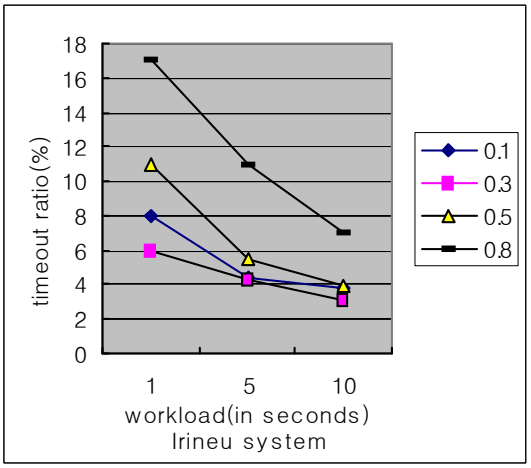
IV. 평 가

AEffectPush는 100Mbps 랜 환경 하에서 진행되었으며, 컴퓨팅 환경은 모두 펜티엄 III 256M RAM MS Windows XP 프로페셔널 버전 2002에서 진행하였다. 그리고 시스템의 구현물은 자바 JDK1.4.1로 이루어졌다. 본 연구의 실험에 적용된 시간간격 값과 작업량 기준은 다음과 같다.

- 시간 간격 값은 0.1, 0.3, 0.5, 0.8 (4개 요소)
- 작업량 평가기준 1, 5, 10 (3개 요소)

(그림 8)은 Irineu 시스템에 대해서 각 작업량과 평균 타임아웃 비율 측면에서의 비교 결과이다. 각 작업량의 평균 타임아웃 비율이 가장 높은 경우는 시간 간격값이 0.8이고, 작업량이 1인 요소로써 약 17%에 이른다. 각 작업량 당 최상의 값을 살펴보면 다음과 같다. 작업량이 1인 경우 0.3의 시간간격 값일 때, 작업량이 5인 경우 0.1의 시간간격 값일 때, 작업량이 10인 경우 0.3의 시간간격 값일 때 최상의 결과를 보이고 있음을 확인 할 수 있다.

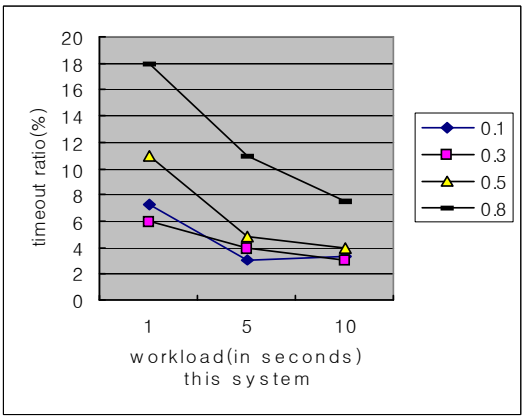
(그림 9)는 AEffectPush 시스템에 대해서 각 작업량과 평균 타임아웃 비율 측면에서의 비교 결과이다. 각 작업량의 평균 타임아웃 비율이 가장 높은 경우는 Irineu와 마찬가지로 시간 간격값이 0.8이고, 작업량이 1인 요소로써 약 18%에 이른다. 각 작업량 당 최상의 값을 살펴보면 다음과 같다. 작업량이 1인 경우 0.3의 시간간격 값일 때, 작업량이 5인 경우 0.1의 시간간격 값일 때, 작업량이 10인 경우 0.3의 시간간격 값일 때 최상의 결과를 보이고 있음을 확인 할 수 있다.



interval \ workload	1	5	10
0.1	8	4.3	3.8
0.3	6	4.4	3.1
0.5	11	5.5	3.9
0.8	17	11	7

그림 8. Irineu 시스템 평가
Fig 8. Evaluation of Irineu system

Irineu와 AEffectPush를 각 작업량 당 최상의 값 차원에서 비교한 결과에서 나타난 원소 단위의 패턴은 유사하였다. 하지만 작업량이 1에서 5로 넘어가는데 따른 변화와 5에서 10으로 넘어가는데 나타나는 결과값의 변화 양상에서 비교할 만한 차이를 보이고 있다. Irineu에서는 작업량이 1에서 5로 넘어갈 때에 타임아웃 비율의 변화가 적었고, AEffectPush에서는 작업량이 5에서 10으로 넘어갈 때 나타나는 타임아웃 비율의 변화가 적었다. 이러한 타임아웃 비율에 따라 각 작업량의 변화에 대한 폭이 달랐는데, 특히 최고 원소값만을 비교해 보면, AEffectPush의 장점구간에서의 차이는 Irineu의 2배에 달했으나 Irineu의 장점구간에서의 차이는 1.19배 차이로 가치 있는 비교 결과를 확인할 수 있었다. (그림 10)에서 이 두 시스템에서의 비교 결과에 대한 설명을 한눈에 확인할 수 있다.



interval \ workload	1	5	10
0.1	7.3	3	3.3
0.3	6	4	3.1
0.5	11	4.8	3.9
0.8	18	11	7.5

