ISSN: 1226-7244 (Print)
ISSN: 2288-243X (Online)
논문번호 15-03-11
76

j.inst.Korean.electr.electron.eng.Vol.19,No.3,357~365,September 2015
http://dx.doi.org/10.7471/ikeee.2015.19.3.357

멀티코어 기반 파티셔닝 운영체제를 이용한 분산 복구 블록 설계 기법 및 응용

Design Technique and Application for Distributed Recovery Block Using the Partitioning Operating System Based on Multi-Core System

박 한 솔*★

Hansol Park*★

Abstract

Recently, embedded systems such as aircraft and automobilie, are developed as modular architecture instead of federated architecture because of SWaP(Size, Weight and Power) issues. In addition, partition operating system that support multiple logical node based on partition concept were recently appeared. Distributed recovery block is fault tolerance design scheme that applicable to mission critical real-time system to support real-time take over via real-time synchronization between participated nodes. Because of real-time synchronization, single-core based computer is not suitable for partition based distributed recovery block design scheme. Multi-core and AMP(Asymmetric Multi-Processing) based partition architecture is required to apply distributed recovery block design scheme. In this paper, we proposed design scheme of distributed recovery block on the multi-core based supervised-AMP architecture partition operating system. This paper implements flight control simulator for avionics to check feasibility of our design scheme.

요 약

최근 항공기, 자동차와 같은 시스템들은 크기, 무게, 전력 등의 문제로 기존 연합형(Federated) 구조에서 모듈형(Modular) 구조로 개발되는 추세이며, 단일 하드웨어에 파티션 개념을 적용하여 다수의 논리적 노드들을 운용할 수 있는 파티션 운영체제도 등장하고 있다. 분산 복구 블록은 실시간 시스템에 적용 가능한 소프트웨어 결함 허용 기법으로 다수의 물리적 노드들을 동기화 시켜 동작시킴으로써 실시간 절체가 가능하도록 하는 설계 기법이다. 분산 복구 블록은 노드들 간의 실시간 동기화를 필요로 하기 때문에 단일 코어 기반의 파티션 구조에는 적합하지 않으며, 적용을 위해서는 멀티코어를 기반으로 하고 또한 AMP(Asymmetric Multi-Processing) 방식을 이용한 파티션 구조에 적용되어야 한다. 본 논문에서는 멀티코어 기반 supervised-AMP 가상화 방식의 파티션 운영체제를 이용한 분산 복구 블록 설계 기법을 제안한다. 또한 제안된 설계 기법의 유용성을 보이기 위하여 항공기용 비행제어시스템 시뮬레이션을 이용한 사례 연구를 보인다.

Key words : Partition Operating System, Distributed Recovery Block, Fault Tolerance System, Multi-Core, AMP Structure

* Dept. of Software Group(C4I), Hanwha Thales.

★ Corresponding author email : hansol.park@gmail.com, phone:+82-31-8020-7303

Manuscript received Aug. 4, 2015; revised Aug. 26, 2015 ; accepted Aug 27. 2015

Ⅰ. 서론

최근 항공기, 자동차 등을 포함한 내장형 탑재 시스템들은 탑재 장비의 크기, 무게, 전력 등의 문제로 기존의 다수의 장비가 탑재되어 연동되는 방식인 연합형(Federated) 구조에서 단일 하드웨어 내부에서 모듈단위로 구분되는 모듈형(Modular) 구조로 개발되고 있다[1]. 또한 항공전자분야에서는 모듈형 구조를 지원하고 소프트웨어의 실시간 요구사항 만족 및 안정성을 보장하기 위한 표준규격으로 ARINC 653과 같은 파티션 개념을 지원하는 표준도 개발되었으며[2], ARINC653 표준을 따르는 파티셔닝 운영체제로는 VxWorks 653, Integrity-178, Lynx178 등의 상용 운영체제와 eCos 기반의 TMO.p[3] 등이 개발 및 연구되고 있다.

소프트웨어를 이용한 결함 허용을 지원하는 분산 복구 블록은 순차적 실행 특성을 갖는 실시간 시스템에 적합한 구조로 구성되어 있으며, 소프트웨어와 하드웨어의 결함을 동시에 처리하고, 하드웨어 노드 단위의 다중화 시스템을 지원할 수 있는 소프트웨어 결함 허용 설계 기법이다[4]. 분산 복구 블록은 다수의 하드웨어 노드들 간의 실시간 동기화를 기반으로 동작하는 구조이며, 개별 노드들은 자체 수락시험(Acceptance Test)과 상대 노드 결과를 제한 시간 내에 검토하여 결함 여부를 판단하게 된다.

단일 처리 유닛(Single Processor/Core)상에 하드웨어 추가 없이 다수의 파티션을 운용하는 구조에 분산 복구 블록을 적용하게 되면 각 파티션이 단일 처리 유닛의 실행 시간을 나누어 순차적으로 실행되기 때문에 파티션간 실시간 동기화의 불가능에 따르는 제약이 발생하게 된다. 이와 같은 제약을 극복하기 위해서는 단일 처리 유닛을 넘어선 멀티코어 또는 멀티프로세서를 이용한 파티션 개념이 적용되어야 하며, 최근 경향을 고려한다면, 단일 프로세서에 다수의 코어를 탑재하여 저전력 성능을 향상시키는 방식인 멀티코어 기반 AMP 방식의 파티션 적용이 필요하다.

