NCQ와 입출력 스케줄러의 기대 불일치로 인한 입출력 요청의 굶주림 현상 제거

(Eliminating Request Starvation due to Expectation Discrepancy between NCQ and I/O Scheduler)

유 영 진 † 신 동 인 †
(Young Jin Yu) (Dong In Shin)
정 임 영 †† 염 헌 영 †††
(Im Young Jung) (Heon Young Yeom)

요 약 Native Command Queueing(이하 NCQ)는 디스크 드라이브 내의 명령어 큐에 존재하는 요청들의 순서를 재조정하여 throughput을 최대화하는 기술이다. NCQ는 최신 S-ATA 2의 표준 스펙에 포함되었고, 다수의 디스크 벤더들이 자사의 디스크 모델에 이를 구현하고 있다. 하지만 이 새로운 기술이 운영체제와 디스크 드라이브 간의 정보 차이를 유발할 가능성이 있다. 운영체제는 자신이 지시한 순서대로 디스크가 입출력 요청을 서비스할 것이라 생각하지만, NCQ가 지원되는 디스크는 이를 무시하고 throughput을 최대화할 목적으로만 요청을 처리할 것이다. 이것을 기대 불일치라 부를 수 있다. 이로 인해 성능에 이상한 현상이 발생하거나, 입출력 요청이 심각하게 굶주릴 가능성이 있다. 본 논문에서는 기대 불일치로 인한 입출력 요청의 굶주림 현상을 실제로 확인하고, 이를 해결하기 위한 해결책을 제시한다. 이 해결책은 간단하고, 특별한 하드웨어의 추가나 변경을 요구하지 않으며, 이식성이 좋다. 이를 실험 결과를 통해 확인하도록 한다.

키워드 : 입출력 스케줄러, NCQ, 기대 불일치, 요청 굶주림

Abstract Native Command Queueing is a technology to maximize throughput of disk system by reordering requests in its command queue. Recent S-ATA 2 standard specifies a protocol for the purpose of dealing with NCQ feature, making most vendors implementing it in their disk devices. However, the new feature, NCQ, may lead to information gap between OS and disk drive. OS considers that disk will service I/O requests in the order as OS has dispatched. Unfortunately, it isn't true any more since NCQ would simply ignore the policy of OS and reorder the sequence OS has expected. Let us define the term as expectation-discord. Due to the discord, serious performance anomalies or starvation of requests may occur in I/O subsystem. In this paper, we confirm that the expectation-discord actually brings about starvation of requests. We propose a solution to settle it. Our solution is very simple, effective, cheap(not requiring any hardware modification) and portable across various OS. Experimental results show that our solution can balance throughput and response time very well.

Key words : I/O scheduler, NCQ, expection-discord, starvation

1. 서 론

벤더들이 최신의 소프트웨어/하드웨어적인 특성을 디스크 모델에 구현하면서, 현대 디스크 드라이브의 성능은 과거에 비해 비약적으로 증가하였고, OS 역시 디스크 드라이브를 효율적으로 사용하기 위해 다양한 기법들을 연구하고 구현해 왔다. 하지만 OS와 디스크간의 좁은 인터페이스[1]로 인해, 다양한 정보를 주고받는데 한계가 있었고 입출력 시스템의 효율적인 설계가 저해되어 왔다. OS와 디스크 드라이브의 커뮤니티들은 최상의 입출력 시스템을 설계하기 위한 협조나 소통이 부족했다고 평가받고 있다.

이러한 문제를 해결하기 위해 Object-Based Storage [2]와 Type-safe disk[3]와 같은 연구들이 수행되었다. 그것들은 모두 디스크 인터페이스를 확장하여 OS와 디스크간의 경계를 허물려는 시도였다. 제안된 시스템은 새로운 입출력 시스템의 가능성을 보여주었으며 연구의 경계를 확장하였다. 하지만, 디스크 벤더들이 변화된 인터페이스를 받아들이는 데는 매우 소극적이었으며, 때문에 위 연구들은 산업적으로 활용되지 못하고 여전히 학문적 영역에만 머물러 있는 실정이다.

