실시간 분산시스템의 결함 마스킹을 위한 투명성 부가 중복 전략

김분희[0] 김영찬
중앙대학교 컴퓨터공학과
{bhkim[0], yckim}@sslab.cse.cau.ac.kr

Transparent Replica Strategy for Fault-Masking in Real-Time Distribution System

Boon-Hee Kim[0] Young-Chan Kim
Dept. of Computer Science and Engineering, Chung-Ang University

요 약

결함허용은 어떠한 시스템 요소에 결함이나 오류가 발생하더라도 시스템이 정상적으로 동작하게 하는 방안으로써 실시간 분산 시스템에서 그 효용성이 극대화 된다. 본 연구는 실시간 분산 시스템의 결함 허용 기법 중 시간 제약성 측면에서 강한 여분기반 결함허용 기법을 수용한다. 이 기법의 구성 요소인 어플리케이션 서버는 그 상태가 결정적(deterministic)이냐 비결정적(nondeterministic)이냐에 따라 그 처리 기법을 달리하고 있다. 그 중 SAR(Semi-Active Replication)이 자원 활용도 측면에서 그 효율성 증명된바 있다. 본 논문에서는 SAR의 단점인 응답시간 지연문제와 클라이언트 측면에서의 결함 마스킹(fault-masking) 문제를 해결한 구조를 제안한다.

1. 서 론

결함허용 기법에는 크게 진단, 탐지, 여분 기반 결함허용 기법이 있는데 여기서 여분기반 결함 허용 기법이 타 방법에 비해 시간 제약성 측면에서 우수하다. 여분기반 결함허용 기법은 실제 실시간 시스템(boing 737 외 다수), 상용트랜잭션 시스템(AT&T 5ESS교환기 외 다수)에서도 이용되고 있는 기법이다[1].

결함 허용 기법은 크게 하드웨어와 소프트웨어 요소로써 분류되는데, 이들 기법 중 어느 한쪽에만 치우쳐 사용할 경우 시스템에 충분한 결함허용 능력을 부여하기에는 제약이 따르므로 조화롭게 각 시스템 구성요소에 적용하는 것이 바람직하다[2]. 이와 같은 요소를 효과적으로 반영한 여분기반 결함 허용 기법으로 소프트웨어 복제기법을 들 수 있는데, 소프트웨어 서버가 수행되는 노드의 하드웨어 결함을 허용할 수 있도록 동일한 버전의 서버를 여러 노드에 분산 시키는 기법이다.

최근 프로그래밍 경향은 멀티 쓰레드 기법이 주류이므로 소프트웨어 복제 기법 중 수동 여분기법의 반영은 필수 요소이다. 그러나 결함허용 시스템의 빠른 응답시간 부여를 위해서는 능동 여분 기법 또한 배제할 수 없다. 그래서 서버의 상태가 결정적이냐 비결정적이냐에 따라 적절한 대응이 이루어지는 기법인 반 능동 여분 기법이 탄생하였다.

본 논문에서는 이러한 반 능동 여분기법에서 나타나는 응답시간 지연문제와 클라이언트 측면에서의 결함 마스킹 문제를 해결하기 위한 구조를 제안한다.

본 논문의 구성은 다음과 같다. 2장에서 결함허용 기법의 관련 연구를 3장에서는 제안한 시스템의 구조를 설명한다. 마지막으로 4장에서 결론을 맺는다.

2. 관련 연구

2.1 여분기반 결함허용 기법

정적 여분 기법의 기본 구조는 그림 1과 같다. 세 개의 복제객체(replica object)는 동일한 입력을 받아서, 각자 계산 수행 후 그 결과값을 투표자에게 전달하는 방식으로 투표자는 다수의 결과값만을 출력하게 된다. 대표적인 방법으로 TMR, NMR, N버전 프로그래밍등이 있다.

