PCI-버스 기반 RAID 시스템의 버스 성능 분석
(Performance Analysis of a PCI-Bus based RAID System)

이 찬 수 † 성 영 락 †† 오 하 령 †††
(Chan-Su Lee) (Yeong-Rak Seong) (Ha-Ryoung Oh)

요 약 단일 PCI 버스 세그먼트 내에는 제한된 수의 디스크만이 연결될 수 있으므로 대규모의 RAID 시스템은 여러 PCI 버스 세그먼트로 구성된다. 본 논문에서는 트랜잭션의 주체와 대상에 따라 RAID 시스템 내의 PCI 버스 트랜잭션들을 분류하고 각 트랜잭션의 데이타 전송 시간을 분석한다. 또한 분석 결과를 이용하여 두 개의 RAID 시스템 구성안에 대해서 읽기 및 쓰기 성능을 분석한다. DEVS 형식론을 이용한 RAID 시스템의 시뮬레이션을 통하여 여러 시스템 파라미터들의 변화에 따른 두 구성안의 성능을 평가하고 분석 결과와 비교한다.

키워드 : RAID, 데이타저장시스템, 시스템 성능분석, DEVS 형식론, 시뮬레이션

Abstract A large RAID system may consist of several PCI bus segments since a PCI bus segment can connect only a limited number of disks. In this paper, PCI bus transactions in a RAID system are classified in terms of the initiator and the target of the transaction. Also, the data transfer time of each transaction type is analyzed. By using the analysis results, read and write performance of two RAID system configurations are formulated. From simulation of the RAID system using the DEVS formalism, performance of the configurations are evaluated and compared with the analytical results while changing various system parameters.

Key words : RAID, Data Storage System, System Performance Analysis, DEVS Formalism, Simulation

1. 서 론

최근 들어 인터넷 보급이 확산되면서 기업 데이타는 물론 개인들의 각종 데이타가 급격히 증가하고 있다. 뿐만 아니라 멀티미디어 데이타의 크기가 커지고 시스템에서 사용하는 파일들이 늘어남에 따라 대량의 데이타를 저장하는 것이 필요하다. 또한 결함허용(fault-tolerant) 시스템이나 백업 시스템 등의 수요도 점차 증가하고 있다. 이처럼 많은 사용자들의 다양한 데이타 입출력 요구를 효과적으로 처리하기 위해서는 기존의 단일 대형 시스템보다 비용 대비 성능면에서 좋은 효율을 얻을 수 있는 고속, 고신뢰성의 데이타 저장 시스템이 요구된다. 그러므로 기존의 단일 시스템에 비해 시스템 구축비용, 시스템 유용성, 확장성, 고장 감내의 측면에서 장점을 가질 수 있는 네트워크 저장장치에 대한 연구가 활발히 진행되고 있다[1][2].

SAN(Storage Area Netowork)은 이러한 요구사항에 맞게 개발된 시스템으로서 RAID(Redundant Array of Independent Disks) 시스템[3]을 기본 저장 장치로 하고, 호스트 시스템과 저장 장치를 기존의 네트워크 외에 별도로 고속의 데이타 전용 네트워크인 고속의 Fibre Channel[4][5]로 연결한 시스템이다. Fibre Channel은 1Gbps 혹은 2Gbps의 고속의 전송 속도를 제공하며, 상위 프로트콜로서 SCSI, IP, HIPPI 등 다양한 기존의 입출력 프로토콜을 지원한다. 그러므로 기존의 컴퓨터 시스템에 손쉽게 연결되며, 연결망의 형태도 다양하여 확장성이 매우 우수하다. 현재 많은 웹사이트들이나 기업의 서버들은 SAN을 이용한 시스템을 구축하여 운영중이다.

RAID 시스템에서의 처리속도와 용량을 높이기 위해서는 RAID 시스템 내부에 더 많은 개수의 디스크를 장

† 학생회원 : 국민대학교 전자공학과
chansu@pccpj.org
†† 종신회원 : 국민대학교 전자공학부 교수
yeong@kookmin.ac.kr
††† 비 회 원 : 국민대학교 전자공학부 교수
hroh@kookmin.ac.kr
논문접수 : 2002년 10월 21일
심사완료 : 2003년 3월 21일

착하여야 하며, FC와 같은 인터페이스를 이용하여 호스트 시스템과 빠르게 입출력할 수 있어야 한다. 이 때 시스템 내부에서 사용하는 디스크의 개수가 증가함에 따라 RAID 컨트롤러와 디스크를 연결하는 버스에 부하가 집중된다. 또한 시스템의 각 구성요소들을 연결하는 버스로 일반적으로 많이 사용하는 PCI 버스를 사용한다고 가정했을 때, PCI 버스는 하나의 버스에 연결되는 장치의 수에 제한이 있기 때문에 다수의 디스크를 시스템에 연결하기 위해서는 확장 PCI 버스의 사용이 필요하다. 여기서 확장 PCI 버스를 사용하여 시스템을 구성하는 방법은 몇 가지 다른 구성안을 생각해 볼 수 있다. 따라서 PCI 버스 부하의 관점에서 더 우수한 구성 방법에 대한 연구가 필요하다.

본 논문에서는 PCI 버스 부하의 관점에서 고속 대용량 저장장치의 기본이 되는 RAID 시스템의 성능을 분석한다. RAID는 디스크 어레이(Disk Array)에 부가적인 데이타를 저장하기 위한 디스크들을 추가하여 신뢰성을 강화한 것이다. 그동안 많은 연구를 통해서 RAID는 수준 0에서 수준 6까지 다양한 방식이 존재한다[3]. 본 논문에서는 RAID 수준 5에 중점을 두고 있다. 이 방식은 현재 가장 널리 사용되는 RAID 방식중의 하나로서 특히 여러 사용자들의 입출력 요구를 처리하는 데에 적합한 방식이다.