이러한 정보 차이가 메워지지 않은 상태에서, NCQ가

· 이 논문은 제35회 추계학술대회에서 'NCQ와 I/O Scheduler의 기대 불일치로 인한 입출력 요청의 굶주림 현상 제거하기'의 제목으로 발표된 논문을 확장한 것임

† 비 회 원 : 서울대학교 전기컴퓨터공학
yjyu@dcslab.snu.ac.kr
dishin@dcslab.snu.ac.kr
†† 정 회 원 : 서울대학교 전기컴퓨터공학
iyjung@dcslab.snu.ac.kr
††† 종신회원 : 서울대학교 전기컴퓨터공학 교수
yeom@snu.ac.kr

논문접수 : 2009년 1월 19일
심사완료 : 2009년 3월 26일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제15권 제5호(2009.5)

최근 등장하였고 S-ATA 2 디스크에 구현되었다. 디스크 드라이브는 OS가 어떤 의도로 요청들을 보냈었는지를 개의치 않는다. 단순히 디스크 내의 커맨드 큐에 있는 요청들을 적절히 재조정하여 throughput을 높이는 데에만 모든 관심을 갖는다. 하지만 OS는 자신이 보내준 순서대로만 디스크 드라이브가 요청을 처리할 것이라 믿고 있고, NCQ가 이러한 가정을 어기게 만든다. 이것이 기대 불일치이다.

이 기대 불일치는 예상치 못한 결과를 일으킬 수 있다. 현재 OS에게는 NCQ의 행동을 제어할 수 있는 인터페이스가 제공되지 않기 때문이다. 물론 이를 해결할 수 있는 가장 손쉬운 방법은 디스크의 NCQ를 사용하지 않는 것이다. 그러면 OS의 가정은 여전히 유효하게 되어 디스크는 OS의 의도대로 동작할 것이다.

하지만 그러한 해결책은 NCQ의 잠재적인 이득을 희생해야만 한다는 단점이 있다. NCQ의 스케줄링은 디스크 펌웨어 레벨에서 이루어지기 때문에 논리주소 매핑, 스큐 팩터(skew factor), Seek 곡선, 회전 지연 등의 물리적인 정보를 활용할 수 있다. 이것은 OS가 디스크에 대해 가지는 정보보다 훨씬 많고 정확하므로 더욱 효율적인 스케줄링이 가능하다. 또한 NCQ에 의존할 경우, OS가 따로 디스크를 모델링하거나[4], 디스크 헤드의 위치를 예측할 필요가[5] 없다. NCQ를 사용하는 것은 입출력 시스템을 효율적으로 개선할 수 있도록 하는 기회를 준다. 따라서 쉽게 NCQ를 포기할 수는 없다.

현재 S-ATA 2 디스크는 시장에서 큰 인기를 얻고 있다. 자료에 따르면[6] 3.5인치 드라이브의 66.7%, 2.5인치 드라이브의 44%가 S-ATA 2 인터페이스를 구현했다고 밝히고 있다. 결국에는 S-ATA 2 인터페이스를 가진 디스크 디바이스가 기존의 P-ATA 인터페이스의 디스크를 대체할 것으로 기대하고 있다.

이와 같은 전망에도 불구하고, 이 새로운 디바이스가 기존의 시스템에 결합됨으로써 발생할 수 있는 부작용에 대해서는 언급된 바가 없다. 이 논문에서는 그러한 부작용의 한가지로서 입출력 요청의 굶주림 현상에 대해 확인하고, 해결책을 제안하도록 한다.

