리눅스 멀티 패스를 위한 효율적인 A/A 패스 셀렉터의 제안

편상형[0] 석진선[1], 노재춘[1], 김경훈[2]
세종대학교 컴퓨터 공학과[01], (주)글루시스[2]
coolsmekr@yahoo.co.kr[0], durgatm@naver.com[1], jano@sejong.ac.kr[1], kgh@gluesys.com[2]

Proposal for the Linux Multipath Selector

Sanghyung Pyun[0], Jinsun Suk[1], Jaechun No[1], GyeongHun Kim[2]
Dept. of Computer Engineering, Sejong University[01], Gluesys[2]

요 약

대용량 데이터 처리를 위한 SAN(Storage Area Network)는 각 호스트 마다 두 대의 HBA(Host Bus Adapter)사용 하며, 한대는 Active로서 활동을 하고 다른 한대는 Active가 동작하지 않을 때를 대비하여 Standby로 사용된다. 현재 리눅스 커널에선 두 대의 HBA를 동시에 사용하여 효율성을 극대화하기 위한 멀티패스를 지원하고 있다. 데이터의 패스는 라운드 로빈 방식에 의해 수행되며 단순히 패스를 돌아가며 선택하는 방식을 사용한다. 라운드 로빈 방식의 패스 셀렉터는 워크로드의 양이 일정하지 못할 경우에는 좋지 않은 성능을 보인다. 본 논문은 워크로드의 양이 일정하지 않을 경우에도 동일한 성능을 낼 수 있는 패스 셀렉터를 제안한다.

1. 서 론

SAN(Storage Area Network)[1] 이나 NAS(Network Attached Storage)[2]환경에서는 호스트(서버)와 스토리지 디바이스가 파이버 채널(Fibre Channel)과 같은 인터페이스로 연결된 상태에서 데이터 입출력을 수행한다. HBA는 데이터 입출력의 증가로 발생하는 호스트의 부하를 줄이기 위해 데이터 입출력을 제어하는 역할을 수행한다.

현재 HBA를 사용하는 대부분의 제품은 각 호스트마다 한 대의 HBA를 사용하는 것이 아니라 장애극복(Failover)을 위해 두 대의 HBA를 사용한다. 두 대의 HBA가 동시에 활동하는 것이 아니라 한 대는 데이터 입출력을 수행하고 다른 한 대는 활동하는 HBA의 고장을 대비한 Active/Standby(이후 A/S로 표기)형태를 취한다.

A/S는 두 대의 HBA를 가지고 있어도 한 대만을 사용하기 때문에 한 대를 사용하는 HBA와 같은 성능을 낸다. 한 대의 성능밖에 내지 못하는 A/S를 대신하여 두 대 모두의 성능을 사용할 수 있게 나온 것이 Active/Active(이후 A/A로 표기)이다. A/A는 두 대의 HBA가 모두 데이터의 입출력을 하면서 하나의 HBA가 고장이 나면 다른 HBA가 그것이 하던 일을 이어 받아서 하게 된다.

리눅스 커널은 A/A를 지원하기 위해 디바이스 맵퍼(Device-Mapper)를 사용한다. 디바이스 맵퍼는 리눅스 커널 2.6에서 LVM을 지원하기 위한 새로운 구성요소이며 A/A 지원을 위해 멀티패스(Multipath) 데몬을 사용하고 있다.

디바이스 맵퍼의 멀티패스 데몬에서는 데이터 패스를 선택하기 위한 셀렉터로써 라운드 로빈 방식의 패스 셀렉터를 사용한다. 라운드 로빈 방식의 패스 셀렉터는 한 패스에 부하가 집중되면 좋지 않은 성능을 보인다.

2. 본 론

이 논문에서는 기존의 라운드 로빈 방식을 대신할 새로운 패스 셀렉터(이후 러그(ROUnd-robin aginG라 표기)를 제안한다. 본론에서는 디바이스 맵퍼를 설명하고 기존의 멀티패스의 디자인을 살펴본 후 러그를 설명할 것이다. 마지막으로 결론에서 라운드 로빈 방식과 러그를 비교한다.