그 동안의 많은 연구에서 RAID 시스템의 성능이 분석되었다. Chen과 Towsley[6]는 RAID 수준 1과 수준 5에 대해서 스케쥴링 알고리즘 및 매핑 알고리즘을 개발하고 성능을 비교하였다. Merchant와 Yu[7]는 매핑 알고리즘을 개발하고 해석적인 모델을 만들어서 성능을 분석하였다. Reddy와 Banerjee[8]는 다양한 부하에서 디스크 구성에 따른 성능을 분석하였다. 국내에서도 수년 전부터 RAID 시스템에 대한 연구가 활발히 진행되었다. 그 중 이민영과 박명순은 [9] 연구에서는 RAID 시스템의 재건 성능 분석하였다. Yokota[10]의 연구에서는 RAID 수준 5에서 버스에 의해 대표되는 중앙 통신로를 제거함으로써 시스템의 범위성(scalability)을 향상시킬 수 있는 제공하는 구조를 제안하였다. 또한 RAID 시스템의 성능을 분석하기 위해서 PCI 버스를 사용하는 RAID 시스템의 DEVS 모델[11]을 기술한 연구와, RAID 시스템의 읽기 성능에 대한 연구[12]가 있다. Compaq 등에서는 RAID 시스템의 성능을 향상시키기 위해 내부의 단위 디스크들을 Ultra SCSI나 FC로 연결하여 고가의 시스템[13]을 구성하여 상용 RAID 시스템을 제작, 판매하고 있다. 본 논문에서는 IDE 디스크와 PCI 버스를 이용하여 저가의 RAID 시스템을 구성하는 경우, 호스트의 소프트웨어가 최대성능일 때 RAID 시스템의 PCI 부하의 관점에서 구성에 따른 최대 성능비를 비교해 보고자 한다. 이를 위해 [12]의 연구를 확장하여 디스크 쓰기에 대한 연구와, 다양한 입력 조건에서의 디스크 읽기와 쓰기에 대한 시뮬레이션 결과를 분석하였다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문에서 분석할 RAID 시스템의 하드웨어 모델과 각 구성 요소들의 역할을 살펴본다. 3장에서는 대상 시스템에서 주요 구성 요소들간의 최대 데이타 전송 속도를 분석한다. 4장에서는 두 가지 시스템 구성 예에 대해서 3장의 결과를 가지고 디스크 읽기, 쓰기 성능을 비교한다. 5장은 다양한 입력 조건 하에서 두 구성안에 대한 시뮬레이션 결과를 분석하였으며, 마지막으로 6장은 결론이다.

2. RAID 시스템의 하드웨어 모델

과거 수 십 년간 프로세서의 속도는 눈부시게 발전해 왔다. 그러나 거기에 비해서 디스크는 용량의 집적도면에서는 어느 정도의 발전이 있었으나 처리속도는 많은 변화가 없었다. 이런 이유로 시간이 흘러감에 따라 프로세서 속도와 디스크 입출력 속도의 간격이 더욱 커지는 양상을 보이고 있다. 결국 대부분의 컴퓨터 시스템에서 디스크 입출력 시스템은 시스템의 성능에 있어서 최대의 병목으로 인식되고 있다. 이를 해결하는 방법으로 디스크 어레이가 연구되었다. 디스크 어레이에서는 여러 대의 디스크들을 묶어서 하나의 논리적인 디스크로 만들고 디스크 입출력을 병렬처리하는 방법으로 디스크 입출력 성능을 향상시킨다. RAID는 일반적인 디스크 어레이에 추가로 디스크들을 장착하여 시스템의 신뢰성을 향상시킨 것이다[3].

본 절에서는 본 논문에서 성능 분석의 대상이 되는 RAID 시스템의 하드웨어 모델을 제시한다. 현재까지 매우 다양한 종류의 RAID 시스템이 개발되었으며 하드웨어 구조 또한 다양하다. 본 논문에서 성능 분석의 대상이 되는 RAID 시스템의 모델을 그림 1에 나타내었다. 시스템은 SCSI over FC를 사용하여 호스트 시스템과의 통신을 담당하는 HIU(Host Interface Unit), 디스크와 인터페이스 하는 DCU(Disk Control Unit), 디스크 데이타에 대한 임시 저장 장소인 캐쉬, 시스템 전체를 제어하고 패리티 연산을 수행하는 MCU, 그리고 이들 구성 요소들을 연결하는 3개의 PCI 버스로 구성된다.

HIU는 Fibre Channel과 PCI 버스의 두 가지 인터페이스를 이용하여 고속으로 호스트 시스템과 RAID 시스템을 연결하는 역할을 한다[4][5]. DCU는 MCU가 PCI

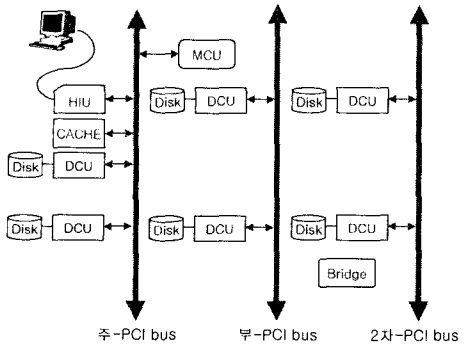


그림 1 PCI 버스를 이용한 RAID 시스템 모델

버스를 통하여 디스크 액세스 명령을 내리면 디스크를 제어하여 디스크와 MCU 사이의 데이타 전송을 중개하는 역할을 한다. 이때 디스크와의 연결은 SCSI, IDE 등의 일반적인 방식이 이용된다.

캐쉬는 RAID 시스템의 성능에 가장 큰 영향을 주는 부분 중의 하나이다. 캐쉬의 역할은 운영체제의 버퍼 캐쉬의 복사본들을 저장하여, 디스크 속도와 호스트 인터페이스 속도의 차이를 완충시키는 역할을 한다. 이처럼 디스크 캐쉬를 사용하면 읽기 요구는 입출력 수행 시간에 대해 많은 이득을 얻을 수 있다. 그러므로 단일 디스크 장치에 비해 입출력 요구에 대한 처리과정이 상대적으로 복잡한 RAID 5의 경우 개별 디스크 내에 장착된 디스크 캐쉬, 또는 트랙 버퍼(track buffer)와는 구별되는 자체의 대용량 캐쉬를 사용한다. 따라서 그 응답 특성과 데이타 신뢰도가 다른 수준의 RAID에 비하여 가격 대비 성능이 우수하기에 많은 상용 RAID 시스템들이 이를 지원하고 있다.

PCI 버스는 시스템의 구성 요소들을 연결하는 역할을 한다. 그런데 PCI 버스는 연결되는 장치들의 수에 제한이 있어 연결되어야 할 장치들의 수가 많을 때에는 PCI-to-PCI 브리지를 이용하여 버스를 확장하여야 한다. 브리지 양쪽의 버스는 서로 독립적으로 작동된다[14].

MCU는 전체 시스템을 제어하는 역할을 한다. 또 디스크 쓰기에 대한 처리에서는 RAID 수준 5를 위한 패리티를 계산한다. MCU는 두 개의 PCI 인터페이스를 가지는 것으로 간주하였다.

그림 1에서는 구성 요소들의 연결 형태만 나타낸 것이고 구체적인 내용들이 포함되어 있지 않다. 그중 하나는 각 구성 요소들과 PCI 버스와의 인터페이스이고 또 하나는 DCU의 개수이다. 본 논문에서는 HIU, 캐쉬, MCU, 브리지는 64비트 인터페이스를 가지며, DCU는 32비트 인터페이스를 가지는 것으로 가정하였다. 그러므로 DCU와 관련된 데이타는 32비트로 전송되고, 나머지는 64비트로 전송된다. 또 DCU는 총 8개로 가정하였다. 각각의 버스에 연결되는 DCU의 개수는 4장에서 구체적인 성능 분석시에 제시하겠다.

```
    unit    algorithm
(1) HIU     receive a request from host;
(2) HIU     send the request to MCU;
(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;
(9) DCU         transfer data to MCU;
(10) MCU        transfer data to cache;
(11) MCU        send the cache address to HIU;
(12) HIU        data transfer from cache;
(13)          endif
(14)        else // write request
(15) MCU      send the cache address to HIU;
(16) HIU      transfer data to cache;
(17) MCU      transfer data from cache;
(18) MCU      send disk read requests to DCUs;  // for parity calculation
(19) DCU      transfer data to MCU;
(20) MCU      transfer data to cache;
(21) MCU      parity generation;
(22) MCU      send disk write requests to DCUs;
(23) HIU      transfer response;
(24)        endif
```

그림 2 시스템제어 및 데이타흐름

그림 2는 호스트 시스템에서 그림 1의 시스템으로 디스크 명령이 전달될 경우에 각 구성 요소들간의 제어 및 데이타의 흐름을 알고리즘으로 나타낸 것이다. 그림 2에서 *unit*는 알고리즘을 수행하는 시스템 구성 요소를 나타낸 것이고, *algorithm*은 시스템 제어 및 데이타 흐름을 나타낸 것이다.