멀티코어 기반 AMP 방식에서는 코어별 별도의 운영체제가 마치 독립된 시스템처럼 동작하게 되며, 공유 자원을 제외한 일반 자원들은 특정 코어에 귀속되게 된다. 그리고 코어별로 운영체제가 동작하여 많은 메모리를 사용되는 단점이 있다. 이와 같은 기능을 제공하는 상용 운영체제로는 Green Hills 사의 Integrity-178 TuMP와 Wind River 사의 VxWorks 653 3.0 MCE가 있다.

본 논문에서는 멀티코어 상에서 파티션 개념을 적용하여 운용할 수 있는 운영체제로 최근 개발되어 시범 적용 중인 VxWorks 653 3.0 MCE(Multi-Core Edition)을 사용하여 가상화 기반의 Supervised-AMP 방식으로 파티션을 운용하고, 해당 파티션들을 이용한 단일 하드웨어 내에서의 분산복구블록 설계 기법을 제안하고자 한다. 제안되는 기법은 단일 하드웨어 내부의 파티션 간 절체가 가능한 결함 허용 구조를 제안하며 소프트웨어 수준에서의 분산 복구 블록을 이용한 결함 허용 기능을 제공하게 된다.

본 논문은 다음과 같은 순서로 작성되어 있다. 본론 1절에서는 배경지식 및 관련연구에 대해서 알아보고, 2절에서는 멀티코어 기반 파티션 운영체제에 분산 복구 블록 적용을 위한 방안을 설명한다. 3절에서는 제안된 파티션 기반의 분산 복구 블록의 유용성을 확인하기 위한 비행제어시스템에 적용되는 분산 복구 블록을 설명하고, 4절에서는 3절에서 설계한 비행제어 시스템용 분산 복구 블록을 싱글 코어와 멀티코어에서 시험하여 실시간 성능 비교 결과를 평가한다. 마지막 결론에서는 본 논문의 결론과 향후 계획에 대해서 기술한다.

Ⅱ. 본론

1. 배경지식 및 관련연구

1.1. 배경지식

1.1.1. 파티셔닝 및 지원 운영체제

파티셔닝이란 여러 어플리케이션들을 각각 개별 파티션 개념으로 분리하여 각 파티션에 대하여 시간, 공간적 분할을 제공하는 개념을 의미한다. 공간분할은 다른 파티션들이 서로의 물리적 메모리 자원에 영향을 끼치지 못하도록 고립시키는 것을 말하며, 시간분할은 파티션에 할당된 시간을 다른 파티션에서 간섭할 수 없도록 구분하는 것을 의미한다.

항공전자 분야에서는 파티셔닝을 위한 표준을 제정하였으며, ARINC653이 대표적인 표준이다. 이 ARINC653 표준을 따라 개발된 제품만이 파티셔닝의 신뢰성을 보장 받을 수 있다. WindRiver사의 VxWorks 653, Green Hills사의 Integrity-178등이 대표적인 ARINC 653 지원 운영체제이다. 다음 그림 1은 VxWorks 653의 구조도를 나타낸다.

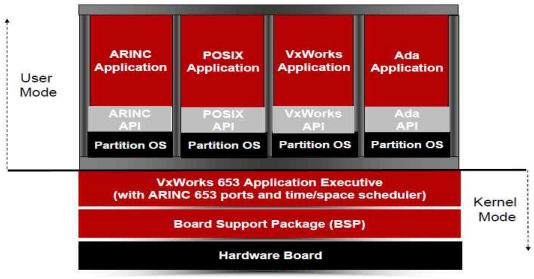

Fig. 1. VxWorks 653 Architecture
그림 1. Vxworks 653 구조도

위 그림 1에서 볼 수 있듯이, VxWorks 653은 하드웨어 상에 ARINC 653 서비스를 제공하며, Partition OS를 이용하여, 시공간적으로 분리된 응용 어플리케이션 운용을 가능하게 하는 운영체제이다.

1.1.2. 멀티코어 기반 파티셔닝 및 지원 운영체제

기존의 파티셔닝 지원 운영체제들은 단일 코어를 기반으로 시간, 공간적으로 구분하여 단일 하드웨어 플랫폼 상에서 파티션으로 구분된 어플리케이션들을 안전하게 동작시킬 수 있었다. 최근에는 멀티코어 기술이 발전함에 따라 멀티코어 플랫폼 상에서 코어별 독립 운영체제 및 어플리케이션을 동작시킬 수 있는 파티션 운영체제들이 등장하였으며 가상화 기술인 하이퍼바이저(Hypervisor)가 적용되어 파티션별로 다양한 게스트 운영체제의 사용이 가능하다.

다음 그림 2는 멀티코어 하드웨어 기반에서 코어별 다중 파티션 운용이 가능한 VxWorks 653 3.0 MCE의 구조도를 나타낸다.

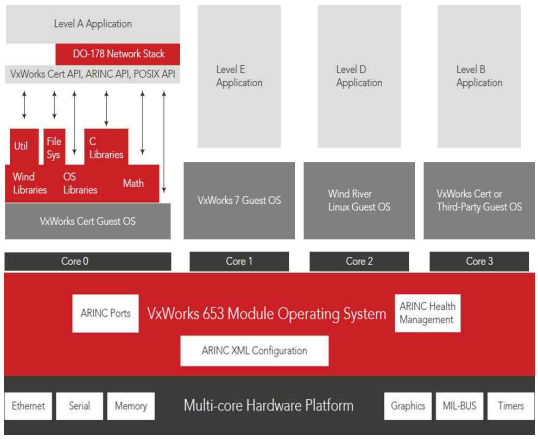

Fig. 2. VxWorks 653 3.0 MCE Architecture
그림 2. Vxworks 653 3.0 MCE 구조도

위의 그림 2와 같이, VxWorks 653 3.0 MCE는 멀티코어 하드웨어 상에서 코어별 파티션 환경을 구성할 수 있으며, 단일 코어상에 다중 파티션 환경을 구성하는 것 또한 가능하다. VxWorks 653 Module Operating System으로 표기된 레이어가 하이퍼바이저 역할을 수행하며, ARINC 653 표준이 제공하는 서비스를 제공한다.

