RAID 시스템의 모델링 및 시뮬레이션

이찬수*, 성영락**, 오하령***

Lee, Chan-Su, Seong, Yeong-Rak and Oh, Ha-Ryoung

Modeling and Simulation of a RAID System

Abstract

In this paper, a RAID system is modeled and simulated by using the DEVS formalism. The RAID system interacts with a host system by using the high-speed Fibre channel protocol and stores data in an array of IDE disks. The DEVS formalism specifies discrete event systems in a hierarchical, modular manner. The RAID system model is composed of three units: primary-PCI unit, secondary-PCI unit and MCU unit. The primary-PCI unit interfaces with the host system and I/O data caches. The secondary-PCI unit includes disks. The MCU unit controls overall system. The control algorithm of MCU and PCI transactions are analyzed and modeled. From the analysis of simulation events, we can conclude that the proposed model satisfies given requirements.

Key Words: RAID, DEVS 형식론, 시뮬레이션, 데이터저장시스템

* 국민대학교 공과대학 전자공학과 박사과정

** 국민대학교 부교수

*** 국민대학교 공과대학 전자공학부 교수

1. 서 론

오늘날 산업 전 부분이 인터넷 환경으로 전환되면서 개인 데이터는 물론 기업의 각종 데이터가 기하급수적으로 증가하고 있다. 인터넷 서비스의 증가로 인터넷 서비스 제공업체(Internet Service Provider: ISP)나 응용 소프트웨어 제공업체(Application Software Provider: ASP) 등이 급증하여 고객들의 정보를 효율적으로 저장할 수 있는 대용량 저장 장치의 구축을 필요로 한다. 또한 백업 서비스와 재해 복구 시스템의 수요도 점차 증가하고 있으며, 호스트 시스템과 저장 장치를 고속의 FC(Fibre Channel)[1]로 연결하는 SAN(Storage Area Network)과 NAS(Network Attached Storage) 등 네트워크 저장 장치[2]의 등장으로 고속 대용량 저장 장치의 구축이 절실히 요청되고 있다.

그러나, 저장 장치 분야의 기술 개발은 저장 용량면에서는 기하급수적인 성장을 하고 있는데 반해, 성능면에서는 매년 7~10 퍼센트의 성장에 머무르고 있다. 이것을 보완하고자 여러 개의 디스크를 동시에 접근함으로써 다수의 디스크가 하나의 빠르고 큰 논리적인 디스크로 보이게 하는 RAID(Redundant Array of Inexpensive Disks) 시스템[3]에 대한 많은 연구가 진행되고 있다. 기존의 많은 연구에서는 RAID 시스템의 성능이 분석되었다. Chen과 Towsley는 RAID 수준 1과 수준 5에 대해서 스케쥴링 알고리즘 및 매핑 알고리즘을 개발하고 성능을 비교하였다[4]. Merchant와 Yu는 매핑 알고리즘을 개발하고 해석적인 모델을 만들어서 성능을 분석하였다[5]. Reddy와 Banerjee는 다양한 부하에서 디스크 구성에 따른 성능을 분석하였다[6]. 국내에서도 수 년 전부터 RAID 시스템에 대한 연구가 활발히 진행되었다. 그 중 [7] 연구에서는 RAID 시스템의 재건 성능을, [8] 연구에서는 비선형 편집기를 위한 병렬 디스크 시스템을 시뮬레이션 기법을 이용하여 분석하였다.

큰 대역폭이 요구되거나 실시간 운영이 요구되는 RAID 시스템은 평면 구조로는 필요한 대역폭을 만족시킬 수 없어 계층적 구조의 형태를 가지는 경우가 많다. 이런 시스템은 성능에 영향을 미치는 여러 가지 파라미터들이 서로 연관되어 최적값을 선택하기 어려울 뿐 아니라 그 성능을 예측하기가 쉽지 않아서 제품 개발 시간과 더불어 최적 시스템을 구현하기가 어렵다. 그러나 형식론적인 모델링을 통한 시뮬레이션은 RAID 시스템의 성능을 빠르고 쉽게 예측할 수 있다. 본 논문에서는 DEVS 형식론을 이용하여 제안된 시스템을 기술하고, 이산사건 시뮬레이션 언어중의 하나인 DEVSim++[9]를 이용하여 시뮬레이션 하였다.

DEVS 형식론은 이산 사건 시스템을 기술하는 수학적 형식론으로서 계층적이고 모듈화 된 방법으로 시스템을 기술한다[10]. 이때 각각의 구성요소들은 원소형 모델로 표현되며 계층적인 구성은 결합형 모델로 나타낸다. 원소형 모델은 외부 사건 또는 내부 시간 초과에 의해 상태가 천이 하는 동적 모델에 해당하며, 출력은 내부 시간 초과에 의해 상태가 천이 될 때에 발생된다. 이러한 출력은 다른 모델의 외부 사건을 발생시켜 다른 모델의 상태 천이를 발생시킨다. 결합형 모델은 그 인터페이스는 원소형 모델과 같지만 구성은 원소형 모델 또는 다른 결합형 모델의 집합으로 되어 있다. 각 모델들은 입력과 출력으로 연결되며, 유한한 상태를 가지고, 스스로 발생시키는 사건과 외부에서 발생하는 수동적인 사건들로 상태가 천이 된다.

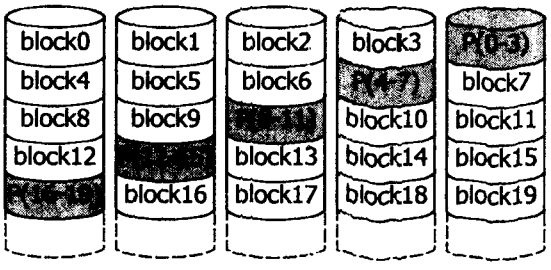
본 논문의 구성은 다음과 같다. 2절에서는 RAID 시스템의 일반적인 특징과 각 수준별 특징에 대해 살펴본다. 3절에서는 본 논문에서 기술한 RAID 시스템의 모델과 각 구성요소들의 역할에 대해 살펴본다. 4절에서는 PCI 트랜잭션의 모델에 대해 상세히 분석하며, 5절에서는 디스크 읽기 트랜잭션을 수행 한 시뮬레이션 결과를 나타내었으며, 6절은 결론이다.