그림 2의 데이타 흐름을 살펴보면 다음과 같다. (1)HIU가 호스트로부터 요청을 받으면 (2)이것을 MCU로 보낸다. (3)MCU에서는 이 요청이 디스크 읽기에 대한 것인지 쓰기에 대한 것인지를 판단한다. 디스크 읽기 요청인 경우에 (4)이미 캐쉬에 요청 데이타가 저장되어 있다면 MCU는 (5)HIU에 데이타가 있는 캐쉬상의 주소를 보내어, (6)HIU로 하여금 캐쉬의 데이타를 가져가도록 한다. 그러나 캐쉬에 요청 데이타가 없는 경우에 (8)MCU는 해당하는 각 DCU에게 디스크 읽기 요청을 전달하고, (9)각 DCU는 디스크로부터 읽은 데이타를 MCU에게 전달한다. (10)MCU는 각 DCU로부터 전달받은 데이타를 캐쉬에 저장하고, (11)HIU에게 캐쉬상의 주소를 보내어 (12)HIU로 하여금 데이타를 가져갈 수 있도록 한다. 반면, 호스트로부터 디스크 쓰기 요청을 받은 경우에 (15)MCU는 입력된 데이타가 저장될 캐쉬의 주소를 HIU에게 보내고, (16)HIU는 이 주소에 데이타를 적는다. 그러면 (17)MCU는 캐쉬로부터 데이타를 읽어들이고, (18)패리티 계산을 위해 필요한 데이타를

읽어들이기 위해 해당 DCU로 디스크 읽기 요청을 보낸다. (19)각 읽기 요청을 받은 각각의 DCU에서는 MCU로 데이타를 보내고, (20)MCU는 전달받은 데이타를 캐쉬에 저장해 놓았다가, (21)계산을 위해 필요한 모든 데이타가 모이면 패리티를 계산한 후, (22)입력된 데이타와 패리티를 저장하고, (23)HIU는 MCU로부터 응답 메시지를 전달받아 호스트로 전달한다.

3. 데이타 전송 시간 분석

본 절에서는 RAID 시스템의 각 부분들간의 데이타 전송을 분석한다. 그림 2를 바탕으로 PCI 버스를 통한 각 부분들의 데이타의 흐름을 살펴보면 다음 표 1과 같이 분류할 수 있다. 이중에서 i)은 HIU와 MCU 사이에 SCSI 작업 요청과 응답을 위한 데이타의 흐름으로서 나머지 것들에 비해 크기가 작아서 전체 성능에 거의 영향을 주지 않는다. 본 논문에서는 i)은 제외하고, ii)와 iii)을 묶어서 캐쉬를 통한 MCU와 HIU 사이의 데이타 전송, iv)는 브리지를 통하지 않은 MCU와 DCU 사이의 데이타 전송, v)와 vi)을 묶어서 브리지를 통한 MCU와 DCU 사이의 데이타 전송으로 구분하여 분석하였다. 표 2는 각 부분에서의 데이타 전송 시간을 수식으로 표현하기 위해서 사용된 시스템 파라미터들의 정의이다.

3.1 캐쉬를 통한 MCU와 HIU 사이의 데이타 전송

입출력되는 데이타들은 항상 캐쉬를 통해 이루어진다. 예를 들어 데이타 읽기의 경우에는 MCU가 캐쉬에 각 DCU로부터 읽은 데이타를 전송한 다음에 HIU가 캐쉬 데이타를 읽어가도록 해야 하며, 디스크 쓰기 경우에는

표 1 데이타 흐름 항목의 분류

	흐름의 종류	흐름의 내용	분석 항목
i	HIU와 MCU 사이	·SCSI 작업 요청/응답 메시지	(분석 대상에서 제외)
ii	HIU와 캐쉬 사이	·캐쉬 요청 메시지 ·입출력 데이타	캐쉬를 통한 MCU와 HIU 사이의 데이타 전송
iii	캐쉬와 MCU 사이	·캐쉬 요청 메시지 ·분배된 데이타	
iv	MCU와 DCU 사이	·디스크 요청 메시지 ·분배된 데이타	브리지를 통하지 않은 MCU와 DCU 사이의 데이타 전송
v	MCU와 브리지 사이	·디스크 요청 메시지 ·분배된 데이타	브리지를 통한 MCU와 DCU 사이의 데이타 전송
vi	브리지와 DCU 사이	·디스크 요청 메시지 ·분배된 데이타	

표 2 시스템 파라미터 종류

기본 파라미터	d	MCU가 DCU의 레지스터를 설정하는 데 걸리는 시간
	a'	PCI 버스를 통해 전달되는 입출력 데이타 크기
	e	바이트당 전송시간에 대한 상수
	a	$a' \times e$
	a / (버스폭/8)	PCI 버스를 통한 a' 바이트 데이타 전송 시간
	b	디스크 액세스 시간
	c	PCI 버스가 휴식 상태로 보내는 시간
	n	DCU의 개수
a' 바이트 데이타 전송시간	T_{MCU}	MCU와 캐쉬간의 데이타 전송 시간
	T_{HIU}	HIU와 캐쉬간의 데이타 전송 시간
	T_{DCU1}	브리지를 통하지 않은 MCU와 DCU 사이의 데이타 전송 시간
	T_{DCU2}	브리지를 통한 MCU와 DCU 사이의 데이타 전송 시간

HIU가 캐쉬에 데이타를 쓰면 MCU는 그 데이타를 읽어 해당하는 DCU에 전송한다. 이 때 MCU는 패리티 계산을 위해서 입력된 데이타와 관련된 DCU로부터 데이타를 전송받아 패리티를 계산하고 이 값을 저장할 DCU에 전달한다. 이 때 전달되는 데이타의 크기가 a' 바이트라고 하면, MCU와 캐쉬간에 데이타 전송 시간 T_{MCU}와 HIU와 캐쉬간의 데이타 전송시간 T_{HIU}는 a'에 바이트당 전송시간에 대한 상수 e를 곱하여 8로 나눈 값으로 표시된다. 이 때 8로 나눈 것은 버스 폭이 64비트이므로 동시에 8바이트씩 전송되기 때문이다. 또한 a'은 디스크 블록 크기에 비해 매우 큰 값으로 가정한다. a'에 e를 곱한값을 a로 표시하면 T_{MCU}와 T_{HIU}는 다음과 같이 표시된다.

$$T_{MCU}=(a'\times e)/8=a/8 \quad (1)$$

$$T_{HIU}=(a'\times e)/8=a/8 \quad (2)$$

3.2 브리지를 통하지 않은 MCU와 DCU 사이의 데이타 전송

MCU와 DCU 사이의 데이타 전송은 디스크 읽기와 디스크 쓰기 두 가지의 경우가 있다. 그러나 MCU와 HIU 사이의 경우와는 달리 디스크 읽기와 쓰기는 세부적인 동작의 순서만 다를 뿐, 기본적으로 같은 동작을 취한다.