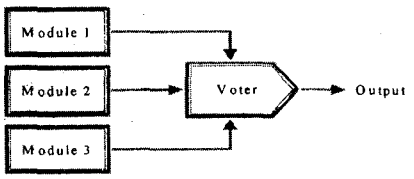

그림 1. 정적 여분 기법의 구조

정적 여분 기법의 기본 구조는 그림 2와 같다. 결함이 있는 복제를 찾아서 정상적인 복제로 교체하는 방법으로 스탠바이 스페어, 부하공유 기법등이 있다.

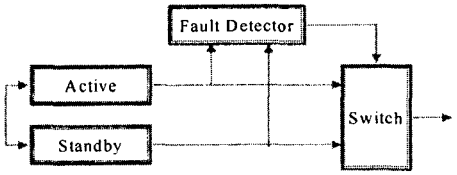

그림 2. 동적 여분 기법의 구조

2.2 상세 분류

여분기반 결함허용 기법은 그림 3과 같이 다양한 분류에 의해 정리될 수 있다.

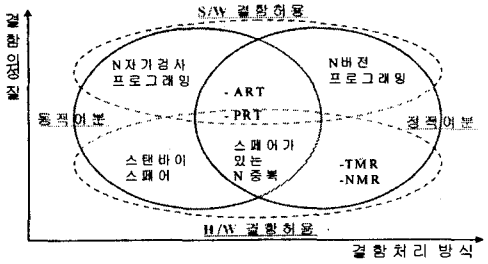

그림 3. 여분기반 결함허용기법 분류

그림 3과 같은 다양한 결함 허용 기법 가운데, 적용 시스템에 알맞은 기법을 선택하기란 어려운 일이다. 이에 Avizienis[2]는 "결함 허용 기법은 크게 하드웨어, 소프트웨어 요소로써 분류되는데, 이들 기법 중 어느 한쪽에만 치우쳐 사용할 경우 시스템에 충분한 결함허용 능력을 부여하기에는 제약이 따르므로 조화롭게 각 시스템 구성요소에 적용하는 것이 바람직하다"라고 언급한 바 있다.

2.3 균형 복제 기법 분류

그림 3의 중심에 해당하는 기법들을 그림 4와 같이 분류할 수 있는데, 소프트웨어 서버가 수행되는 노드의 하드웨어 결함을 허용할 수 있도록, 동일한 버전의 서버를 여러 노드에 분산시키는 방법이다.

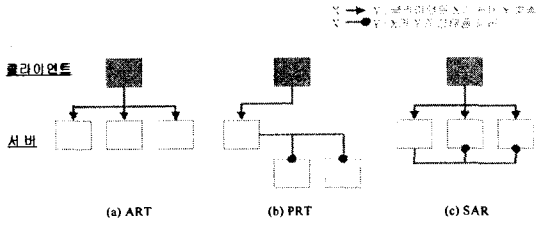

그림 4. 대표적인 균형 복제 기법

ART(Active Replication Technique)은 복제된 모든 서버들이 직접 클라이언트의 요청을 처리하고, 클라이언트는 결과값의 각각을 비교 검토한 후 다음 작업을 수행하게 된다. 이는 서버의 상태가 결정적인 작업에서만 이용할 수 있다. 이러한 ART는 그 메커니즘 때문에 정상적인 경우 실행 오버헤드가 크지만 고장의 경우 복구시간 측면에서 매우 우수하다.

PRT(Passive Replication Technique)는 하나의 서버만 클라이언트의 요청 처리하고 그 결과값을 클라이언트에게 보내게 된다. 비결정적인 서버의 작업에도 동기화 된다. 이러한 PRT는 실행 오버헤드는 위 기법에 비해 작지만 주 서버가 고장 난 경우 복구시간 측면에서 불리하다.

SAR(Semi-Active Replication)은 서버의 상태에 따라 ART와 PRT기법 선택하는 방법이다. 그 외에도 SPR[4]이 있는데, 수동 여분 기법의 멤버쉽 서비스 문제를 해결한 기법이다.