2. RAID 시스템

RAID 시스템은 독립적으로 운용되는 여러

개의 디스크들을 하나의 저장장치처럼 다룰 수 있게 하고, 장애가 발생했을 때 데이터를 잃어버리지 않게 할 수 있다. 1988년 발표된 논문 [11]은 데이터와 패리티 정보를 디스크에 배치하는 방법에 따라 디스크 어레이를 분류하고 있는데, 이것은 이후 RAID 수준이라고 불리게 된다. 초기 RAID의 기본적인 개념은 작고 값싼 디스크들을 연결해서 크고 비싼 디스크 하나(SLED: Single Large Expansive Disk)를 대체하자는 것이다[12]. 그럼으로써 데이터 가용성과 총 저장용량을 증가시키며 여러 물리적 디스크에 데이터를 적절히 분산시킴으로서 효율성을 재고시키는 것이다.

그동안 많은 연구를 통해서 RAID는 수준 0~5 등 여러 방식이 존재한다.[2, 11] 본 논문에서는 RAID 수준 5에 기반한 시스템을 모델링한다. <그림 1>은 RAID 수준 5를 도식화 한 것이다. 그림에서는 5개의 디스크를 사용하고 있으나 디스크의 개수는 더 많거나 적을 수 있다. 각각의 디스크는 일정한 크기의 블록으로 분할된다. 블록 번호는 여러 디스크에 나뉘어 할당되어 외부에서 볼 때에는 전체 디스크들이 하나의 논리적인 디스크처럼 보인다. RAID 수준 5는 독립처리 방식을 이용한다. 이는 각 디스크들의 작동이 서로 무관하다는 것이다. 한 디스크가 읽기 동작을 할 때, 다른 디스크들은 쓰기 동작을 할 수도 있고, 대기 상태로 있을 수도 있고, 또다른 읽기 동작을 할 수도 있다. 이 방식은 데이터의 전송 속도보다는 여러 사용자들의 작업 요구를 처리하는 능력이 요구될 때 사용된다. 그림에서 어둡게 표시된 부분은 패리티가 저장되는 블록이다. 패리티는 사용자의 데이터가 아닌 부가 정보로 저장된 데이터의 일부분이 손상되었을 때 원래의 데이터를 복원할 수 있게 한다. 또한 패리티 정보를 모든 디스크에 나누어 기록하므로 패리티를 담당하는 디스크가 병목현상을 일으키지 않는다.

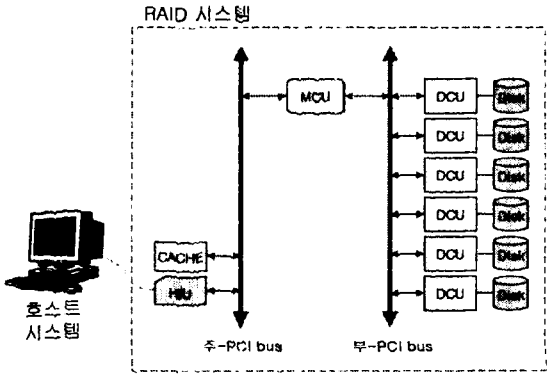


<그림 1> RAID 수준 5

3. RAID 시스템 모델

본 절에서는 본 논문의 대상이 되는 RAID 시스템의 구조를 제시하고 모델링한다. 현재까지 매우 다양한 종류의 RAID 시스템이 개발되었으며 하드웨어 구조 또한 다양하다. 본 논문에서 모델링 대상이 되는 RAID 시스템은 호스트 시스템과는 FC 인터페이스를 가지며 RAID 수준 5를 지원하는 시스템이다. FC 인터페이스를 가정한 것은 i)FC망이 매우 고속이며, ii)일반적인 호스트 시스템들에서 하드디스크의 인터페이스로 사용되는 SCSI를 상위 프로토콜로 탑재할 수 있으며, iii)현재의 많은 상업용 RAID 시스템들에서 적용되고 있는 기술이기 때문이다

<그림 2>는 본 연구의 대상이 되는 RAID 시스템의 하드웨어 구조이다. 효율적인 호스트 인터페이스와 다수의 디스크를 지원하기 위하여 두 개의 PCI 버스를 가지며 그 두 버스는 MCU (Master Control Unit)를 통해 연결된다. MCU는



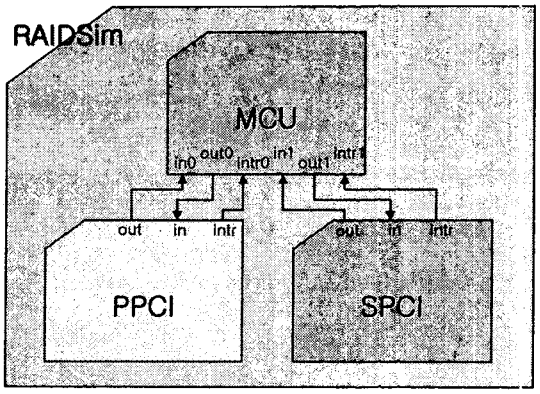
<그림 2> RAID 시스템 구조

호스트로부터 전달된 SCSI 메시지를 적절히 변환하여 각각의 DCU(Disk Control Unit)에 분배하여 DCU로 하여금 하드디스크를 제어하도록 한다. 또한 필요시에는 패리티 계산을 담당한다. 이러한 기능을 지원하는 MCU 중의 하나인 인텔사의 i960 RM/RN 프로세서[13]를 이용한 시스템을 모델링 하였다.

주-PCI 버스는 호스트 인터페이스 및 캐쉬 기능을 제공한다. HIU(Host Interface Unit)는 FC 인터페이스와 PCI 버스 인터페이스를 가지고 있으면서 FC 프로토콜을 통하여 전달된 SCSI 명령을 주-PCI 버스를 통하여 MCU로 전달한다. 그러면 MCU는 캐쉬의 주소를 지정하여 HIU로 알려주고, HIU는 DMA로 지정된 주소의 데이터를 읽어서 호스트로 보내거나, 호스트로부터 전달된 데이터를 지정된 주소에 쓴다. 본 논문에서는 MCU와 HIU간의 메시지 전송은 MCU의 로컬 메모리를 이용하여 버퍼링되는 것으로 간주한다. 그러므로 MCU와 HIU는 상대방의 상태에 무관하게 메시지를 전송할 수 있다. 캐쉬는 호스트 인터페이스 속도와 하드디스크 속도의 차이를 완화시켜주는 역할을 하는 임시 데이터 저장장소로서 주-PCI 버스에서는 목적자(target)의 역할만 한다.