예를 들어 디스크 읽기의 경우를 살펴보자. 디스크 읽기를 위해서는 먼저 MCU가 해당되는 DCU들의 레지스터들을 설정해야 한다. 그러면 각각의 DCU들은 설정값에 따라 디스크를 액세스하여 데이타를 읽어들인 후, MCU에게 PCI 버스를 통해 데이타를 DMA 전송한다. 데이타 전송이 끝나면 각각의 DCU들은 MCU에게 인터럽트를 걸어서 데이타의 전송이 끝났음을 알린다. 일반적으로 MCU 내부에서는 많은 디스크 연산들이 대기중일 것이므로, MCU는 다음 디스크 연산을 시작하기 위해 다시 DCU의 레지스터들을 설정할 것이다. 디스크 쓰기의 경우에는 디스크 액세스와 PCI를 통한 DMA 전송의 순서만 바뀔 것이다.

이 과정이 여러 DCU에서 병행해서 발생하는 경우를 그림 3에 나타내었다. 초기에는 DCU에 아무런 작업 요청이 없을 것이고, PCI 버스도 쉬는 상태로 가정한다. 그러다가 t_1에서 MCU가 DCU1의 레지스터를 설정하여 디스크 입출력을 시작한다. 그 후에는 계속해서 다음 DCU들을 설정한다. 각 DCU의 레지스터들을 설정하는 데에 걸리는 시간은 d라고 하고, 액세스되는 DCU의 수가 n개라면, 각 DCU의 첫 번째 레지스터 설정이 끝나는 시각 t_2는 다음의 식 (3)과 같이 정해진다.

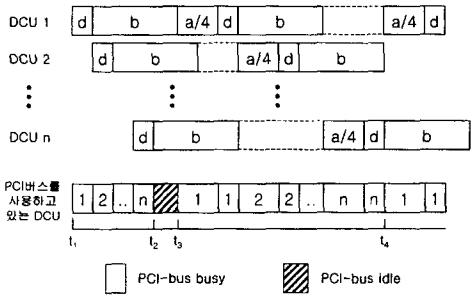


그림 3 브리지를 통하지 않은 MCU와 DCU의 데이타 전송

$$t_2=t_1+n\times d \quad (3)$$

DCU는 레지스터가 설정되면, 설정된 값에 따라 디스크를 액세스하게 된다. 레지스터 설정이 끝난 직후부터 PCI 버스로 데이타 전송 준비가 끝나는 시점까지의 시간을 b라고 정의하자. 여기에는 DCU가 디스크의 장치 레지스터들을 액세스하는 시간, 디스크 헤드의 탐색(seek) 및 회전 시간이 포함된다. 실제적으로 이 값은 파일 시스템과 파일 액세스 패턴에 따라 달라지는 가변적인 값이다. 전체 시스템의 성능 분석에서 이것을 고려하는 것은 복잡도만 증가할 뿐이므로 임의의 일정한 값으로 가정한다. DCU1의 레지스터 설정이 끝나고 b시간이 지나고 나면, DCU1은 PCI 버스로 데이타를 보낼 준비가 된다. PCI 버스로 데이타 전송을 시작하는 시점을 t_3라고 하면,

$$t_3=\begin{cases} t_1+nd & \text{if } b\le(n-1)d \\ t_1+nd+c & otherwise \end{cases} \quad (4)$$

이다. 여기서 $c(=b-(n-1)d)$는 버스가 휴식상태로 보내는 시간이다. 이때 전송되는 데이타의 양이 a'라고 하면, 2.1절에서와 마찬가지 방법으로 DCU의 버스 폭이 32비트이기에 데이타 전송 시간은 $a/4$로 나타낼 수 있다. 데이타 전송이 끝나면 MCU는 다음 작업을 DCU에 요청하기 위하여 다시 d시간을 소요할 것이다. 그러므로 PCI 버스는 t_3이후에는 각 DCU의 레지스터를 설정하는 데이타와 디스크로부터 읽어들인 데이타들이 연속적으로 처리될 수 있다. 이 때 t_1~t_2을 과도상태(transition-state), t_3 이후를 정상상태(steady-state)로 정의한다. 정상상태에서 DCU 동작들의 한 주기(그림에서 [t_3, t_4] 구간) $T_{DCU,1}$은 다음과 같다.

$$T_{DCU,1}(n)=MAX[b+d+a/4, n(d+a/4)] \quad (5)$$

여기서, $b<(n-1)(d+a/4)$이면 $T_{DCU,1}$은 디스크 액세스 시간 b에 무관하게 된다. 즉, b보다 n-1개의 DCU들을

설정하고 각각이 a' 바이트의 데이타를 MCU로 전송하는 데에 걸리는 시간이 더 크다면, b가 DCU의 DMA 데이타 전송에 전혀 영향을 주지 못하게 되며 데이타 전송이 파이프라인 형태로 이루어짐을 의미한다. 실제로 멀티미디어 동영상과 같이 대규모 연속 데이타의 경우, 시스템 성능을 향상하기 위하여 RAID 시스템의 하나의 디스크 블록을 크게 설정하거나, 인접한 영역에 데이타를 저장하는 것이 일반적이므로 이 가정이 적절할 것이다. 또한, 일반적으로 $d \ll a/4$ 이므로, DMA 데이타 전송이 완전히 파이프라인 형태로 이루어 질 경우 다음과 같다.

$$T_{DCU,1}(n) \approx na/4 \quad (6)$$

3.3 브리지를 통한 MCU와 DCU 사이의 데이타 전송

그림 4는 브리지를 통한 MCU와 DCU간의 데이타 전송을 도식화한 것이다. 그림 3의 경우와는 달리 레지스터 설정시간은 $3d/2$, 데이타 전송시간은 $3a/8$이다. 이것은 MCU에서 브리지로 메시지를 보내는 시간에 브리지에서 DCU로 메시지를 보내는 시간이 더해진 값이다. 즉, DCU의 레지스터를 설정하는데 걸리는 시간은 DCU에서 브리지까지 64비트 전송하는 시간 $d/2$에 브리지에서 DCU까지 32비트 전송하는 시간 d를 더한 값이다. 마찬가지로 a' 바이트의 데이타를 DCU에서 MCU로 DMA 전송하는 데에 걸리는 시간은 DCU에서 브리지까지 전송하는 시간 $a/4$에 브리지에서 MCU까지는 64비트 전송을 하는 $a/8$를 더한 값이다.