2. 입출력 요청의 굶주림

기대 불일치는 심각한 입출력 굶주림을 일으킬 수 있다. S-ATA 2 스펙에는 특정 요청이 다른 요청보다 먼저 서비스 될 수 있도록 하는 어떠한 프로토콜도 포함되어 있지 않기 때문이다. 입출력 스케줄러가 선택한 요청이 디바이스 드라이버에 의해 디스크로 보내진 이후에는, 해당 요청이 끝나기까지 얼마나 오랜 시간이 걸릴지를 전혀 예측할 수가 없게 된다.

요청을 처리하기까지 정해진 시간제한이 전혀 없다면, 서비스 받을 때까지 요청이 지나치게 오랜 시간을 기다리게 될 경우가 생길 수 있는데 이를 요청의 굶주림이라 정의하겠다. 현재의 S-ATA 2 디스크 디바이스를 사용하게 되면, 특정 사용자나 애플리케이션이 요청한 입출력 요청이 심하게 굶주릴 가능성이 있고 이것은 User-Interactive 애플리케이션이나 QoS 기반의 시스템에서 심각한 문제가 된다.

2.1 요청의 굶주림 지표

입출력 요청 R이 디스크 커맨드큐에 도달한 시간을 Arrival(R), 서비스 완료된 시간을 Complete(R) 이라 하겠다. 그러면 Reordering은 다음과 같이 정의된다.

정의. 두 요청 R, R′에 대해서 Arrival(R)<Arrival(R′)이고 Complete(R)>Complete(R′)일 경우 R과 R′은 Reordering 되었다고 정의한다. 이 때 R′이 R을 이겼다라고 말한다.

NCQ는 내부의 동작을 외부에 알려주지 않기 때문에 요청의 굶주림 여부를 알기 위해서는, OS가 간접적으로 추측하는 수밖에 없다. 요청의 굶주림을 나타내는 정도를 굶주림 지표라 한다면, delay count는 효과적인 굶주림 지표가 된다. 특정 요청 R의 delay count를, R을 이긴 다른 요청들의 총 개수라 정의하겠다. 본 논문에 포함되어 있지는 않지만, 대체로 delay count가 높을수록 요청의 응답시간이 길어짐이 밝혀져 있다. Delay count가 정해진 값보다 커지게 되면, 디바이스 드라이버는 해당 요청이 굶주리고 있다고 판단할 수 있다.

2.2 요청의 굶주림 현상 발견

NCQ의 동작을 확인해보기 위해 입출력 스케줄러를 noop으로 설정하고, NCQ depth, 프로세스 수를 변화시켜 가며 실험을 해보았다. 이 Synthetic 벤치마크는 블록디바이스 파일을 열고 지정된 수만큼의 프로세스를 포크하여 디스크 상의 랜덤한 위치에서 한페이지의 데이터를 읽어온다.

그림 2에서 확인할 수 있듯이 NCQ depth가 변함에 따라 요청의 응답시간의 범위가 점점 넓어지는 것을 확인할 수 있는데, 특히 요청의 굶주림의 분포가 변하는 것이 눈에 띈다. NCQ depth가 1에서 31로 변할 때 85%의 요청들은 더 일찍 서비스 되지만, 나머지는 더 오랜 시간이 걸림을 확인할 수 있다. 그런데 이 때 최악의 경우 1.5초 이상의 긴 시간이 걸리는 경우가 발생하기도 한다.

이러한 현상이 발생하는 이유는 NCQ가 throughput을 높이기 위해 특정 입출력 요청을 굶주리게 만들었기 때문이다. 그림 1의 delay count를 보면 어떤 요청들은 다른 요청에게 150회 이상 지게 되는 경우가 발생하기도 하였다. 그 불행한 입출력 요청이 만약 bash 쉘과 같은 User-Interactive 애플리케이션이었다면, 사용자는