1.2. 관련연구

1.2.1. 분산 복구 블록

분산 복구 블록은 기능 단위의 결함 허용을 가능하게 하는 복구 블록을 분산 또는 병렬 환경에서 실행 가능하도록 확장한 설계 기법으로, 하드웨어와 소프트웨어의 결함을 동일한 방식으로 처리하며, 복구 블록 기법과는 다르게 전향적 복구를 지원하는 설계 기법이다. 다음 그림 1처럼 분산 복구 블록은 2개의 주(Primary) 노드와 백업(Backup) 노드로 구분되며, 운용 중 모드가 변경될 수 있다. 그리고 주 노드와 백업 노드로 구성된 단위를 '복구 스테이션' 이라고 정의한다.

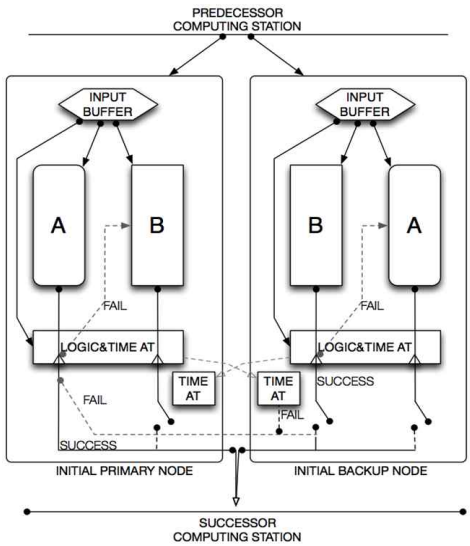

Fig. 3. Conceptual diagram of Distributed Recovery Block[4]
그림 3. 분산 복구 블록 개념도[4]

위 그림 3에서 볼 수 있듯이, 분산 복구 블록의 주 노드와 백업 노드는 처리 결과(Logic)와 시간(Time)에 대한 수락시험(Acceptance Test)을 통해 절체를 수행하는 구조로 동작하게 된다. 또한 절체 시 이전

노드의 데이터를 사용하는 방식이 아닌 절체 대상 노드의 데이터를 사용하는 전향적 복구를 지원하고 모드 전환에 많은 시간이 소요되지 않아 실시간 처리가 요구되는 분야에 적합한 특징을 갖는다.

주 노드와 백업 노드의 2중화만 지원하는 기존의 분산 복구 블록을 확장하기 위한 연구의 하나로 주 노드와 백업 노드를 2개 이상으로 확장하여 이중화 이상의 구성을 가능하게 하고, 세부 모듈단위의 관리도 가능하게 하는 연구[5]도 진행되었으며 이외에도 다양한 형태로의 확장 적용 연구도 수행되었다. 이와 같이 분산 복구 블록은 다양한 가능성과 확장성을 갖는 소프트웨어 결함 허용 설계 기법이다.

1.2.2. 파티셔닝 운영체제 기반 결함 허용 기법

파티셔닝을 지원하는 운영체제를 활용한 결함 허용 연구는 주로 항공 탑재 장비에서 수행되었다. 항공전자분야에서는 파티셔닝에 관한 ARINC 653 국제 표준이 있으며, 이를 따르는 시스템을 IMA(Integrated Modular Architecture)라고 한다. IMA 기반의 결함 허용 연구로 헬스 모니터링(Health Monitoring)을 이용한 결함 허용에 관한 연구[6] 및 관성 항법 장치의 이중화를 파티션 운영체제를 기반으로 적용한 연구[7] 등이 수행되었다.

수행되었던 연구들은 단일 처리기(Single Core / Processor) 기반의 시스템에서 다수의 파티션을 운용하면서 결함 발생 시 다른 파티션으로 전환하는 기법을 사용하거나, 이중화된 하드웨어에 파티셔닝을 적용 후 노드 단위의 파티션 간 절체를 수행하는 기법을 사용하였다. 이와 같은 결함 허용 기법들은 단일 하드웨어 상에서 파티션 간 실시간 절체를 적용하지는 못하였으며, 이를 위해서는 멀티코어와 같은 다중 처리기에 처리기별 파티셔닝을 적용하여 동시에 병렬로 처리 가능한 파티셔닝 개념이 요구된다. 또한 파티션간의 독립적 수행을 위해 AMP 구조의 코어 설정이 적용되어야 한다.

2. 멀티코어 기반 파티셔닝 분산 복구 블록 설계

2.1. 파티션 기반 분산 복구 블록 개념 설계

파티션 기반의 분산 복구 블록은 주 노드와 백업 노드를 각각의 파티션에 할당하고 두 개 파티션간의 통신 기능을 이용하여 기존 분산 복구 블록들을 연동시키는 구조로 설계되어야 한다. 다음 그림 4는 본 논문에서 제안하고자 하는 두 개의 복구 블록을 연동시키는 파티션 기반의 분산 복구 블록 설계 개념도를 나타낸다.