부-PCI 버스에는 하드디스크들을 제어하는 DCU들과 MCU를 연결한다. DCU에서는 MCU가 부-PCI 버스를 통해 전달한 디스크 조작 명령에 따라 하드디스크를 제어하고 하드디스크와 MCU 사이의 데이터 전송을 중개한다. 하드디스크는 DCU의 종류에 따라서 IDE 혹은 SCSI 디스크가 연결될 수 있으나, 본 논문에서는 저렴한 IDE 디스크를 가정하였다. 본 논문에서는 MCU와 HIU와의 인터페이스와는 달리 DCU와 MCU간에는 명령들이 버퍼링되지 않는 것으로 간주하였다. 그러므로 MCU는 DCU가 휴식 상태임을 확인한 후에 다음 명령을 보내어야 한다. 이렇게 HIU-MCU, DCU-MCU 인터페이스를 다르게 가정한 것은 현재 판매되고 있는 유사한 기능을 가지는 소자들의 특성을 감안한 것이다. 그러나 버퍼링 기능을 가지는 DCU의 경우에 대해서도 본 연구의 내용을 쉽게 변경하여 적용할 수 있다.

<그림 3>은 <그림 2>의 시스템을 모델링한 것이다. RAID 시스템은 크게 MCU, PPCI, SPCI로 나뉘어 구성된다. 모델링한 각 부분은 실제 시스템의 하드웨어, 소프트웨어적인 면을 추상화하고 개념화하여 각각의 동작특성을 기술한다. 각각의 상세한 모델은 다음과 같다.



<그림 3> RAID 시스템 모델

3.1 MCU

MCU는 HIU로부터 전달받은 SCSI 요청 메시지를 분석하여 명령에 따라 캐쉬 히트 여부를 조사하고, 필요시 패리티를 생성하며, 각 RAID 수준에 따라 적절히 디스크를 매핑하여 데이터를 분배 혹은 취합한다. 입출력이 빠르게 처리되는 캐쉬나 버퍼링 기능을 가지고 있는 HIU와는 달리 DCU는 하나의 명령이 완전히 처리된 후에야 비로소 다음 명령을 처리할 수 있기 때문에 이를 위한 조정 작업도 담당한다. 또한 하나의 SCSI 명령이 처리되고 나면 호스트 시스템으로 응답 메시지를 보내어 다음 명령을 처리할 준비를 한다. 이처럼 MCU에서는 HIU로부터 받은 명령을 처리하기 위해 명령을 분석하고, 메모리 요청 메시지, IDE 요청 메시지, SCSI 응답 메시지 등 여러 메시지들을 생성하고 전달한다.

MCU의 동작을 보다 자세히 설명하기 위하여 PCI 버스에서 전달되는 메시지들을 <개시자, 목

적자, 메시지>로 표현하자. 여기서 개시자(initiator)는 PCI 버스의 트랜잭션을 요구하는 장치이고 목적자(target)는 요구의 대상이 되는 장치이다. 전달되는 메시지는 (유형, 시작주소, 크기)로 표현된다. <표 1>은 전달되는 메시지의 유형이다.

<표 1> 전달 메시지의 유형

메 시 지	의 미
(SCSI-READ,n,m)	호스트가 n블록부터 m개의 블록에 대한 읽기를 요청
(SCSI-WRITE,n,m)	호스트가 n블록부터 m개의 블록에 대한 쓰기를 요청
(IDE-READ,n,m)	하드디스크의 n블록부터 m개의 블록에 대한 읽기를 요청
(IDE-WRITE,n,m)	하드디스크의 n블록부터 m개의 블록에 대한 쓰기를 요청
(MEM-READ,n,m)	캐쉬의 n블록부터 m개의 블록에 대한 읽기를 요청
(MEM-WRITE,n,m)	캐쉬의 n블록부터 m개의 블록에 대한 쓰기를 요청

```
unit algorithm
(1) HIU  receive a read request from host;
(2) HIU  send the request to MCU;                    // <HIU, MCU, (SCSI-READ, 0, 5)>
(3) MCU  if ( the request is read )
(4) MCU     if ( the data is already stored in the cache )
(5) MCU        send the cache address to HIU;
(6) HIU        transfer data from cache;
(7)         else
(8) MCU        send disk read requests to DCUs; // <MCU, DCU0, (IDE-READ, 0, 1)>
                                                // <MCU, DCU1, (IDE-READ, 0, 1)>
                                                // <MCU, DCU0, (IDE-READ, 1, 1)>
                                                // <MCU, DCU2, (IDE-READ, 1, 1)>
                                                // <MCU, DCU1, (IDE-READ, 2, 1)>
(9) DCU        transfer data to MCU;
(10) MCU       transfer data to cache;           // <MCU, CACHE, (MEM-WRITE, 0, 5)>
(11) MCU       send the cache address to HIU;    // <MCU, HIU, (SCSI-READ, 0, 5)>
(12) HIU       data transfer from cache;
(13)        endif
(14)     endif
```

<그림 4> 읽기동작에 대한 시스템제어 및 데이터흐름

호스트는 다양한 종류의 명령을 RAID 시스템으로 보내지만 대부분은 블록 읽기와 블록 쓰기이다. 본 논문에서는 블록 읽기를 중심으로 설명한다. <그림 4>는 RAID 시스템 내에서 이루어지는 제어 알고리즘이다. 설명을 위해 알고리즘의 오른쪽에는 호스트가 0번 블록에서부터 연속적으로 5개의 블록에 대한 읽기 요구를 발생한 경우에 PCI 버스로 전달되는 메시지를 나타낸 것이다.