이러한 값들을 가지고 3.2절과 같은 방법으로 분석하면, 브리지가 있는 상태에서 정상상태의 DCU 동작들의 한 주기 $T_{DCU,2}$는 다음과 같다.

$$T_{DCU,2}(n) = MAX[b + 3d/2 + 3a/8,\ n(3d/2 + 3a/8)] \quad (7)$$

이 경우, $b < (n-1)(3d/2+3a/8)$이면 $T_{DCU,2}$는 디스크 액세스 시간 b에 무관하게 된다. 일반적으로 $3d/2 \ll 3a/8$이므로, DMA 데이타 전송이 완전히 파이프라인 형태로이루어질 경우 $T_{DCU,2}$는 다음과 같다.

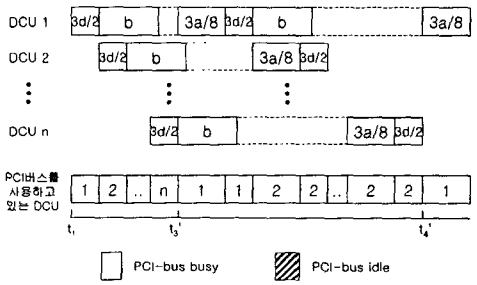


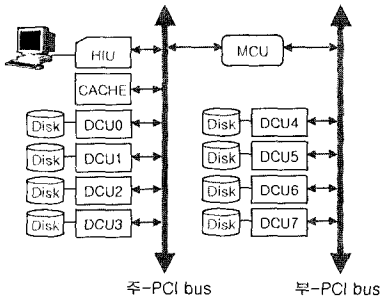
그림 4 브리지를 통한 MCU와 DCU 데이타의 전송

$$T_{DCU,2}(n) \approx 3na/8 \quad (8)$$

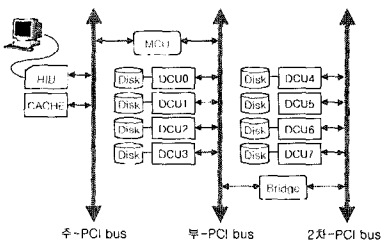
4. 성능 분석

그림 1에서 PCI 버스에 DCU를 배치하는 방법에 따라서 다양하게 시스템을 구성할 수 있다. 본 절에서는 두 가지 구성안을 제안하고, 3장의 결과를 바탕으로 성능을 비교 분석한다. 그림 5는 성능 분석에 사용된 시스템의 구성이다. 구성 1에서는 버스의 수를 최소화한 것이 특징이고, 구성 2에서는 PCI-to-PCI 브리지와 2차 PCI 버스를 이용하여 HIU, 캐쉬, MCU 사이의 작업량의 분산을 고려한 것이 특징이다.

각각의 구성에 대해 디스크 읽기와 디스크 쓰기 동작에 대한 분석 결과를 나타내었다. 구성하는 여러 구성 요소들간의 구조와 동작 과정에서 개개의 구성 요소들에 비해 매우 복잡한 상호 연관성을 지니고 있기에 일관된 수학적인 분석 모델에 의하여 정확하게 기술하는 것은 매우 어렵다. 따라서 다음의 분석에서는 디스크 읽기 캐쉬 히트와 디스크 지연 쓰기에 대한 수학적인 분석은 다루지 않는다. 즉, 읽기 캐쉬 히트율 0%, 디스크 지연 쓰기율 0%인 경우 PCI 버스 부하의 관점에서의 연속적인 블록에 대한 기본적인 디스크 입출력 성능에 대해서만 수식으로 표현한다.



(a) 구성 1



(b) 구성 2

그림 5 성능 분석 시스템의 구성

4.1 시스템 구성 1의 성능 분석

호스트에서 a' 바이트 읽기 명령이 내려진 경우, 각각의 DCU가 $a'/8$ 바이트의 데이타를 MCU에게 보내고(부-PCI, 주-PCI), MCU에서 캐쉬로 그 데이타들을 보낸 후에, HIU가 캐쉬를 액세스한다(주-PCI). 결과적으로 MCU와 캐쉬, 캐쉬와 HIU 사이에는 a'바이트가 전송된다.

주-PCI 버스와 부-PCI 버스에서 데이타를 전송하는 데에 걸리는 시간을 계산하면 다음과 같다.

$$T_{read,주-PCI,1} = T_{MCU} + T_{HIU} + T_{DCU,1}(4)/8 = a/8 + a/8 + a/8 = 3a/8 \quad (9)$$

$$T_{read,부-PCI,1} = T_{DCU,1}(4)/8 = a/8 \quad (10)$$

또한 전체 데이타 전송 소요 시간을 계산하면 다음과 같다.

$$T_{read,1} = MAX[T_{read,주-PCI,1}, T_{read,부-PCI,1}] = 3a/8 \quad (11)$$

일반적으로 RAID에서는 많은 작업들이 계속해서 요청되므로 MCU에서 디스크 입출력을 하면서, 동시에 캐쉬를 거쳐 HIU로 데이타를 전달하는 경우가 많기에 MAX 함수가 사용되었다. 식 (9)와 식 (10)으로부터 주-PCI 버스가 부-PCI 버스에 비해서 3배나 많은 시간이 소요되어 시스템의 병목이 되는 것을 알 수 있다.

호스트에서 a' 바이트 쓰기 명령이 내려진 경우에는 패리티 계산이 필요하기 때문에 DCU에 디스크 읽기와 디스크 쓰기 명령이 발생한다. HIU가 캐쉬로 a' 바이트의 데이타를 보내고, 그것이 MCU에게 전달되어 디스크에 저장되는 경우를 살펴보자. 하나의 데이타 블록만을 쓰는 경우에는 그 데이타에 대한 디스크 블록과 패리티 블록을 읽어와서 패리티 계산 후에 데이타 블록과 패리티 블록을 수정하면 디스크 입출력 명령을 줄일 수도 있다. 그러나 디스크 지연쓰기를 고려하고, 연속되는 데이타 블록을 저장하는 경우에는 패리티 계산을 위해 읽어들일 블록이 이미 캐쉬에 있기 때문에 패리티 계산을 위한 추가의 디스크 조작이 필요 없을 수 있다. 본 논문에서는 디스크 쓰기에서 패리티 계산을 위해 두 번째 제시한 방법을 사용한다. 그러나 캐쉬 히트와 디스크 지연 쓰기에 대한 고려는 생략하였기에, 호스트로부터 받은 한 블록에 대한 모든 쓰기 명령은 7번의 MCU와 캐쉬간의 데이타 전송과, 8번의 MCU와 DCU 사이의 데이타 전송(디스크 읽기 6번, 디스크 쓰기 2번)을 유발시킨다.

$$T_{write,주-PCI,1} = 7T_{MCU} + T_{HIU} + T_{DCU,1}(4)/8 \times 4 = 7a/8 + a/8 + 4a/8 = 3a/2 \quad (12)$$

$$T_{write,부-PCI,1} = T_{DCU,1}(4)/8 \times 4 = a/2 \quad (13)$$

$$T_{write,1} = MAX[T_{write,주-PCI,1}, T_{write,부-PCI,1}] = 3a/2 \quad (14)$$

디스크 쓰기의 경우 역시 주-PCI가 전체 시스템의 병목이 되고 있다.