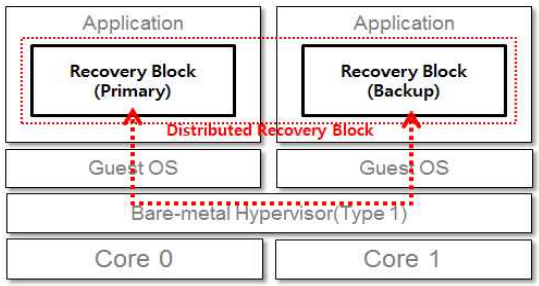

Fig. 4. Partition assignment diagram per each core
그림 4. 각 코어별 파티션 할당 다이어그램

위 그림 4와 같이 파티션 기반의 분산 복구 블록은 코어별 파티션에 할당된 Guest OS 상에서 Application의 일부로 동작하게 된다. 각 Application 내에서는 복구 블록으로 동작하며, 다른 파티션의 복구 블록과 연동하여 하나의 분산 복구 블록을 형성한다. 분산 복구 블록을 구성하기 위해서는 복구 블록 간 통신이 필요하며, 이를 위해서는 소켓통신과 같은 디바이스를 이용한 통신 방법을 이용하는 것이 가능하나, 파티션간의 통신에서는 ARINC 653에서 제공하는 Port를 이용하면 Guest OS를 경유하여 하이퍼바이저를 통해 외부 통신 인터페이스 없이 파티션간의 통신을 수행할 수 있다.

따라서 본 논문에서는 멀티코어를 지원하는 파티셔닝 운영체제에 분산 복구 블록을 적용하기 위하여 복구 블록 구조, 복구 블록 관리자 그리고 복구 블록 간의 통신방법에 대한 상세 설계를 제안한다.

2.2. 파티션 기반 분산 복구 블록 상세 설계

2.2.1. 복구 블록 구조 설계

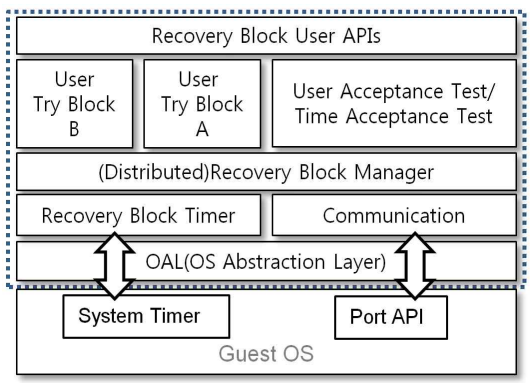

Fig. 5. Recovery Block Structure
그림 5. 복구 블록 구조

분산 복구 블록의 가장 중요한 구성요소인 복구 블록은 파티션 상에서 다음 그림 5와 같은 구조로 설계된다.

위 그림 5의 복구 블록 구조에서 볼 수 있듯이, 복구 블록은 사용자에 의해 작성되어야 하는 두 개의 Try Block을 가지고 있으며, 해당 Try Block의 결과를 검증할 수 있는 사용자 Acceptance Test 로직으로 구성된다. Application 실행 시, 첫 번째 Try Block을 수행 후, Acceptance Test로 결과를 검증하여 실패하면 두 번째 Block을 시도하는 방식으로 자체 복구 블록 기능을 수행하도록 설계하였다.

분산 복구 블록을 이용한 복구 스테이션으로 구성된 경우에는 주 노드의 모든 Try Block 실패 시, 결과를 백업 노드에게 알려주어 절체 하거나, 백업 노드가 일정 시간 이내에 주 노드의 결과가 성공으로 입력되는지 확인(Time Acceptance Test)하여 절체 여부를 결정한다.

위와 같이 분산 복구 블록의 결함 허용 기능을 제공하기 위해서는 주 노드와 백업 노드 사이의 통신 및 Deadline 처리 기능이 요구된다. 노드간 통신 기능은 이전 장절에서 언급한 ARINC 653의 Port를 이용하며, Deadline 처리를 위해서는 Guest OS에서 제공하는 System Timer 기능을 사용하게 된다.

본 논문에서는 향후 확장을 위해 Port및 Timer 기능을 Guest OS 종류에 관계없이 독립적으로 제공하기 위한 OAL(OS Abstraction Layer)을 이용하여 Recovery Block Timer 모듈, Communication 모듈과 연동시켜 필요한 OS 기능들을 복구 블록에서 사용할 수 있도록 구조를 설계하였다.

사용자는 Recovery Block User API를 통해 Try Block 및 Acceptance Test 로직을 구현하고, 파티션 간 통신에 관련된 파라미터 등을 설정하게 된다. 분산 복구 블록 기능을 관리하기 위한 (Distributed) Recovery Block Manager는 사용자가 작성한 User Try Block들과 Acceptance Test 로직을 연동하여 복구 블록 기능 및 분산 복구 블록 기능을 수행한다.

2.2.2 복구 블록 관리자 설계

복구 블록 관리자(Recovery Block Manager)는 분산 복구 블록 기능을 수행하기 위한 핵심 모듈로 복구 블록의 기능 및 주/백업 노드간 연동과 결함 판단을 통해 절체여부를 결정하는 역할을 수행한다. 다음 그림 6은 제안된 복구 블록 관리자 구조를 나타낸다.