2.1 HBA Active / Standby

[그림 1]의 좌측 그림은 HBA A/S를 나타낸다. 그림과 같이 HBA는 스토리지 디바이스와 호스트 사이에서 파이버 채널로 연결되어 있다. 한 대의 HBA는 Active로 활성화 되어 데이터 입출력을 관리하고 나머지는 Standby로써 활동하는 HBA의 고장에 대비하여 대기상태로 있다.

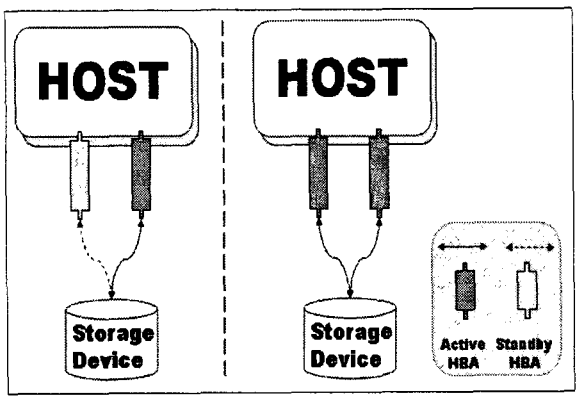
A/S 구조는 한 대의 HBA가 항상 대기를 하고 있기 때문에 높은 안정성을 제공하며, 또한 한대의 HBA만이 데이터 입출력을 관리하기 때문에 일정한 성능을 나타낸다.

2.2 HBA Active / Active (Multipath)

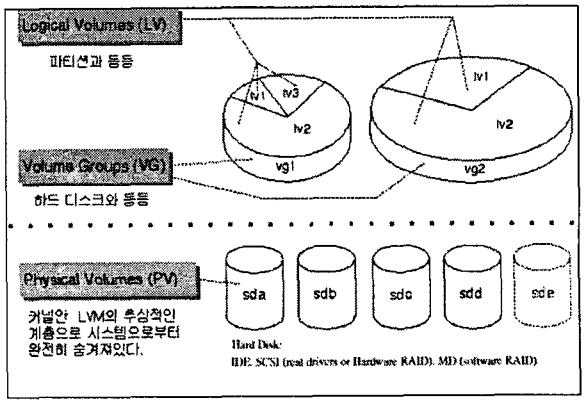
[그림 1]의 우측 그림과 같이 두 대의 HBA가 동시에 Active로 동작하는 방식이 HBA A/A이다. A/A는 두 대의 HBA를 동시에 동작시켜 HBA A/S에 비해 성능을 극대화 시켰다. 만약 두 대 중 한 대의 HBA가 동작하지 않는다면 다른 한 대가 그 몫을 대신하여 모든 입출력을 처리한다.

2.3 디바이스 맵퍼

디바이스 맵퍼[3][4]는 LVM(Logical Volume Management)[5][6]을 지원하기 위한 리눅스 커널의 새로운 구성요소이다. 커널 2.6부터 디바이스 맵퍼를 통하지 않고는 LVM을 사용 할 수 없다. LVM은 물리적으로



[그림 1] Active/Standby(좌) 와 Active/Active(우)



[그림 2] LVM의 구조

분리되어 있는 스토리지를 논리적으로 묶을 수 있고 묶은 스토리지를 다시 볼륨을 나눌 수 있게 해준다. LVM은 [그림 2]와 같은 구조로 스토리지 디바이스를 구성한다.(커널 2.6버전에서는 LVM2를 디바이스 맵퍼를 통해 지원한다.)

디바이스 맵퍼는 두 가지의 데이터 저장 방식을 지원한다. 첫 번째로 한 대의 스토리지 디바이스에 하나의 데이터 청크(Data Chunk)를 저장하는 **선형(Linear)** 방식이 있고 두 번째로 데이터 청크를 같은 크기의 블록으로 나눠서 하나 이상의 스토리지 디바이스에 저장하는 **스트라입(Stripe)** 방식이 있다.

디바이스 맵퍼가 지원하는 멀티패스 데몬은 [그림 3]과 같은 구조를 가지고 있다[3]. 그림을 보면 디바이스와 연결된 패스가 있고 각 패스는 우선순위 그룹(priority group)을 형성하고 있다. 그리고 각 우선순위 그룹은 다른 우선순위 그룹들과 연결되어 있다. 우선순위 그룹에는 데이터 입출력 요청이 있을 경우 어느 패스로 보낼지 결정하는 **패스 셀렉터(path selector)**가 있다. 멀티패스 데몬은 패스를 돌아가면서 사용하는 라운드 로빈 방식의 패스 셀렉터를 사용한다.