4.2 시스템 구성 2의 성능 분석

호스트에서 a' 바이트 읽기 명령이 내려진 경우에는 DCU로 디스크 읽기 작업이 요청된다. 즉, 각각의 DCU가 $a'/8$바이트의 데이타를 MCU에게 보내고(부-PCI, 2차-PCI), MCU에서 캐쉬로 그 데이타를 보낸 후에, HIU가 캐쉬를 액세스한다(주-PCI).

주-PCI 버스와 부-PCI 버스에서 데이타를 전송하는 데에 걸리는 시간을 계산하면 다음과 같다.

$$T_{read,주-PCI,2} = T_{MCU} + T_{HIU} = a/8 + a/8 = a/4 \quad (15)$$

$$T_{read,부-PCI,2} = MAX[T_{DCU,1}(4), T_{DCU,2}(4)]/8 = MAX[a, 3a/2]/8 = 3a/16 \quad (16)$$

MCU가 브리지 이후의 DCU를 액세스하는 것과 브리지 앞의 DCU를 액세스하는 것은 완전하게 병렬로 수행될 수 있으며, 그 때 최대의 성능을 낼 수 있다. 그러므로 부-PCI 버스의 수식에서 MCU가 부-PCI 버스의 DCU를 액세스하는 시간과 2차-PCI 버스의 DCU를 액세스하는 시간 중의 큰 값이 전체 액세스 시간을 결정한다. 식 (15)와 식 (16)으로부터 전체 데이타 전송 소요시간을 계산하면 다음과 같다. 두 버스의 부하가 비슷하지만 여전히 주-PCI가 전체 시스템의 병목이 되고 있다.

$$T_{read,2} = MAX[T_{read,주-PCI,2}, T_{read,부-PCI,2}] = a/4 \quad (17)$$

4.1절에서와 같은 방법으로 분석할 경우 디스크 쓰기에 대한 소요시간은 다음과 같다.

$$T_{write,주-PCI,2} = 7T_{MCU} + T_{HIU} = a \quad (18)$$

$$T_{write,부-PCI,2} = MAX[T_{DCU,1}(4), T_{DCU,2}(4)]/8 \times 8 = MAX[a, 3a/2] = 3a/2 \quad (19)$$

$$T_{write,2} = MAX[T_{write,주-PCI,2}, T_{write,부-PCI,2}] = 3a/2 \quad (20)$$

디스크 쓰기의 경우 구성 1과는 달리 부-PCI 버스가 시스템 전체의 병목이 되는 것을 알 수 있다.

4.3 성능 비교

4.1절과 4.2절에서 산출한 식 (11), 식 (14), 식 (16), 식 (20)을 표 3에 정리하여 나타내었다. 정상상태에서 디스크 읽기의 경우 구성 2가 구성 1에 비해서 디스크 읽기의 경우에는 33% 더 우수한 성능을 보였으며 두 구성 모두 주-PCI 버스에서 병목이 발생함을 알 수 있

표 3 구성 1과 구성 2의 성능 비교

	읽기		쓰기	
	소요시간	병목	소요시간	병목
구성 1	3a/8	주-PCI	3a/2	주-PCI
구성 2	a/4	주-PCI	3a/2	부-PCI

다. 또한 정상상태에서 디스크 쓰기의 경우에는 두 구성이 같은 성능을 보였으나 디스크 읽기와는 달리 구성 1은 주-PCI 버스에서, 구성 2는 부-PCI 버스에서 병목이 발생되는 것을 알 수 있다.

5. 시뮬레이션

이산사건 시스템을 계층적이고 모듈화한 방법으로 기술하는 DEVS 형식론[15]을 이용하여 본 논문에서 성능 평가하고자 하는 두 가지 구성의 RAID 시스템을 모델링하였다. 또한 이를 이산사건 시뮬레이션 언어중의 하나인 DEVSim++[16]로 구현하였다. 이것을 위해 브리지를 포함하고 있지 않은 RAID 시스템의 기본 모델에 대한 [11] 연구에서 브리지 모델을 추가하고, 주-PCI 버스에도 DCU를 연결할 수 있게 하기 위해 수정하여 구성하였다. 본 절에서는 다양한 입력 조건에서의 디스크 읽기 및 쓰기에 대한 시뮬레이션을 하였다.

시스템 성능에 관계되는 파라미터로는 디스크 요청 수, 각 요청에서 처리되는 블록 수, 디스크 액세스 시간, 캐쉬 히트율을 고려할 수 있다. 각 실험에서 가정한 파라미터들은 시뮬레이션을 위해 실세계의 값들을 일정 비율로 줄인 값들이다. 본 논문에서는 각 요청에서 처리할 블록 수를 변경해 가면서 디스크 요청 수, 디스크 액세스 시간, 캐쉬 히트율을 변경해 가며 실험하였다. 또한 처리되는 데이타 블록이 적은 경우 입출력되는 데이타/패리티 디스크의 위치에 따라 성능 차이가 날 수 있으므로, 실험의 공정성을 위해 패리티 디스크가 1번부터 8번까지 변화하는 동안에 각기 나머지 디스크들에 대한 연속되는 n개의 블록 액세스를 실험한 후 이를 평균내어 나타내었다.

5.1 디스크 요청 수 별 디스크 읽기 시뮬레이션 결과

캐쉬 히트율이 0%일 때 하나의 요청에서 처리하는 블록 수를 바꿔가며 구성 1에 대한 구성 2의 성능향상 비를 실험한 결과를 디스크 요청 수 별로 그림 6에 나타내었다. 실험에서 사용한 주요 파라미터로 DCU 설정 메시지의 크기는 128byte, 디스크 블록 크기는 16kbyte, 디스크 액세스 시간 b는 12,000 시뮬레이션 단위 시간으로 가정하였다. 여기서 b는 3.2절에서 가정한대로 하나의 디스크 블록의 단위 시뮬레이션 전송속도인 16k/4의 3배에 조금 못미치는 값으로 임의로 설정하였다. 이 b 값에 대한 성능비교는 5.3절에서 실험하였다. 그림 6에서 보면 각각의 요청에서 같은 수의 블록을 조작하는 경우 요청 수가 늘어남에 따라 성능비가 향상되는 것을 확인 할 수 있다. 또한 같은 요청 수라도 하나의 요청에서 처리하는 블록 수가 늘어나면 성능비가 향상되게 된다. 그리고 단일 디스크 요청에서 16개 미만의 블록을 조작하는 경우에는 구성 2에서 메시지들이 정상상태에서 각 버스를 파이프라인되어 처리되는 이점보다는 브리지를 통해 전달하는 부하가 더 커서 브리지를 사용하지 않는 구성 1이 더 좋은 성능을 나타낸다. 하지만 다수의 요청의 경우에서는 각각의 요청에서 하나의 블록만을 처리하는 경우라도 연속되는 요청 수가 늘어남에 따라 30%에 근접해 가는 것을 확인 할 수 있다. 여기서, 같은 개수(요청 수×단위 요청당 처리되는 블록 수)의 블록을 연속적으로 처리하는 경우에는 한 번에 처리되는 블록 수가 많은 경우보다는 요청 수가 많은 경우에 더 높은 성능을 나타내고 있음을 확인 할 수 있다. 이것은 구성 2에서 요청이 여러 개로 나누어짐에 따라 T_{HII}가 분산되어 처리되는 효과가 전체 성능에 높은 영향을 미치는 것으로 해석할 수 있다. 연속되는 요청 수도 많고, 각 요청에서 처리하는 블록 수도 많은 경우에는 구성 2에서 PCI 버스에서의 데이타가 정상상태로 처리되기에 충분하기에 구성 1에 대한 구성 2의 성능 향상 비가 33%에 근접해 가는 것을 확인 할 수 있다. 이 결과는 4장에서 수식으로 계산한 이론적인 최대 성능비 값과 일치한다.