<그림 4>의 (1)에서 HIU는 호스트로부터 메시지를 전달받아 이를 MCU에 전달한다(2). MCU는 이 메시지가 읽기 메시지인지 검사하고(3), 읽기 메시지인 경우 MCU는 해당되는 데이터가 이미 캐쉬에 있는지 검사한다(4). 이미 캐쉬에 데이터가 있으면, MCU는 바로 캐쉬상의 데이터가 위치한 주소를 HIU로 보내어 HIU로 하여금 데이터를 읽어 갈 수 있게 한다(5~6). 반면, 캐쉬에 데이터가 없을 때에는 디스크로부터 데이터를 읽어 들이기 위해 필요한 메시지들을 만들어 각각의 DCU로 보낸다(8). 이 메시지들은 RAID 수준에 따라 실제 데이터가 저장된 디스크들의 물리적 주소를 매핑하여 데이터를 가져올 수 있게 한다. 여기서는 3개의 DCU를 가지며 RAID 수준 5로 동작하는 경우를 예로 들었다. 그러므로 데이터 블록 0~1번은 DCU0, DCU1에, 그리고 DCU2에는 0~1번 블록에 대한 패리티 정보가 저장되고, 데이터 블록 2~3번은 DCU0, DCU2에, 그리고 DCU1에는 2~3번 블록에 대한 패리티 정보가 저장되며, 데이터 4번 블록은 DCU1번에 저장된다. 이 메시지를 받은 DCU는 디스크 읽기 동작에 의해 읽어들인 데이터를 MCU로 전달하고(9), MCU는 이를 캐쉬에 저장한다(10). 그리고 HIU로 하여금 데이터를 읽어갈 수 있도록 MCU는 캐쉬 주소를 HIU로 보낸다(11). HIU는 이 주소의 데이터를 읽어감으로써 읽기 동작이 완료된다(12).

위의 예에서 나타낸 것처럼 MCU가 HIU로부터 메시지를 받으면, 그 메시지를 처리하기 위해 어떤 종류의 메시지를 생성하여 어떤 장치로 보내야 하는지는 정적으로 판단된다. 그러나 각 메시지의 전송 시간은 메시지의 처리가 진행됨에 따라 동적으로 결정된다. 본 논문에서는 이를 그래프로 표현하여 MPG(Message Precedence Graph)라고 정의한다. 위의 예에서 MPG를 따로

나타내면 <그림 5>와 같다. 즉, MCU는 DCU들로 보낸 메시지들이 모두 완료되면 CACHE로 메시지를 보내고, 또 그것이 완료되면 HIU로 메시지를 보내야 한다. MPG에서 같은 단계의 메시지들이라 하더라도 각 메시지의 처리 종료시간은 그 목적자의 상태와 해당 PCI 버스의 상태에 따라서 다르게 결정된다.

<MCU, DCU1, (IDE-READ, 0, 1)>
<MCU, DCU2, (IDE-READ, 0, 1)>
<MCU, DCU1, (IDE-READ, 1, 1)>
<MCU, DCU3, (IDE-READ, 1, 1)>
<MCU, DCU2, (IDE-READ, 2, 1)>
↓
<MCU, CACHE, (MEM-WRITE, 0, 5)>
↓
<MCU, HIU, (SCSI-READ, 0, 5)>

<그림 5> MPG(Message Precedence Graph) 예

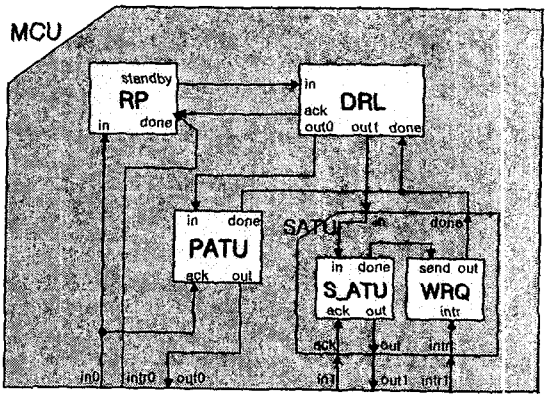
MCU의 일련의 동작들을 표현하기 위해 <그림 6>과 같이 모델링 하였다. MCU 모델은 크게 RP(Request Pool), DRL(Device Request List), PATU(Primary Address Translation Unit), SATU(Secondary Address Translation Unit)로 구성된다.

RP 모델은 HIU로부터 MCU로 SCSI 메시지가 전달되었을 때 그 메시지를 분석하여 MPG를 생성하고 각 메시지들이 처리 가능한 상태가 되었을 때 DRL로 보낸다.

DRL 모델은 MCU와 연결된 각 장치들을 큐로 관리한다. 각각의 큐에는 해당 장치로 보내어져야 할 메시지들이 순서대로 관리된다. 어떤 장치가 사용 가능한 상태이고 그 장치의 큐에 메시지가 있을 경우, 큐의 첫 번째 메시지를 꺼내어 그 장치가 주-PCI 버스에 연결된 경우에는 PATU로, 아닐 경우 SATU로 메시지를 보낸다. <그림 2>에서 나타난 바와 같이 주-PCI 버스에는 HIU와 캐쉬가, 부-PCI 버스에는 DCU들이 연결되어 있다.

PATU와 SATU는 각각 주-PCI 버스와 부-PCI 버스와의 인터페이스를 담당한다. 이를 위하여 각각의 내부에는 큐를 두어서 차례대로 메시지들을 PCI 버스를 통하여 전달한다. 앞서 언급한대로 HIU는 버퍼링 기능을 가지는 데에 비하여 DCU는 버퍼링 기능이 없다. 이런 차이로 인하여 SATU는 다시 S_ATU(Secondary Address Translation Unit)와 WRQ(Wait Response Queue)로 나뉜다. S_ATU는 PATU와 동등한 모델이며, WRQ는 각 DCU로 보내어진 메시지가 완료되는지를 확인한다.

각 메시지들에 대한 처리가 완료되었을 경우에는, 요청 메시지와는 반대로, PATU(SATU), DRL, RP의 순으로 메시지가 전달된다. 이때 PATU, SATU는 다음 메시지를 PCI 버스로 전송하고, DRL은 해당 장치의 큐를 휴식상태로 바꾸어 다음 메시지를 PATU나 SATU로 전달하며, RP는 MPG의 해당 메시지를 삭제하고 다음 단계의 메시지들을 스케줄 한다.