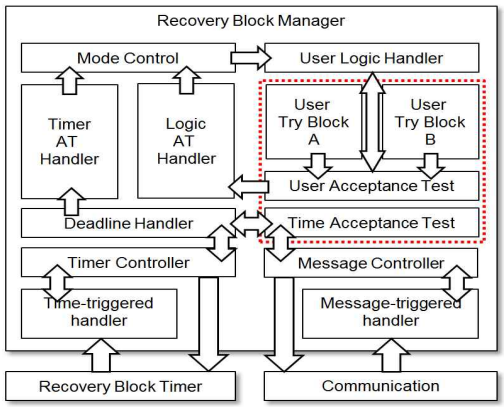

Fig. 6. Structure of Recovery Block Manager
그림 6. 복구 블록 관리자의 구조

위 그림 6의 구성을 살펴보면 복구 블록 매니저의 기능들은 크게 4가지 기능으로 구분되어 설계되었음을 알 수 있다. 먼저 사용자에 의해 작성되는 User Try Block과 Acceptance Test, 그리고 Recovery Block Timer에 의한 Deadline 처리 부분과 노드간 통신을 위한 통신 처리 부분, 마지막으로 Time AT(Acceptance Test)와 Logic AT의 결과를 기반으로 주/백업 모드를 관리하는 모드 제어 부분까지 총 4개의 기능부로 구성된다. 각 기능에 대한 설명은 다음과 같다.

- *User Try Block/AT* : Recovery Block User API를 이용하여 등록되는 사용자 처리 로직과 AT 처리 함수로, 처리 로직은 같은 결과를 제공하지만 처리 방식이 다른 함수로 구현해야 하며, AT는 처리 결과에 대한 검증을 수행하도록 작성되어야 한다. AT는 등록되어 있는 Try Block에 대해서 성공 또는 실패 여부를 판단하여 Mode Control 부분의 Logic AT에 결과를 보고한다.
- *Mode Control* : Logic AT와 Time AT의 결과를 종합하여 둘 중에 하나라도 실패가 발생하는 경우 Backup 노드가 제어권을 가져와 주 노드의 역할을 수행하도록 절체를 수행한다.
- *Time Control* : 백업 노드에서 사용자에 의해 정의된 Time AT 파라미터를 기준으로 로컬 Recovery Block 및 AT 처리 완료 후 일정 시간(Deadline) 이내에 상대 노드의 결과를 받아 Time AT 결과를 업데이트 한다. Time AT는 상대 노드의 실행 결과 또는 Timeout 여부로 결정된다.
- *Message Control* : 주/백업 노드간 AT 결과를 주고받기 위한 메시지 처리 모듈로, 전송 및 수신 처리기로 구성된다. 수신 처리기에서는 상대 노드의

결과 입력 시, Time AT 처리부로 전달한다.

2.2.3 분산 복구 블록 통신 방법 설계

멀티코어 기반 파티셔닝 운영체제인 VxWorks 653 3.0의 GuestOS는 가상의 하드웨어 위에서 동작하는 것처럼 구성되며 실제로는 하이퍼바이저에 의해 가상화된 환경에서 구동된다. 따라서 VxWorks 653 3.0의 하이퍼바이저인 Module OS의 ARINC 653의 서비스 중, 파티션간 통신이 가능한 Port를 이용하여 다음과 같은 구조로 설계가 가능하다.

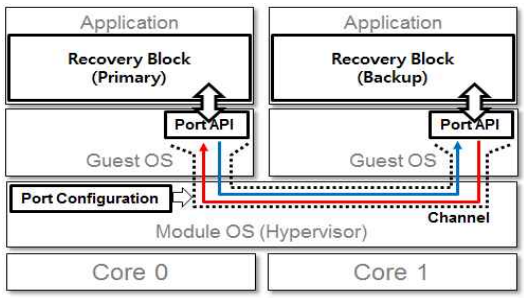

Fig. 7. Distributed Recovery Block Communication(Port)
그림 7. 분산 복구 블록 통신 방법(Port)

위 그림 7과 같이 Port를 이용하여 파티션간 통신을 하기 위해서는 Guest OS에서 Port 사용을 위한 API를 제공하고, Module OS상에서의 Port Configuration을 제공해야 한다. Port Configuration은 통신을 수행해야 하는 파티션들을 지정하여 해당 파티션간에 Channel을 구성하게 한다. VxWorks 653 3.0에서는 Module OS에서 지정된 Channel을 통해서만 파티션간 통신이 가능하다.

VxWorks 653 3.0에서 파티션간 통신을 위해 제공되는 Port는 크게 두 가지 종류로 구분되며, 다음과 같다.

- *Queuing Port* : Queuing Port는 파티션간 데이터를 주고받기 위한 포트 중 메시지의 전달의 신뢰성에 높은 비중을 갖는 통신 방식이다. Guest OS는 하이퍼바이저에 의해 제공되는 채널을 할당받고 해당 채널을 통해서 필요한 데이터들을 Queue 방식으로 전송하거나 수신할 수 있다.
- *Sampling Port* : Sampling Port는 Queuing Port와 마찬가지로 채널을 기반으로 하며, Queue 방식이 아닌, Overwrite 방식으로 갱신되는 형태로 데이터를 공유 한다.