3. TRS 설계

기존 소프트웨어 복제 기법들의 문제점은 다음과 같다. 우선 ART는 자원의 효율적인 이용 측면에서 비용부담이 크고, 서버의 상태가 결정적일 때에만 이용될 수 있는 방법이다. PRT는 응답시간 지연의 문제와 클라이언트 측면에서의 결함 마스킹의 문제가 있다. 이에 대해 SAR은 서버의 상태에 따라 ART, PRT을 혼용 하였으나 실시간 분산 시스템에 적용하기에는 클라이언트 측면에서의 응답시간 지연문제와 결함 마스킹의 문제를 해결하여야 한다. 이에 본 논문에서는 SAR의 문제점을 해결한 그림 5와 같은 TRS(Trans- parent Replica Strategy) 기법을 제안한다.

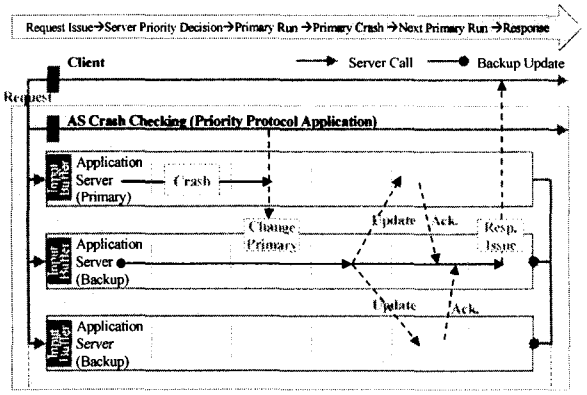

그림 5. TRS(Transparent Replica Strategy) 수행 구조

PRT라 할지라도 클라이언트 요청은 무조건 모든 AS(application server)에 알린다. AS 간의 우선순위 적용으로 결함이 일어났으면서 다른 AS에 갱신(update) 정보를 주지 못해서 클라이언트의 요청의 재발행(request reissue)이 일어났어야 했던 수동여분 기법과는 달리, AS에 결함 일어나면 바로 AS 간 우선순위에 의해 다음 우선순위의 서버가 주AS가 되어 재시작 하게 된다.

본 모델은 SAR(semi active replication) 기법을 기반으로 제안된 구조로써 SAR의 기본적인 가정과 연합하였다. 따라서 SAR 부분에 대한 구조적 기능적 요소는 [3]에 기인한다. 본 논문에서 제안한 그림 6의 TRS 알고리즘은 아래의 표시법(the notation)을 따른다.

- ∂S : 주 어플리케이션 서버
- $\forall S$: 서버 프로세스 집합
- $\forall C$: 클라이언트 프로세스 집합
- $\mathring{A}ASCC$: 서버 프로세스 상의 ASCC(Application

Server Crash Checking) 프로토콜

```
1:  Initialization. TRS.
2:                  ÅASCC → set();
3:                  ∀C → set();
4:                  ∀S → set();

5:  AP.StartAll(ÅASCC);
6:  ÅASCC.PriDecision(∀S);
7:  ready(∀S.req);              //client request issue: wait
8:  inRequest→bufSet(∀S);

9:  try {
10:                 run(∂S);
11:                 ∀C.response = ∂S.resIssue();
12:                 ∂S.updateIssue();
13: } catch(∂S.Crash) {
14:                 ÅASCC.rePriDecision(∀S); //renew AP
15:                 rerun(∂S);
16: }

17: Completion. TRS.
```