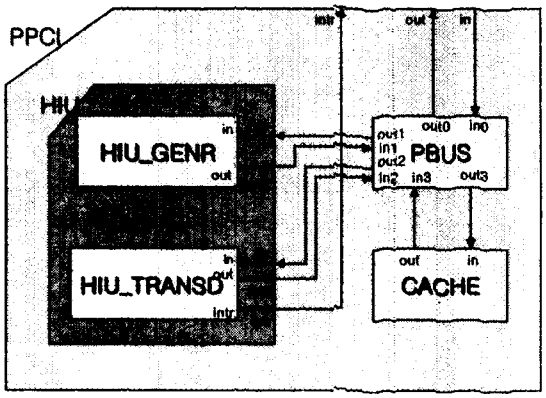


<그림 6> MCU 모델 구성도

3.2 PPCI

PPCI는 <그림 7>과 같이 HIU, CACHE, PBUS로 구성된다. HIU는 FC 프로토콜로 외부 호스트와 데이터를 입출력하는 역할을 추상화하여 모델링 한다. 예를들어 블록 읽기 동작시에는 SCSI 요청 메시지를 발생하는 역할과, 처리된 응답 메시지를 받아 요청한 메시지가 적절히 처리되었는지 확인하는 역할을 담당한다. 이 역할들

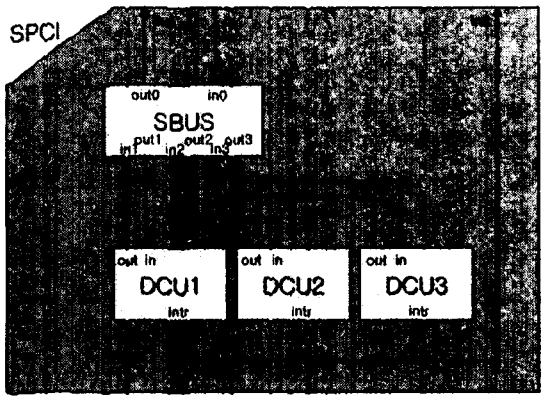
은 기능상 독립적이므로 HIU 모델은 두 개의 모델 HIU_GENR와 HIU_TRANSD로 분리하여 모델링 한다. 또한 PBUS는 PCI 버스 중재기 역할을 하여 HIU, CACHE, MCU 간의 데이터 전송을 맡는다. 그리고 하나의 PCI 명령으로 송수신되는 데이터는 그 전송이 한번의 버스 중재로 완료될 수 있도록 하나의 트랜잭션이 시작되면 그 트랜잭션이 종료할때까지 다른 트랜잭션들은 스케쥴링 되지 않도록 설계한다.



<그림 7> 주-PCI 모델 구성도

3.3 SPCI

SPCI는 <그림 8>과 같이 DCU들과 SBUS로 구성한다. 본 논문에서는 세세한 디스크 입출력 패러다임 보다는 전체적인 RAID 시스템의 구조에 촛점을 맞추었으므로 디스크와 디스크 컨트롤러의 동작을 하나의 모델 DCU로 추상화하여 모델링 한다. <그림 2>의 RAID 시스템에서 DCU의 개수는 시스템의 사양에 따라 가변적으로 구성할 수 있다. 본 논문에서는 설명을 간단하게 하기 위해 3개의 DCU가 연결된 구성에 대해서 모델링 한다. 추후에 보다 실제적인 시스템의 시뮬레이션을 위해서는 DCU의 개수를 조절한 후 MCU에서 입출력 데이터를 적절히 분배할 수 있도록 디스크 매핑 알고리즘만 조정하면 된다. SBUS는 PBUS와 그 역할과 모델 구조가 동일하다.



<그림 8> 부-PCI 모델 구성도

4. PCI 트랜잭션 모델링

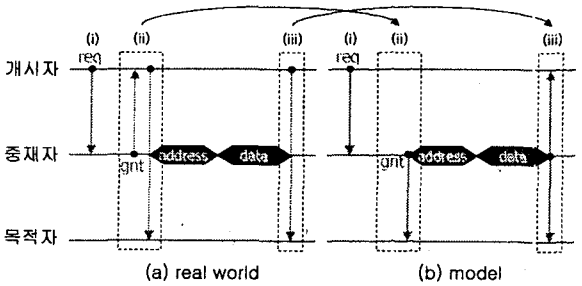
전체 시스템은 모두 19개의 모델로 구성되며, 전체 모델에 대한 기술은 너무 방대하므로 본 논문에서는 지면 관계상 그 중에서 시스템 내부의 구성 요소들을 연결하여 각종 메시지를 전달하는 역할을 하는 PCI 버스의 DEVS 모델(PBUS, SBUS)에 대해서만 살펴보겠다.

PCI 버스는 보통 데이터 버스가 32비트 혹은 64비트이고, 클럭 주파수도 33MHz와 66MHz가 표준이다[14]. 그러므로 최대 4.224Gbps의 전송속도를 가지며 숨겨진 중재(hidden arbitration)라는 독특한 버스 중재 방식을 사용하여 버스 중재에 관련한 부하가 매우 적은 특징을 갖는다. 본 논문에서는 주-PCI 버스는 동작 주파수가 33MHz이고 데이터 버스의 폭은 64비트, 부-PCI 버스는 동작 주파수가 33MHz이고 데이터 버스의 폭은 32비트로 간주하였다. 그러나 다른 임의의 동작 주파수, 데이터 버스폭에 대해서도 손쉽게 확장할 수 있다.