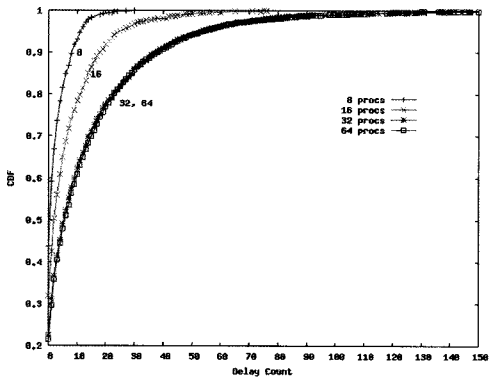

그림 1 Delay Count CDF

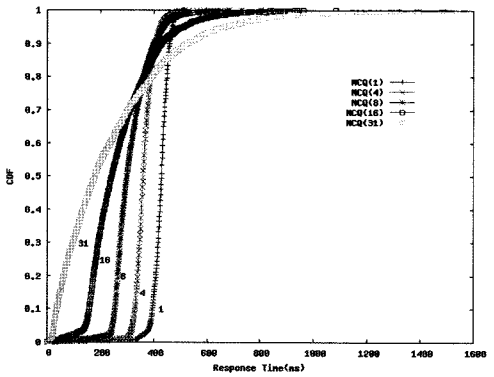

그림 2 응답시간 CDF

표 1 응답시간의 변화와 throughput의 관계

Cumulative Rate	NCQ Depth Level				
	1	4	8	16	31
10 %	397	330	256	160	36
30 %	417	348	275	193	97
50 %	430	361	299	243	179
70 %	441	377	338	311	304
90 %	460	408	404	431	527
95 %	470	427	437	512	671
99 %	490	472	527	697	1030
99.9 %	509	531	681	930	1507
100 %	513	651	838	1083	1565
Tx/s	76.2	91.4	103.2	114.3	123.1

커다란 불편을 느끼게 될 수 밖에 없다.

단순히 NCQ depth를 낮추는 것은 좋은 해결책이 될 수가 없다. 표 1에서 확인할 수 있듯이 최악의 응답시간을 줄이는 효과는 있지만 throughput을 지나치게 희생하기 때문이다. 타이밍 제한이 매우 심한 실시간 애플리케이션에서조차도 성능 역시 중요한 목표로 두고 있음을 상기해야 한다[7].

Semantically-Smart Disk system[8]에서처럼, OS와 디스크 간의 중간 레이어에 간단한 기능을 추가하는 방법을 생각해 볼 수 있다. 이를 통해 throughput과 최악 응답시간의 균형을 효과적으로 이룰 수 있다. 이에 대한 방법이 다음 장에 소개된다.

3. 요청 굶주림 해결책(RSD 메커니즘)

이 장에서는 요청의 굶주림을 해결하기 위해 RSD(Request Starvation Detection) 메커니즘을 제안하고 구현한다. 최근 리눅스 배포판(2.6.19이후)에는 S-ATA 2 디스크 디바이스를 위한 코드가 포함되어 있다. 본 논문에서의 구현은 2.6.22를 기반으로 한다. 현재 S-ATA 2 디스크는 자신을 일종의 SCSI 디바이스로서 등록을 한다. 기존 SCSI 시스템에서 사용했던 태그 관리나 다수의 요청을 동시에 디스크에 보내기 위한 루틴을 그대로 사용하기 위해서이다. 디바이스 드라이버는 S-ATA 2 스펙에 맞추어 적절한 포트 시그널을 가함으로써 입출력을 수행한다.

RSD의 기본 아이디어는 요청의 공평성(fairness)를 위해 블러킹 방법을 사용하는 것이다. 입출력 요청에 태그를 할당할 경우, 만약 굶주리고 있는 요청이 있다면 태그 할당을 미룸으로써 디스크 보내는 것 역시 뒤로 미루겠다는 것이다. 이후에 굶주리던 요청이 서비스 완료 되면 미뤄두었던 태그 할당을 다시 재개하게 된다.

요청의 굶주림 지표로서는 추정된 delay count값을 사용한다. 엄밀히 이 값은 delay count의 최소 추정값으로서 실행 도중에 추정이 가능하므로 구현이 매우 용이하다.