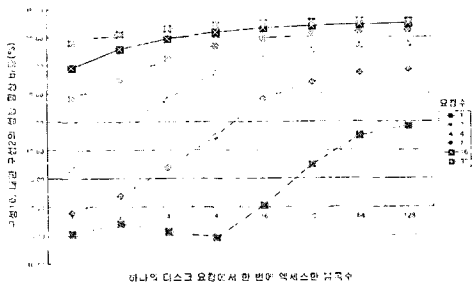


그림 6 요청 수에 따른 성능비교 - 디스크 읽기

5.2 디스크 요청 수 별 디스크 쓰기 시뮬레이션 결과

캐쉬 히트율이 0%일 때 디스크 쓰기에서 구성 1에 대한 구성 2의 성능비를 실험한 결과를 그림 7에 나타내었다. 실험에서 사용한 주요 파라미터는 5.1절의 디스크 읽기 시뮬레이션의 경우와 같게 설정하였다. 그림 7

에서 보면 공통적으로 한 요청에서 처리하는 블록 수가 적은 경우에는 디스크 읽기와 마찬가지로 요청 수가 증가함에 따라 성능비가 향상되어 나타난다. 그러나 처리 블록 수가 증가하여 정상상태가 되어감에 따라 구성 1에 대한 구성 2의 성능비가 점차 0%로 수렴하고 있는 것을 볼 수 있다. 즉, 단일 요청인 경우를 제외하고 블록 수가 16개 이하인 경우 앞 절에서 수식으로 계산한 결과와는 달리 구성 2가 구성 1에 비해 10% 이상 좋은 성능을 나타내고 있다. 이는 다수의 요청인 경우, 처리되는 블록 수가 전체 데이타 흐름에 비해 상대적으로 작기 때문에 T_{HUI}가 파이프라인되어 처리될 수 있는 구성 2의 장점이 전반적으로 좋은 성능을 가져온다 볼 수 있다. 하지만 디스크 쓰기의 경우에도 처리되는 블록 수가 커지게 되면 정상상태에서의 이론적인 최대 성능 향상비 값인 0%로 점차 수렴해 가는 것을 볼 수 있다. 다시말해 블록 수가 적은 경우에 구성 2가 좋은 성능을 나타내는 이유로는 수리적 분석에서 가정한 부분들에 대한 오차와, 구성 1은 구성 2와는 달리 주-PCI 버스가 병목이기에 상대적으로 처리할 요청 수가 많음에 따라 주-PCI 버스의 병목이 심하기에 부-PCI 버스가 병목인 구성 2가 더 나은 성능을 나타냄을 보인다.

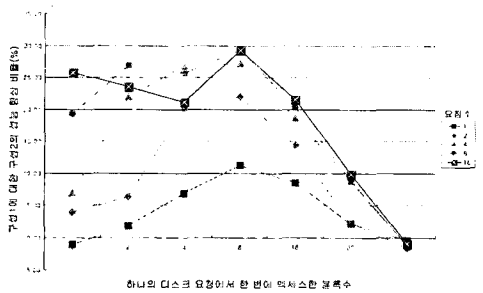


그림 7 요청 수에 따른 성능비교 - 디스크 쓰기

5.3 디스크 액세스 시간에 따른 성능 평가

5.1절과 5.2절에서 어느 정도 충분한 요청 수와 블록 수를 가져 정상상태의 입출력이 이루어지는 16개의 요청에서 디스크 액세스 시간(b)에 따른 성능차이를 아래 그림 8과 그림 9에 나타내었다. 실험에서 사용한 주요 파라미터로 DCU 설정 메시지의 크기는 128byte, 디스크 블록 크기는 16kbyte로 가정하였다. 그림 8에서 보면 디스크 액세스 시간 b가 5,000 시뮬레이션 단위 시간인 경우에는 파이프라인되어 처리되기에 b가 충분하지 못하기에 다른 경우와 상이한 성능을 보이지 못하나, 나머지의 경우에는 디스크 읽기, 쓰기의 경우 모두 앞 절의 디스크 읽기와 디스크 쓰기의 분석을 만족함을 볼 수 있다. 다시 말해 디스크 읽기의 경우 각 요청에서 조작하는 블록 수가 많아짐에 따라 점차 성능이 30%대로 향상 되어가는 것을, 디스크 쓰기의 경우에는 성능이 점차 0%로 수렴되어 가는 것을 확인할 수 있다. 그러나 각 시스템을 구성하는 단위 디스크의 b가 커질 수록 구성 1과 구성 2의 성능 차이는 줄어드는 현상을 볼 수 있다. 이것은 b값의 증가로 인해 버스 IDLE 시간이 발생하여 두 구성의 성능차가 줄어들기 때문이다.

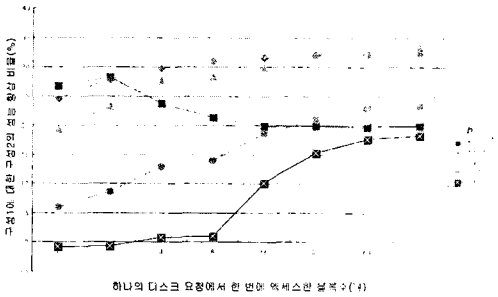


그림 8 DAT에 따른 성능비교 - 디스크 읽기

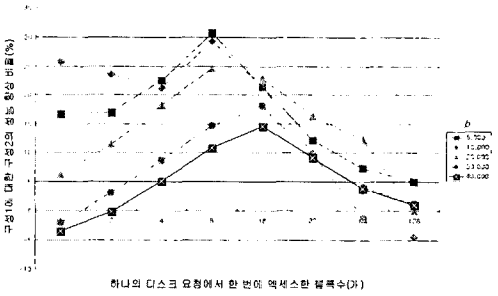


그림 9 DAT에 따른 성능비교 - 디스크 쓰기

5.4 디스크 읽기에서 캐쉬 히트율에 따른 성능 평가

5.1절에서 어느 정도 충분한 요청 수와 블록 수를 가져 정상상태의 입출력이 이루어지는 16개의 요청에서 캐쉬 히트율(CHR)에 따른 디스크 읽기의 성능 차이를 그림 10에 나타내었다. 실험에서 사용한 주요 파라미터로 DCU 설정 메시지의 크기는 128byte, 디스크 블록 크기는 16kbyte, 디스크 액세스 시간 b는 12,000 시뮬레이션 단위 시간으로 가정하였다. 그림 10에서 보면 캐쉬 히트율이 0%일 때 두 구성의 성능 차이가 제일 크게 나며 점차 캐쉬 히트율이 높아짐에 따라 차이가 줄어들고, 캐쉬 히트율이 100%인 경우에는 두 구성의 성능차이가 전혀 나지 않는 것을 확인할 수 있다. 캐쉬 히트율이 100%인 경우에는 디스크 읽기 요청에 대한 것이 두 구성 모두 디스크의 조작없이 캐쉬에 있는 데이