분산 복구 블록에서 전송되는 데이터들은 패킷단위의 보장 및 실시간성이 모두 보장되어야 한다. Sampling Port는 패킷단위의 보장이 힘들기 때문에 본 논문에서는 Queuing Port를 사용한 분산 복구 블록 구조로 설계를 제안한다. 다만, Queuing Port의 실시간성에 대한 특성을 예측하기 위하여 Queuing Port를 통해 전송되는 Payload 크기별 전송 지연시간의 경향을 측정하여 분석을 수행하였다.

3. 비행제어 시스템을 위한 분산 복구 블록 설계 적용

3.1. 비행제어 시스템(FLCC) 사용자 구현부 설계

비행제어 시스템은 항공기의 각종 센서들로부터 신호를 입력받아 처리한 후 항공기의 자세를 제어하는 역할을 수행하는 장비이다. 해당 장비는 크게 신호 입력 부와 처리부 그리고 제어부로 구성되며, 본 논문에서는 신호 처리부의 일부인 자세 제어 처리 모듈을 분산 복구 블록을 이용하여 처리할 수 있도록 다음 그림8과 같이 사용자 구현 부분을 설계하였다. 설계 조건에서 센서로부터 입력되는 신호는 25Hz 주기로 동작한다고 가정하여 비행제어 시스템의 Time AT(Acceptance Test)의 Deadline은 40ms로 결정하였다.

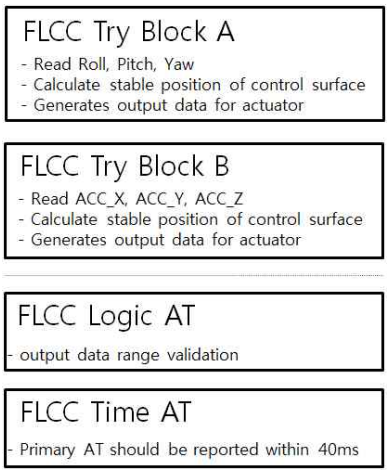

Fig. 8. FLCC User Try Block/AT Design
그림 8. FLCC 사용자 구현부 설계

위 그림 8의 설계에서 볼 수 있듯이, User Try Block A/B, 그리고 Logic AT, Time AT 부분으로 나누어 작성된다. 각 부분별 내용은 아래와 같다.

- *User Try Block A* : 항공기의 자세 데이터 중, Roll, Pitch, Yaw(Heading) 값을 읽고, 안정 자세(또는 목표 자세)와의 차이 값을 계산하여 해당 차이만큼의 제어를 수행할 수 있는 제어 값을 출력한

다.

- *User Try Block B* : Block A가 실패할 경우, 다른 소스인 가속도계 값(ACC)을 축 별로 입력받아 (X, Y, Z) 해당 값을 이용하여 자세를 계산하고, Block A와 마찬가지로 차이 값과 제어 값을 출력한다.
- *Logic AT* : 제어 값의 제어 범위를 이용하여 잘못된 계산 여부를 판단한다.
- *Time AT* : 주 노드는 40ms 이내에 AT를 마치고 결과를 백업 노드에게 전송하여야 한다.

3.2. 비행제어 시스템(FLCC) 분산 복구 블록 구조

앞서 설계한 사용자 구현부를 기반으로 분산 복구 블록을 지원하기 위한 전체 구조는 다음 그림 9와 같다.

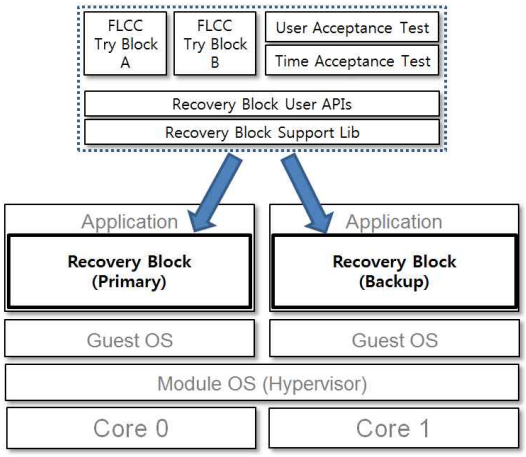

Fig. 9. FLCC Distributed Recovery Block Design
그림 9. FLCC 분산 복구 블록 설계

위 그림 9와 같이, 사용자 구현부를 제외한 나머지 3개 부분은 모두 Recovery Block Support Library형태로 표현하였으며, 노드 구분을 위하여 Core 0와 Core 1에 개별 파티션으로 할당하도록 설계하였다. 이와 같은 설계 기법은 Application 특성에 따라 다양한 응용이 가능하며, 복잡한 Application 이라도 Recovery Block을 모듈단위로 구성하면 다양한 응용이 가능하다는 장점이 있다.

3.3. 분산 복구 블록 Time AT의 Deadline 조건

제안된 비행제어 시스템을 위한 분산 복구 블록의 백업 노드의 Time AT 시간은 아래의 조건을 만족해야 한다.

$$T_{Tat} > T_{pri} + T_{com} \quad (1)$$

(단, $T_{pri} = T_{UTa} + T_{Lat} + T_{UTb} + T_{Lat}$)

위 수식 (1)의 T_{Tat}는 Time AT의 Deadline을 의미하며, T_{pri}는 주 노드의 분산 복구 블록 최대 처리 시간(WCET, Worst Case Execution Time)을 의미한다. T_{UTa}와 T_{UTb}는 User Try Block A, B WCET이며, T_{Lat}는 Logic AT WCET를 의미한다. 따라서 그림 8의 설계에서 제시된 Time AT의 Deadline에 의해 다음과 같은 조건을 만족해야 한다.

$$40ms(T_{Tat}) > T_{pri} + T_{com} \quad (2)$$

4. 시뮬레이터 기반 시험 및 결과 평가

4.1. 시뮬레이터 기반 시험 환경

3절에서 설계 한 FLCC 분산 복구 블록의 실시간 성능 시험을 위한 시험 환경은 오픈소스 기반 비행 시뮬레이터인 'Flight Gear'를 이용하여 구성하였다. 'Flight Gear'는 내부에 항공기의 상태 정보를 생성하는 모델을 탑재하고 있어, 실시간 항공기 정보를 생성하는 것이 가능하다. 다음 그림 10은 시험 환경 구성도를 나타낸다.