PCI 버스에서 버스를 통해 메시지를 보내고자 하는 장치를 개시자(initiator), 메시지를 받고자 하는 장치를 목적자(target)라고 부른다. 개시자는 PCI 버스를 통해 목적자의 주소를 지정하고, 메시지를 보내게 된다. PCI 버스 상에서, 어느 한 시점에서 버스 사이클을 제어할 수 있는 개시자는 하나 뿐이다. 그러나 하나의 버스 상에

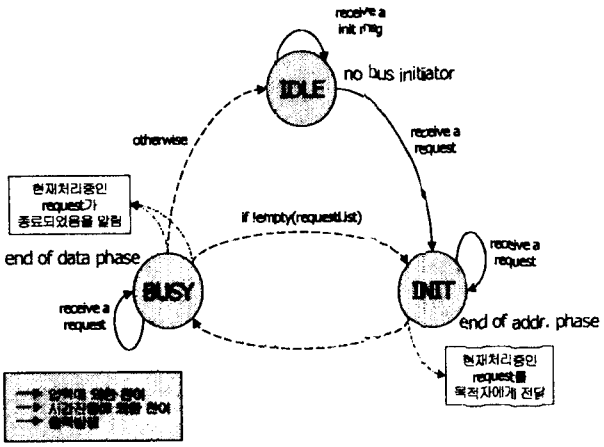
는 복수개의 개시자가 존재할 수 있으므로 그 시점마다 어떤 개시자가 버스의 사용권을 취득하여 버스 사이클을 실행하는지에 관한 조정작업이 필요하게 된다. 이와같은 조정작업을 버스 중재(arbitration)라 부르며, 이 기능을 포함하는 장치를 버스 중재기(arbiter)라 부른다. 버스 중재기는 각 개시자로부터 버스 사용의 요구(req)를 접수하고, 동시에 두개 이상의 버스 요구가 있을시 어떤 장치에 버스 사용권을 부여할지 결정하며, 버스 사용이 허용되는 장치에 버스 사용 허가(gnt)를 통지한다. 이후 버스 사용 허가를 받은 개시자는 목적자에 데이터를 전송하게 된다. PCI 버스 사용 허가는 중재 알고리즘에 따라 선입선처리 알고리즘, 우선순위 알고리즘과 같은 비교적 간단한 알고리즘이 사용되고 있다. 본 모델링에서는 우선순위에 기초한 버스 중재가 일어난다고 가정하였다. PCI 버스가 다른 중재 알고리즘을 사용하고자 할 경우에는 PCI 버스 모델의 중재 알고리즘 부분만 대체하면 손쉽게 변경할 수 있다.

위와 같은 과정을 모델링하기 위해서 <그림 9>와 같이 추상화한다. PCI 버스에서 개시자가 중재기에게 버스 사용을 요구하는 신호를 보내는 것은(<그림 9(a)-(i)>) 개시자 모델이 중재기 모델에게 버스 사용 요구 메시지를 보내는 것으로 모델링 하였다(<그림 9(b)-(i)>). 버스 사용 요구 메시지에는 개시자 ID, 목적자 ID, 전송 데이터 크기 등의 내용을 포함한다. 중재기가 선택된 개시자에게 버스 사용 허가를 통지하여 버스 사용 허가를 받은 개시자가 목적자로 전송을 시작하는 과정은(<그림 9(a)-(ii)>) 중재기 모델에 대기중인 PCI 요청 메시지 중 선택된 하나의 메시지를 해당하는 목적자로 보내는 것으로 모델링 하였다(<그림 9(b)-(ii)>). 실제로는 중재기의 버스 사용 허가를 받은 후에 개시자가 목적자에게 메시지를 전달하기 시작하나, 중재기 모델에서는 개시자 모델이 버스 사용 요구 메시지를 중재기 모델에게 전달할 때 이미 필요한 정보가 충분히 전달되었기에 이 과정을 추상화하여 중재기 모델이 개시자로부터 전달받은 메시지를 목적자로 전달한다. 개시자가 데이터 전송을 마치는 과정은(<그림 9(a)-(iii)>), 중재기에 의해서 트랜잭션을 종료하도록 하였다. 중재기 모델은 버스 요구 메시지에 의해 데이터의 크기를 알고 있고 자신의 파라미터로 버스 폭, 버스 클록 등을 가지므로 전송 시간을 계산할 수 있어, 이 시간만큼 지연된 후 전달 종료 메시지를 개시자와 목적자로 각각 보내는 것으로 모델링 하였다(<그림 9(b)-(iii)>).



<그림 9> PCI 트랙잭션의 개념도

PCI 버스의 상세 모델은 다음과 같다. 하나의 PCI 버스에는 7개의 장치가 연결될 수 있다고 가정하여 구성하였다. 그러므로 각각의 개시자로부터 PCI 명령을 받아들이는 입력포트와 입력된 명령을 해당 목적자로 전달하는데 사용되는 출력포트는 각각 7개씩 가지며, in0~in6, out0~out6로 표시한다. <그림 10>은 PCI 버스 모델의 상태 천이도를 나타낸다. 보통의 경우 PCI 버스는 IDLE 상태이다. <그림 9(b)-(i)>에 나타낸 것처럼 PCI 버스 요청이 들어오면 INIT 상태로 천이하며 버스 중재를 수행한다. <그림 9(b)-(ii)>의 버스 허가시 BUSY 상태로 천이하며, BUSY 상태가 끝나면 지연된 버스 요청을 확인하여 요청이 있으면 다시 INIT 상태로, 요청이 없으면 IDLE 상태로 천이 한다. INIT 상태가 끝날 때(BUSY 상태가 시작될 때) 버스 요청 내용이 목적자에 전달되고, <그림 9(b)-(iii)>에 나타낸 각 장치에 전송이 마침을 알리기 위한 과정은 BUSY 상태가 끝날 때 접속종료를 개시자와 목적자에 전달함으로써 이루어진다. BUSY 상태의 지속 시간은 초기화 때에 개시자와 목적자의 버스폭, 데이터 전송량에 의해서 결정된다.