타를 보내게 되므로 같은 성능을 나타낸다. 또한 캐쉬 히트율 0%인 경우를 제외하고는 디스크 읽기에 대한 성능 분석에서 보았듯이 캐쉬 히트율이 바뀌더라도 하나의 요청에서 조작하는 블록의 수가 많아질수록 성능 차이가 증가한다.

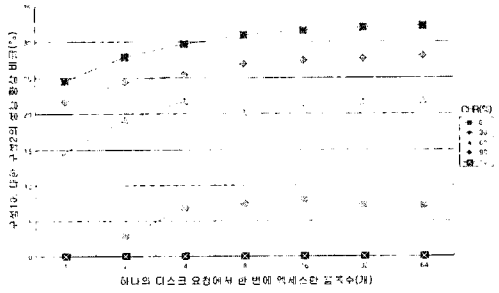


그림 10 캐쉬히트율에 따른 디스크 읽기 성능 평가

6. 결 론

본 논문에서는 하드웨어, 특히 버스 부하의 관점에서 RAID 시스템의 성능을 분석하였다. RAID 시스템의 구성 요소들간의 트래픽을 분석하여 몇 개의 범주로 나누고 그때 일정량의 데이타를 전송하는 데에 걸리는 시간을 분석하였다. 이것을 바탕으로 버스의 수를 최소화한 구성과 세 개의 버스를 이용하여 각 버스의 작업량의 분산을 고려한 구성 예에 적용하여 시스템의 읽기 및 쓰기 요청에 대한 처리 시간을 비교하였다. 또 시뮬레이션을 통해 다양한 입력 패턴이 주어졌을 때의 성능을 분석하였다. 정상상태에서의 수학적인 분석 결과와 시뮬레이션 결과로부터 같은 구성 요소들로 이루어진 시스템임에도 불구하고 읽기의 경우 세 개의 버스를 이용하여 각 버스의 작업량의 분산을 고려한 구성 2가 약 30% 정도 더 낮은 성능을 보이고, 쓰기의 경우는 두 구성 모두 비슷한 성능을 보였다. 시뮬레이션 결과 분석으로부터 디스크 읽기에서 브리지를 통한 2차-PCI 버스는 적은 수의 명령 요구에 대해서는 브리지 자체가 병목으로 작용하여 나쁜 성능을 초래할 수 있지만, 다수의 명령 요구에 대해서는 각 명령들이 서로 파이프라인되어 처리될 수 있고, PCI 버스상의 각 장치간의 작업량을 적절히 분산시킬 수 있어 전체적인 성능향상을 가져올 수 있음을 확인하였다. 또한 디스크 액세스 시간의 경우 각각의 구성에서 데이타의 전송이 정상상태로 전송될 수 있을 만한 시간만큼 충분하다면 4장에서 수식의 계산한 결과를 만족한다. 그리고 캐쉬 히트율이 높아짐에 따라 두 구성의 성능 차이가 점점 줄어드나 여전히 구성 2가 구성 1에 비해 좋은 성능을 나타내고 있을 확인할 수 있었다. 그러므로 다수의 디스크를 연결하여 대용량의 RAID 시스템을 구축하고자 할 때에는 상대적으로 많은 요청을 처리해야 하는 디스크 읽기에서 주-PCI 버스가 병목이므로 확장 PCI를 이용하여 시스템을 구성하는 것이 바람직하다. 추후과제로 좀 더 일반적인 경우의 두 구성안의 성능을 평가하기 위해 디스크 지연 쓰기를 고려한 성능 평가와, 연속적이지 않은 디스크 블록의 읽기/쓰기에 대한 성능 분석도 필요하다.

참 고 문 헌

[1] Thomas E. Anderson, et. al., "Serverless Network File Systems," Proceedings of the 15th ACM Symposium on Operating Systems Principles, pp.109-126, Copper Mountaion Resort, Colorado, December, 1995.

[2] Marc Farley, *Building Storage Networks*, McGraw Hill, 2000.

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

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

[5] Clit Jurgens, "Fibre Channel: A Connection to the Future," IEEE Computer, Vol.28, No.8, pp.82 90, August, 1995.

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

[7] 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.

[8] A. L. Narasimha Reddy and Prithviraj Banerjee, "An Evaluation of Multiple Disk I/O Systems," IEEE Trans. Computers, Vol38, No.12, 1989.

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

[10] H. Yokota. "DR nets: Data Reconstruction Net works for Highly Reliable Parallel Disk Systems. In Proc. of 2nd Workshop on I/O in Parallel Computer Systems," pp.105 116, April 1994.

[11] 이찬수, 성영락, 오하령, "RAID 시스템의 모델링 및 시뮬레이션", 시뮬레이션학회 논문지, 제11권, 제1호, 2002.

[12] 이찬수, 성영락, 오하령, "RAID 시스템의 읽기 동작에 대한 성능평가", 2002년도 춘계사료저장시스템 학술대회 논문집, 한국정보처리학회 사료저장시스템연구회, pp.166 172, 2002

[13] http: h18000.www1.hp.com storage array

systems.html
[14] Tom Shanley and Don Anderson, *PCI System Architecture*, Addison Wesley, 1995.
[15] Bernard P. Zeigler, "Object-Oriented Simulation with Hierarchical, Modular Models," Academic Press, 1990.
[16] 안명수, 박성봉, 김탁곤, "DEVSim++: 의미론에 기반한 이산사건 시스템의 객체지향 모델링 및 시뮬레이션 환경," 한국정보과학회논문지, 제21권, 제9호, pp.1652-1664, 1994.

이 찬 수
1995년 국민대학교 전자공학과(공학사) 1997년 국민대학교 전자공학과(공학석사) 2003년 국민대학교 전자공학과(공학박사) 관심분야는 멀티미디어 시스템, 시스템 모델링 및 시뮬레이션, DEVS 형식론

성 영 락
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. 관심분야는 멀티미디어 시스템, 시스템 모델링 및 시뮬레이션, 병렬처리, 내장형 시스템

오 하 령
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~2001년 국민대학교 공과대학 전자공학부 부교수. 2001년~현재 국민대학교 공과대학 전자공학부 교수. 관심분야는 컴퓨터구조, 운영체제, 분산 및 병렬처리, 멀티미디어