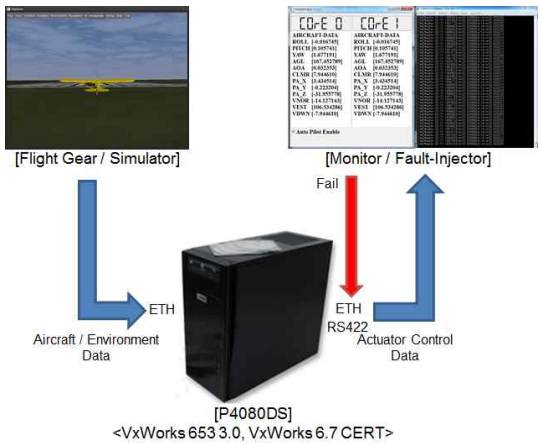

Fig. 10. FLCC Simulation Test Environment
그림 10. FLCC 시뮬레이션 시험 환경

그림 10의 구성은 'Flight Gear' 시뮬레이터와, FLCC를 모사하기 위한 FreeScale사의 P4080DS (Octa-Core) 개발 보드 그리고 모니터링을 위한 모니터 프로그램과 콘솔 프로그램을 사용하였다. 모니터

링 프로그램은 Core 0(파티션 1)과 Core 1(파티션2)의 상태를 모니터링 하는 것이 가능하며, 주 노드인 Core 0에 결함을 주입하는 것이 가능하다.

4.2. 시험 조건

본 시험에서는 주 노드의 AT결과가 40ms 이내에 백업 노드에서 확인되는 것을 목표(T_{Tat})로 하고 있다. 노드별 동작 주기를 40ms로 설정하기 위해 'Flight Gear'의 출력 주기를 50Hz(20ms, 2배)로 설정하여 입력 패킷의 손실이 없도록 하였다. P4080DS의 Core별 처리 속도(1.5GHz)와 프로그램 복잡도를 고려하여 $T_{pri} \leq 1ms$로 가정하였고, T_{com}은 다음 그림 11의 파티션간 Queuing Port 통신 시험결과를 이용하여 메시지의 크기를 기준으로 산정($T_{com} \leq 1ms$)하였다.

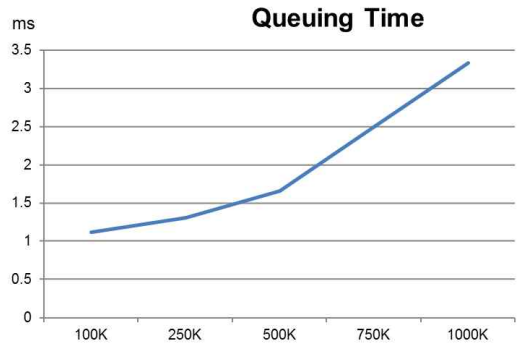

Fig. 11. Queuing Port Time Measurement
그림 11. Queuing Port 시간 측정

'Flight Gear'의 전송 주기를 기준으로 WCET를 아래와 같이 산정할 수 있으므로, 수식 (2)조건은 만족된다.

$$40ms > 20ms(T_{fg}) + 1ms(T_{pri}) + 1ms(T_{com}) \quad (3)$$

위 수식(3)의 T_{fg}는 'Flight Gear'의 비행데이터 생성 주기(20ms) 시간을 의미하며, 복구 블록이 수행되는 시점에서 최대 20ms 후에는 패킷이 입력된다는 것을 의미한다.

4.3. 시험 비교군 및 시험 결과

본 논문에서 제안하는 멀티코어 기반의 파티션 운영체제를 위한 분산 복구 블록의 가용성을 측정하기 위해 싱글코어 기반의 복구 블록과 멀티코어 기반의 복구 블록에 대한 비교 시험을 수행하였다. 파티션 구성은 다음 그림 12와 같으며, 싱글코어 및 멀티코어 기반 분산 복구 블록 모두 Partition 1과 2에 각각 40ms의 시간을 할당하였다('Flight Gear'에서 전송되는 누적된 패킷은 사용하지 않음). 단, 싱글코어 기반의 분산 복구 블록은 Partition 1과 2를 모두 수행하는데 80ms의 시간이 소요된다.