그림 9. AEffectPush 시스템 평가
Fig 9. Evaluation of AEffectPush system

V. 결론 및 향후 연구

메시지 수신시간의 타임아웃을 기준으로 하는 결함 진단과 그에 따른 대응체제로써 고장 모니터링 메커니즘은 실제 비동기 분산시스템에서는 리플리카 자체의 결함과 네트워크 과부하를 구별하기 어려운 체제이다. 평균 전송시간보다 낮은 타임아웃 값은 잘못된 장애 검출을 야기하고, 평균 전송시간보다 더 큰 타임아웃 값은 충돌 발생시 장애검출의 지연 가능성이 높다. 이에 메시지 평균 전송시간과 타임아웃의 상관관계에 따른 신뢰성 부여 모델의 장애 검출시기 지연 문제를 해결한 절충형 모델이 요구된다.

본 논문에서는 분산 객체 시스템에서 제공하는 리플리케이션 관리 시스템의 풀 모니터링 스타일에서 나타날 수 있는 단점을 극복하고, 시간 복잡도를 줄인 절충형 장애 검출 기반의 AEffectPush를 제안하였다. 제안된 푸쉬 결함 모니터링 스타일의 실험결과 비교 대상 시스템에 비해 장애 검출과 관련한 작업량과 평균 타임아웃 비율 측면에서 가치 있는 결과를 확인 할 수 있었다. 향후 P2P의 기본 통신기반 기술[12]과 분산 미들웨어의 역할을 확장하여 유비쿼터스 컴퓨팅 환경에서 지역적인 시나리오에 적합한 통신 기술로 확장하고자 한다.

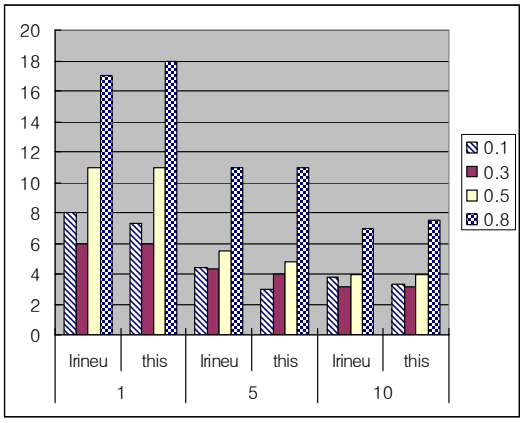


그림 10. 두 시스템 평가
Fig 10. Evaluation of two system

참고문헌

[1] 박종선, 장용철, 오수열, "적응적 중복 객체 알고리즘을 이용한 객체 복제본 관리 연구," 한국컴퓨터정보학회, pp.51-59, 2003.3.

[2] P. Narasimhan, L. E. Moser, P. M. Melliar-Smith, "Replica Consistency of CORBA Objects in Partitionable Distributed Systems," Distributed Systems Engineering, vol. 4, no. 3, pp. 139-150, Sep. 1997.

[3] L. Bobet, "The Push Model in a Java-Based Network Management Application", Mar., 1999.

[4] S. Bagchi, K. Whisnant, Z. Kalbarczyk, and R. K. Iyer, "Chameleon: A Software Infrastructure for Adaptive Fault Tolerance," Proc. of the 17th IEEE Symposium on Reliable Distributed Systems, Oct. 1998.

[5] K. P. Birman, "Reliable Distributed Computing with the Isis Toolkit, IEEE, 1994.
[6] J-Ch. Fabre and T. Perennou, "A Metaobject Architecture for Fault-Tolerant Distributed Systems: The FRIENDS Approach," IEEE Trans. on Computers, vol. 47, no. 1, pp. 78-95, 1998.
[7] Z. Fei, S. Bhattachaarjee, E. Zegura, and M. Ammar. A Novel Server Selection Technique for Improving Response Time of a Replicated Serviced. In Process. Of the IEEE INFOCOM'98, March 1998.
[8] T. Chandra, "Unreliable Failure Detector for Reliable Distributed Systems", Journal of the ACM, 43(2), 225-267, 1996.
[9] B.-H. Kim, Y.-C. Kim, "Transparent Replica Strategy for Fault-Masking in Real-Time Distribution System", Proceedings of the 29th KISS fall conference, Oct, 2002.
[10] O. Othman, "Transparent Resource Management for QoS-Enabled DRE Systems", 2002.
[11] I. Sotoma, E. Madeira, ADAPTATION-Algorithms to Adaptive Fault Monitoring and Their Implementation on CORBA, DOA '01 Proceedings 3rd International Symposium, pp219-228, 9. 2001.
[12] 김분희, 이준연, "단계별 OK 기법 기반 효과적 P2P 검색 알고리즘", 한국컴퓨터정보학회 논문지, 10권 2호, 2005. 5.
[13] CORBA Specification v3.0, 2002.

저 자 소 개

김 분 희

2005년 2월 중앙대학교 컴퓨터공학과 공학박사
2005년~현재 동명정보대학교 멀티미디어공학과 전임강사
<관심분야> 분산 시스템, P2P 검색 기법, 유비쿼터스 컴퓨팅