라운드 로빈 방식의 패스 셀렉터는 패스가 데이터 입출력에 에러를 내기 시작한다면 해당 패스를 failed로 마크하고 다음 패스로 데이터 입출력을 넘겨준다.

만약 우선순위 그룹이 고장(fail) 나면 다음 우선순위 그룹(다음 그룹이 존재한다면)을 사용한다. 우선순위 그룹 내에서 모든 패스가 고장 나면 필요한 정보를 유지하고 유효한 패스가 나타날 때 까지 기다린다.

2.4 라운드 로빈 방식의 패스 셀렉터

라운드 로빈 방식의 패스 셀렉터는 유효한 패스(valid_paths) 리스트와 유효하지 않은 패스(invalid_paths)리스트를 가지고 있다. 각 패스를 초기화 할 경우 패스를 유효한 패스 리스트에 삽입하고 패스 고장이 발생하면 해당 패스를 유효하지 않은 패스 리스트로 이동 시킨다. 패스가 복구되면 유효한 패스로 이동 시킨다.

라운드 로빈 방식 패스 셀렉터는 패스를 선택할 때 유효한 패스 리스트의 첫 패스를 선택한다. 그리고 마지막 패스를 첫 패스로 만들고 선택된 패스를 다음 패스로 연결한다.

2.5 러그 패스 셀렉터

라운드 로빈 방식의 패스 셀렉터는 항상 동일한 크기의 데이터 입출력이 일어날 때는 좋은 성능을 보인다. 하지만 일정치 않은 크기의 데이터 입출력이 일어날 경우 한 패스로 워크로드가 긴 데이터 입출력이 집중되면 **로드 밸런싱(Load Balancing)**에 문제가 일어난다. 본 논문은 라운드 로빈 방식의 패스 셀렉터에서 로드 밸런싱의 문제를 해결하기 위한 새로운 패스 셀렉터 러그를 제안한다.

[그림 4]는 두 대의 HBA를 사용한 러그의 동작의 예이며 이것은 다음의 룰을 따른다.

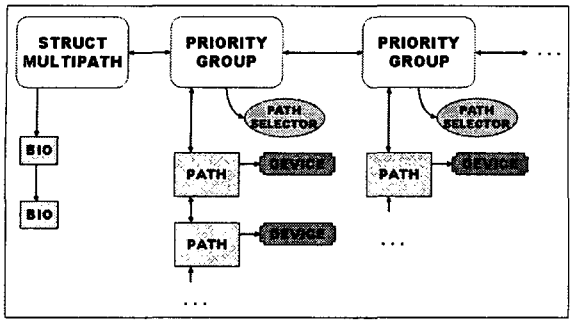
1. 데이터의 입출력(이하 이 설명에선 일이라 지칭하겠다.)이 시작되면 에이징 카운터의 값을 1로 초기화. (일이 없을 땐 0)
2. 모든 패스에 에이징 카운터 값이 4보다 작거나 4와 같다면 라운드 로빈 방식으로 패스를 선택한다. 선택된 패스의 워크큐(다음 할 일을 담고 있는 큐)에 일을 추가한다. 만약 에이징 카운터의 값이 모두 4보다 작다면 일이 추가되지 않은 패스의 에이징 카운터 값을 1씩 증가 시킨다.
3. 에이징 카운터 값이 4가 되는 패스가 하나라도 존재한다면 그 패스의 워크 큐에는 더 이상 일을 넣지 않는다. 그리고 일이 추가되지 않은 패스의 워크큐는 2와 같이 한다.

2.6 시뮬레이션

시뮬레이션은 기본적으로 1GB HBA 2대에 100KB ~ 100MB 사이의 파일 10000개의 입출력을 가진다고 상정하고 다음의 두 가지 경우를 100번씩 테스트하였다.

A의 경우 : 한 패스에 50MB이상의 데이터가 집중된 입출력
B의 경우 : 모든 패스에 동일한 크기의 데이터 입출력

[테이블 1]은 A의 경우와 B의 경우를 100번씩 시뮬레이션 한 결과값의 평균이다.