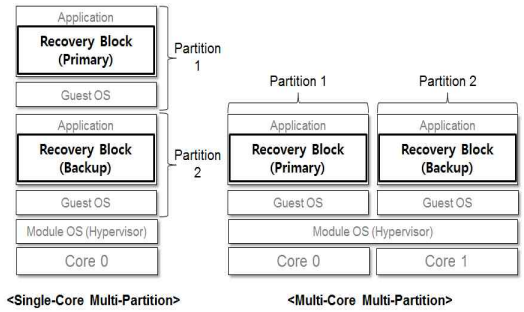

Fig. 12. Comparison Test Condition
그림 12. 비교 시험 조건

위 그림 12와 같은 비교 조건에서 시험을 수행하였을 때 시험 결과는 다음과 같다. 세로축의 각 출력

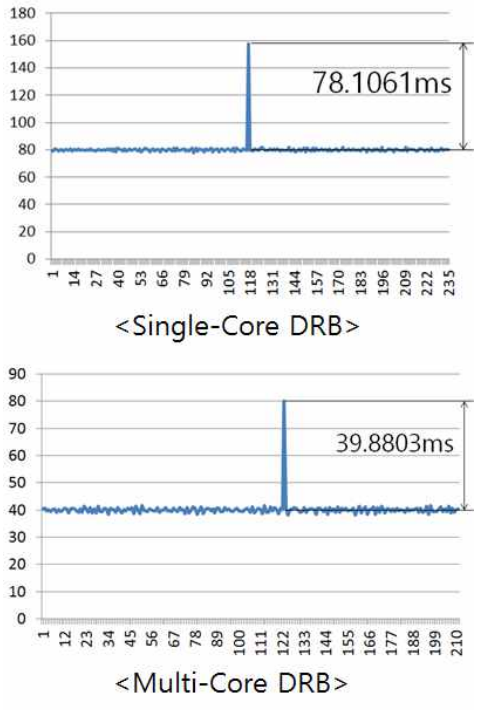

Fig. 13. Result of comparison test
그림 13. 비교 시험 결과

값은 이전 출력과 현 출력의 시간차이를 의미하여 외부 모니터 프로그램에 의해 측정된 시간이다. 가로축은 측정 횟수를 의미한다.

위 그림 13의 결과를 보면, 싱글코어 기반의 분산 복구 블록은 코어 시간을 나누어 동작되는 이유로 같은 동작 조건(40ms)에서 멀티코어 기반의 분산 복구 블록 보다 평균 약 2배의 출력 시간 차이(80ms / 40ms)가 발생하며, 주 노드에서 백업 노드로의 절체에도 약 2배 정도의 시간이 더 소요됨(78.1061ms / 39.8803ms)을 알 수 있다. 비교 실험을 통하여 초기 동작 주기 설계 요건(40ms)을 기준으로 싱글코어 기반의 분산 복구 블록은 해당 조건을 만족하지 못하며, 멀티코어 기반의 분산 복구 블록은 해당 조건을 만족하는 것을 확인할 수 있다.

III 결론

본 논문에서는 소프트웨어 결함 허용 기법 중 분산 복구 블록 설계 기법을 멀티코어 기반의 파티셔닝 운영체제에 적용하기 위한 설계 기법 및 응용에 관한 연구를 수행하였다. 단일 하드웨어 기반의 모듈화 특성을 갖는 파티셔닝 운영체제의 특성에 맞는 분산 복구 블록 설계 구조를 제안하였으며 시뮬레이터를 이용한 테스트베드를 활용하여 제안된 분산 복구 블록의 유용성을 검증하였다. 그리고 실험을 통해 싱글코어 기반의 분산 복구 블록에서는 설계 동작 주기인 40ms의 두 배인 80ms가 소요되었으나 멀티코어 기반의 분산 복구 블록에서는 설계 주기대로 40ms가 소요되는 것을 확인하여, 싱글코어에서의 분산 복구 블록 적용의 어려움 및 멀티코어에서의 분산 복구 블록 적용의 가용성을 확인하였다.

제안된 설계 구조는 2중화 구조의 분산 복구 블록만을 지원하는 한계를 가지고 있으며, 향후 3중화 이상의 분산 복구 블록 적용을 위한 다중 분산 복구 블록에 대한 연구를 수행할 계획이다.

References

[1] Morgan, M.J., "Integrated modular avionics for next-generation commercial airplanes", Proceedings of the IEEE 1991 National, pp43-49 vol.1. NAECON, 1991.

[2] Wind River Systems, "ARINC 653 - An Avionics Standard for Safe, Partitioned Systems". IEEE Seminar. August 2008. Retrieved 2009-05-30.

[3] Lee. J, T. Kim. J, G. "TMO.p Model and its scheduler for partition computing", Journal of KIISE, Vol 18-11, pages 733-741, 2012.11.

[4] Kim, K. H, Welch, H. O, "Distributed Execution of Recovery Blocks: An Approach for Uniform Treatment of Hardware and Software Faults in Real-Time Applications", IEEE Transactions on Computers, Vol. 38, No. 5, Pages 626-636, May 1989.

[5] Hansol Park, "Fault-tolerant design technique and application for high-reliable real-time systems based on the distributed recovery block", Ph.D thesis, Department of Computer Science, Konkuk University, 2013.

[6] Ko. Y, G. et al., "HM System Design for Fault Tolerance on the IMA System", Journal of KOCON, Vol 12-8, pages 77-86, 2012.8,.

[7] Jung B. Y, Kim J. G, "A Fault-tolerant Inertial Navigation System for UAVs Based on Partition Computing", KIISE Transactions on Computing Practices, Vol.21, No. 1, pp. 29-39, 2015.

BIOGRAPHY

Hansol Park (Member)

2004 : BS degree in Aerospace Engineering, Konkuk University.
2006 : MS degree in Computer Engineering, Konkuk University.
2013 : PhD degree in Computer Engineering, Konkuk University.
2008~2012 : Research Engineer, Agency for Defense Development.
2012~2015 : Senior Engineer, Hanwha Thales.