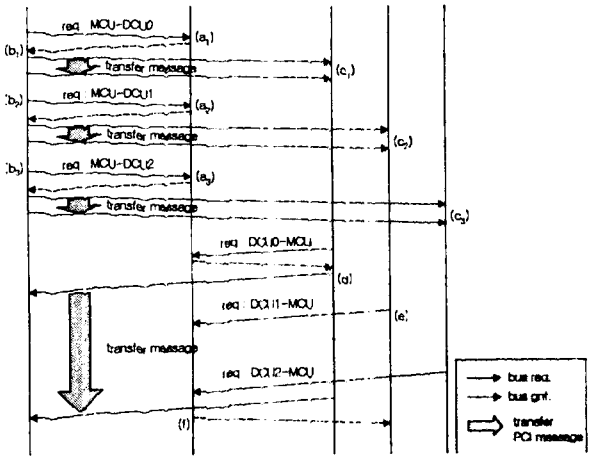
<그림 10> PCI 버스 모델의 상태 천이도

5. 시뮬레이션 결과

시뮬레이션을 위해서는 시뮬레이터의 검증(verification)과 모델의 검증(validation) 과정이 모두 이루어져야 한다. 모델의 검증을 위해서는 정확한 파라미터를 가지고 시뮬레이션하고 이것을 실세계(read-world)와 비교, 분석하여야 한다. 본 논문에서는 모델의 검증은 추후과제로 남겨두고 시뮬레이터의 검증만을 위해 <표 3~4>와 같이 임의로 파라미터를 설정하여 시뮬레이션한 결과를 살펴본다.

시뮬레이션을 위해 앞 절에서 모델링 한 RAID 시스템의 전체 모델들을 DEVSim++로 기술하였다. 전체 모델의 구성은 <그림 3>, <그림 6~8>에 나타낸 바와 같이 원소형 모델 13개, 결합형 모델 6개로 모두 19개의 모델로 구성된다. 디스크 읽기, 디스크 쓰기, 캐쉬 히트 여부 등 다양한 입출 패턴에 대한 실험을 하였으나 지면 관계상 본 논문에서는 <그림 4>의 예제에서와 같이 캐쉬 히트율 0% 일 때 0번부터 5개 데이터 블록을 읽어들이는 과정을 시뮬레이션 한 결과를 나타낸다.

시뮬레이션에서 발생한 사건들의 궤적을 분석하기에 앞서 실제 시스템에서의 처리되는 일련의 동작들을 살펴보면 <그림 11>과 같다. MCU에서 DCU로 IDE-READ 요구를 하는 과정은 MCU에서 PCI 버스에 버스 요청을 하고(a_i), PCI 버스 중재기는 요청에 대한 버스 허가를 MCU로 보내고(b_i), 버스 사용 허가를 받은 MCU는 해당 목적자로 메시지를 전송(c_i)하게 된다. 이후 이 메시지를 받은 DCU들은 해당 블록의 데이터를 하드디스크에서 읽어들여 MCU로 보낸다. MCU로부터의 요청 메시지를 받은 DCU가 하드디스크로부터 읽은 데이터 블록을 전송할 때 소요되는 시간에 관련된 파라미터는 <표 2>와 같이 설정하였다. 그러므로 IDE-READ 요구를 받은 DCU는 최소한 t_{SR} + t_{DA} 후에 읽은 블록 데이터를 전송할 준비가 갖추어진다. 그러므로 <그림 11>의 (a_1)과 (d) 사이에는 다른 요청들이 끼어들 수 있다.



<그림 11> 디스크 읽기 트랜잭션 패러다임

<표 2> 시뮬레이션 파라미터 - 디스크 읽기 동작 관련

파라미터		설정값
t_{SR}	read setup time	10
BW_{IDE}	IDE bandwidth	16
SZ_{block}	block size	1024 byte
t_{DA}	disk access time	1024 / 16 = 64

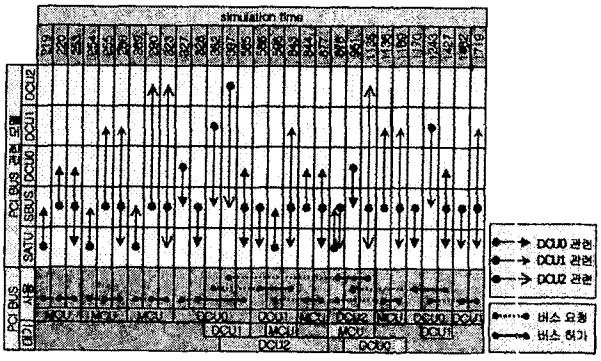
또한 MCU가 요청 메시지를 보내는 경우와 DCU가 데이터 블록을 전송하는 경우에 소요되는 시간에 관련된 파라미터는 <표 3>에 나타내었다. PCI 버스를 통해 메시지를 전달하는 시간은 SZ_{msg} / W_{bus} 로 결정되는데, IDE-READ 요청 메시지와 디스크로부터 읽혀진 데이터 블록은 메시지 크기의 차이로 인해 요청 메시지를 전달하는 것에 비해 데이터 블록을 전달하는 시간이 상대적으로 길다. 그러므로 DCU1에서 읽혀진 데이터를 MCU로 보내는 도중(d), 다른 DCU가 PCI 버스 사용허가를 요청(e)할 수 있다. 그러나 현재 PCI 버스는 DCU1에 의해 사용되어지고 있으므로, DCU1이 전송을 마친 (f)에 이르러서야 버스 중재기에 의해 DCU2로 버스 허가가 되어 메시지가 전달된다.

<표 3> 시뮬레이션 파라미터 - PCI 메시지 전달 관련

파라미터 \ 메시지 종류		IDE-READ 요청 전달시	데이터 블록 전달시
W_{bus}	bus width	32 bit	
SZ_{msg}	message size	128 byte	1024 byte
t_{AddrP}	address phase time	1	
t_{DataP}	data phase time	128/(32/8) = 32	1024/(32/8) = 256

<그림 12>는 <그림 5>의 MPG에서 나타낸 메시지들에 대한 시뮬레이션 결과를 나타낸 것이다. 우선 하나의 PCI 버스 트랜잭션이 시뮬레이션 되는 과정을 살펴보면 다음과 같다. <그림 9 (i)>와 <그람 11(a_i)>에서 나타낸 것처럼 개시자는 PCI 버스로 버스 사용 허가를 요청하고 (t=219), <그림 9(ii)>와 <그림 11(b_i~c_i)>과 같이 버스 중재기는 해당 개시자의 메시지를 목적자로 전달하기 시작함으로서 버스 사용 허가가 이루어진다(t=220). 이 트랜잭션은 <그림 9 (b)~(iii)>과 같이 PCI 버스 모델이 개시자와 목적자로 버스 사용 종료 메시지를 보냄으로서 버스 사용을 끝마치는 것을(t=253) 보여준다.