그림 6. TRS(Transparent Replica Strategy) 알고리즘

그림 6은 우선 TRS 초기화 관련 작업이 이루어진다. 다음으로 어플리케이션 서버들이 모두 ÅASCC 프로토콜 하에 동작되게 된다. ÅASCC는 ∀S의 우선순위를 부가하여 ∂S의 결함 발생을 감지하고, ∀S의 우선순위를 부가하여 결함 발생 이후 변경될 ∂S를 결정하여 클라이언트 요청을 변경없이 재수행 시킴으로써 동작 상태인 시스템에게 결함 허용성을 부여하는 프로토콜이다. 이렇게 초기화 작업이 끝나면, 모든 AS는 ∀C의 요청이 발행되기를 무한 루프 상태로 기다린다. 요청이 들어오면, ∀S에게 요구 내용을 보내고 ∂S가 실행된다. 이때 ∂S에 결함이 발생되지 않으면 정상적으로 실행된 결과를 나머지 AS에게 갱신하고, ∀C에 응답 메시지를 발행하게 된다. 그렇지 않고 ∂S에 결함이 발생되면 ÅASCC 프로토콜에 따라 다음 주 서버를 결정하고, 다시 위의 동작 상황을 반복하게 된다.

이로써 AS의 결함 발생에도 불구하고 클라이언트와 서버간의 메시지 재발행으로 야기되는 문제점을 해결하였으며, 클라이언트 측면에서 AS의 결함을 인식 할 필요없이 동작될 수 있는 결함 마스킹이 이루어지게 된다.

4. 결론 및 향후연구

본 연구는 실시간 분산 시스템에 적합한 새로운 결함 허용 기법을 제안하였다. 이는 기존 기법의 응답시간 지연 문제와 클라이언트 측면에서의 결함 마스킹 문제를 해결하기 위한 결함허용 기법이다.

향후 본 연구는 실시간 분산 시스템의 신뢰도 향상을 위해 결함 허용 기법을 도입하기 이전에 결함이 일어날 것을 검증하는 결함 방지에 대한 연구가 필요하다. 이를 위해 형식명세를 도입하고, 실제 검증 및 테스트 할 필요가 있겠다.

또한 실제 운영되고 있는 시스템에서 네트워크 관련 부분 및 손실채널(lossy channel) 확장 부분에 대한 결함 허용도 측정 평가를 통해 적용효율성 측면을 판단할 필요가 있겠다.

참고문헌

[1] 임형택, 고 신뢰도의 결함 허용 실시간 시스템 구축을 위한 모델과 실행환경, 숭실대 박사학위 논문, 2001.

[2] Avizienis, A. and Kelly, J.P.J. "Fault Tolerance by Design Diversity: Concepts and Experiments", Computer, vol. 17, no. 8, pp. 67-80, Aug. 1984.

[3] A.Schiper, Software-Based Replication for Fault Tolerance, 1997.

[4] A.Schiper, Semi-Passive Replication, SRDS Conference, 1998.

[5] K.Tanaka, Pseudo-Active Replication of Distributed Objects in Wide-Area Networks, DOA'99 Conference.

[6] S. Ghosh, Guaranteeing Fault Tolerance Through Scheduling in Real-Time Systems. PhD thesis, University of Pittsburgh, 1996.

[7] M.Pandya and M. Malek, Minimum achievable utilization for fault-tolerant processing of periodic tasks, IEEE Trans. Computer Vol. 47, pp.1102-1112, Oct. 1998.

[8] X. Castillo, S. McConnel, and D. Siewiorek, Derivation and calibration of a transient error reliability model, IEEE Trans. Computers, vo.31, pp.658-671, July 1982.

[9] J.C. Fabre and T. Perennou, A Metaobject Architecture for Fault-Tolerant Distributed Systems: The Friends Approach, IEEE Trnasactions on Computers, pp.78-95, Jan. 1998.

[10] J.C.Fabre, V.Nicomette, Implementing Fault Tolerant Applications using Reflective Object-Oriented Programming, Proc. FTCS'95, IEEE Computer Soc., pp. 489-498, 1995.

[11] W. Torres-Pomals, Software Fault Tolerance : Tutorial, NASA/TM-2000-210616, Oct. 2000.