[그림 3] 멀티패스 지원을 위한 디바이스 맵퍼의 구조

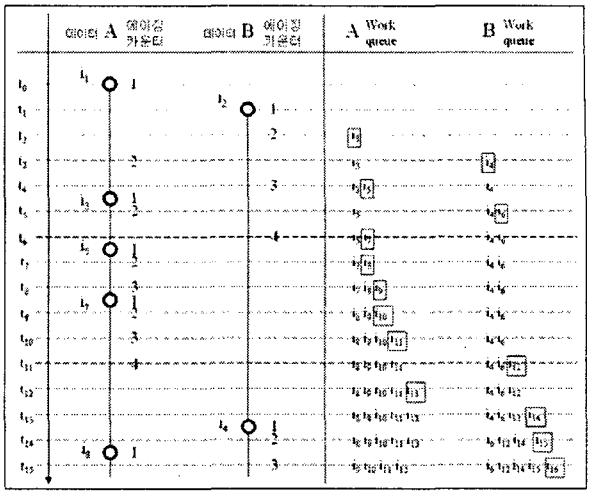
3. 결 론

리눅스는 두 개 이상의 스토리지 디바이스를 관리하기 위하여 디바이스 맵퍼를 지원한다. 그리고 디바이스 맵퍼는 두 개 이상의 HBA를 동시에 사용하기 위해 멀티패스 데몬을 사용한다. 현재 멀티패스 데몬은 패스를 선택하기 위하여 라운드 로빈 방식의 패스 셀렉터를 사용하는데 현재 라운드 로빈 방식의 패스 셀렉터는 크기가 동일한 데이터의 입출력에서는 좋은 성능을 보이지만 크기가 동일하지 않은 데이터의 입출력에 대해선 로드 밸런싱의 문제가 생긴다. 이 문제를 해결하기 위해 러그를 제안하였다. 러그는 라운드 로빈과 에이징 기법을 혼용하여 패스를 선택한다.

러그와 라운드 로빈 방식의 패스 셀렉터를 비교한 실험에서 라운드 로빈 방식과 러그는 크기가 동일한 데이터의 입출력에서는 같은 성능을 보인다. 하지만 크기가 일정하지 않은 데이터 입출력에선 러그가 라운드 로빈 방식보다 1.5배가량 빠르다. 그리고 라운드 로빈 방식은 HBA간 데이터 입출력 시간은 3배정도 차이가 나지만 러그는 차이가 거의 없으므로 라운드 로빈 방식보다 HBA 활용률도 높다. 앞의 결과와 같이 러그는 라운드 로빈 방식의 패스 셀렉터보다 우수한 성능을 가진다.

	Round-robin				RR + Aging			
	A상황		B상황		A상황		B상황	
	걸린시간	데이터 입출력수	걸린시간	데이터 입출력수	걸린시간	데이터 입출력수	걸린시간	데이터 입출력수
1st HBA	126.94 sec	5000	250.90 sec	5000	248.83 sec	5005	251.51 sec	5000
2nd HBA	375.07 sec	5000	248.55 sec	5000	248.35 sec	4995	251.15 sec	5000
평균	251.01 sec	5000	249.72 sec	5000	248.59 sec	5000	251.33 sec	5000

[테이블 1] 라운드 로빈 과 라운드 로빈 + 에이징의 패스 셀렉터를 비교



[그림 4] 두 대의 HBA를 사용한 러그의 예 (i_{xx}는 데이터를 의미하고 □안의 i_{xx}는 그 시점에 추가된 데이터를 의미한다.)

3. 참고자료

[1] Storage area network, en.wikipedia.org/wiki/Storage_area_network
[2] Network-attached storage, en.wikipedia.org/wiki/Network-attached_storage
[3] corbet, Multipath support in the device mapper, lwn.net, 2005-2-23
[4] Device-mapper Resource Page , sources.redhat .com/dm/
[5] ajl, LVM HOWTO, tldp.org, 2005-10-3
[6] Michael Hasenstein, The Logical Volume Manager (LVM), www.suse.com/en/whitepapers/lvm/, 2001
[7] Kernel source, kernel.org
[8] Jonathan Corbet, Alessandro Rubini & Greg Kroah-Hartman, Linux Device Driver, 3rd edition, O'Reilly 2005
[9] Daniel P. Bovet & Marco Cesati, Understanding the Linux Kernel, 3rd edition, O'Reilly 2005