다음으로 여러 PCI 버스 요구가 있을 때 중재기에 의해 버스 중재가 일어나는 경우가 시뮬레이션 되는 과정을 살펴보면 다음과 같다. <그림 11(d)>와 같이 DCU0이 읽은 데이터를 MCU로 보내기 위해 PCI 버스 사용 허가를 받아 데이터를 전송하는 동안(t=328~585) <그림 11(e)>와 같이 DCU1과 DCU2도 데이터를 MCU로 보내려고 버스 사용 허가를 요청한다(t=362, 397). 하지만 이미 DCU0이 PCI 버스를 점유하고 있기에 <그림 11(f)>처럼 DCU0으로부터의 전송이 끝난 t=585까지 대기하였다가 t=586에서 DCU1이 버스 사용 허가를 받아 버스를 사용하는 것을 나타내고 있다. 또한 t=843에서는 버스 중재가 일어나 대기중인 MCU와 DCU2 중 우선순위가 더 높은 MCU로 버스 사용을 허가하는 것(t=844)을 알 수 있다.



<그림 12> 시뮬레이션 결과(SPCI부)

6. 결 론

본 논문에서는 IDE 디스크들을 이용하고, FC 인터페이스를 갖는 RAID 시스템을 DEVS 형식론으로 기술하였다. 특히 전체 시스템을 제어하는 MCU는 내부 구조와 제어 알고리즘을 통합하여 추상화하였다. 또한 각 구성요소간의 메시지를 주고받는데 사용되는 PCI 버스 트랜잭션의 제어 알고리즘을 분석하고 추상화하여 기술하였

다. 그리고 시뮬레이션 과정에서 발생된 사건들 중 SPCI부의 PCI 관련 메시지들을 분석하여 하나의 요구뿐 아니라 다수의 요구가 발생할 때에도 PCI 트랜잭션이 적절히 수행됨을 보았다.

본 논문에서 만들어진 모델을 이용하면 RAID 시스템의 여러 구성을 손쉽게 모델링하여 시뮬레이션 해 볼 수 있다. 즉, 시뮬레이션을 위해 5장에서 임의로 설정한 각종 파라미터(<표 2~3>)들을 조정함으로써 성능평가를 해보거나, 시스템 구조 변경으로 인한 성능 측정을 통해 좀 더 효율적인 시스템을 구축하는데에 본 논문에서 제안한 모델이 활용될 수 있다.

참고문헌

[1] Fibre Channel Standards, http://t11.org

[2] *William Stallings, Computer Organization and Architecture*, 5th ED., Prentice Hall, 2000.

[3] G. R. Ganger, B. L. Worthington, R. Y. Hou, and Y. N. Patt, "Disk arrays," IEEE Computer, pp. 30-36, March 1994.

[4] Shenze Chen and Don Towsley, "A Performance Evaluation of RAID architecture," IEEE Trans. Computers, Vol. 45, No. 10, 1996.

[5] Arif Merchant and Philip S. Yu, "Analytic Modeling of Clustered RAID with Mapping Based in Nearly Random Permutation," IEEE Trans. Computers, VOl. 45, No. 3. 1996.

[6] A. L. Narasimha Reddy and Prithviraj Banerjee, "An Evaluation of Multiple-Disk I/O Systems," IEEE Trans. Computers, Vol 38, No. 12, 1989.

[7] 이민영, 박명순, "재건 과정중의 예비 디스크 기법의 성능 및 신뢰성 평가", 정보과학회 논문지(A), Vol. 23, No.8, pp. 858-868, 1996.

[8] 백승훈, 김경호, 박규호, "병렬 디스크 시스템의 모델링 및 모의 실험," 정보과학회 '98 가을 학술발표논문집(III), Vol. 25, No. 2, pp. 33-35, 1998.

[9] 안명수, 박성봉, 김탁곤, "DEVSim++: 의미론에 기반한 이산사건 시스템의 객체지향 모델링 및 시뮬레이션 환경," 한국정보과학회 논문지, 제21권, 제9호, pp. 1652-1664, 1994.

[10] Bernard P. Zeigler, "Object-Oriented Simulation with Hierarchical, Modular Models," Academic Press, 1990.

[11] David A. Patterson, Garth Gibson, and Randy H. Katz, "A Case for redundant arrays of inexpensive disks(RAID)", ACM SIGMOD, June 1988.

[12] http://www.raid.co.kr/what/what.htm

[13] i960® RM/RN I/O Processor Developer's Manual, Intel Corporation, 1998

[14] Tom Shanley and Don Anderson, *PCI System Architecture*, Addison Wesley, 1995.

저자소개

이찬수(e-mail : cusco@zeus.kookmin.ac.kr)
1995년 국민대학교 전자공학과(공학사)
1997년 국민대학교 전자공학과(공학석사)
1997년~현재 국민대학교 전자공학과 박사과정
관심 분야 : 멀티미디어, 파일시스템, 시뮬레이션

성영락(e-mail : yeong@kookmin.ac.kr)
1989년 한양대학교 전자공학과(공학사)
1991년 한국과학기술원 전기 및 전자공학과(공학석사)
1995년 한국과학기술원 전기 및 전자공학과(공학박사)
1995년~1996년 한국과학기술원 위촉연구원
1996년~1998년 국민대학교 전임강사
1998년~2002 국민대학교 조교수
2002년~현재 국민대학교 부교수
관심 분야 : 멀티미디어 시스템, 시스템 모델링 및 시뮬레이션, 병렬처리

오하령(e-mail : hroh@kookmin.ac.kr)
1983년 서울대학교 전기공학과(공학사)
1983년~1986년 삼성전자 종합연구소
1988년 한국과학기술원 전기전자과 컴퓨터공학전공(공학석사)
1992년 한국과학기술원 전기전자과 컴퓨터공학전공(공학박사)
1992년~1996년 국민대학교 공과대학 전자공학부 조교수
1996년~2001 국민대학교 공과대학 전자공학부 부교수
2001년~현재 국민대학교 공과대학 전자공학부 교수
관심 분야 : 컴퓨터구조, 운영체제, 분산 및 병렬처리, 멀티미디어