표 2 RSD 메커니즘

```
ASSIGN_TAG():
 1: if wait_tag ≠ 0 then
 2:     Block the request
 3: else
 4:     ≪ItsOwnCode≫
 5:     // Assume "tag" is assigned to current req.
 6:     ageValue[tag] ⇐ -actReqCounter
 7:     actReqCounter ⇐ actReqCounter + 1
 8: end if
FREE_TAG(tag):
 9: clear_bit(tag, wait_tag)
10: ageValue[tag] ⇐ 0
11: for i=0 to MAX_DEPTH-1 do
12:     if i==tag then
13:         continue
14:     else if test_bit(i, Tag_Table)==1 then
15:         ageValue[i] ⇐ ageValue[i] + 1
16:     end if
17:     if ageValue[i] > age_th then
18:         set_bit(i, wait_tag)
19:     end if
20: end for
21: actReqCounter ⇐ actReqCounter - 1
22: ≪ItsOwnCode≫
```

이와 같은 간단한 해결책으로 throughput을 높은 수준으로 유지하면서도 최악 응답시간을 일정 수준 이내로 제한시킬 수 있게 된다. RSD를 구현하기 위해 해주어야 하는 일은 디바이스 드라이버 코드에서 태그를 할당하고 해제해주는 두 루틴을 찾는 것이다. 리눅스 2.6.22의 경우에는 libata-core 모듈의 ata_qc_new()와 ata_qc_free()라는 루틴이 앞의 역할을 담당한다. 몇 개의 변수만을 추가하고 기존의 코드를 그대로 사용할 수 있으므로 매우 이식성이 좋다. 본 구현에서는 libata-core.c 파일을 100라인 이내로 추가하거나 수정하였다. 그림 3에 자세한 알고리즘이 소개되어 있다.

4. 실험 결과

다양한 환경에서 RSD를 구현한 디바이스 드라이버(이하 RSD드라이버)의 성능을 테스트해보았다. 실험 결과(그림 3)로부터 다음의 관찰 결과를 얻을 수 있었다.

관찰 1. 디바이스 드라이버가 RSD를 지원하지 않을 경우, 심각한 입출력 요청의 굶주림 현상이 발생할 수 있다.

RSD를 지원하지 않을 경우 특정 요청이 너무 많이 다른 요청에 지게 될 수 있기 때문이다. 디바이스 드라이버가 NCQ의 기능을 활용하면 활용할수록 최악 응답시간은 매우 커진다.

관찰 2. RSD 드라이버에서 age_th가 커질수록 throughput이 커진다.

age_th는 입출력 요청이 어느 횟수 이상으로는 다른 요청에지지 않도록 제한해주는 기능을 한다. 이 값을 크게 하는 것은 입출력 요청들이 더욱 공격적으로 최적화를 위한 재조정이 되며 그 결과 throughput이 커진다. age_th가 충분히 커지게 되면, 결국 RSD를 사용하지 않는 디바이스 드라이버의 경우와 동일하게 되고 이때의 throughput이 최대이다.

관찰 3. RSD 드라이버에서 age_th가 작을수록 응답시간의 범위가 작아진다.

응답시간이 상위 90%에 해당하는 요청들은 age_th에 상관없이 거의 일정한 값을 가진다. 0~90%의 응답시간은 감소하지만, 상위 90~100%의 요청들은 급격히 응답시간이 증가하게 된다. 즉, age_th가 커질수록 응답시간의 범위가 넓어지게 되는 것이다. 이 경우 사용자에게 일정한 서비스 수준을 보장하는 것이 어렵거나 불가능해 진다. age_th를 낮추어서 응답시간의 범위를 좁게 제한시킬 수 있다.

관찰 4. RSD 드라이버는 non-RSD 드라이버보다 더 효과적으로 응답시간과 throughput의 균형을 맞출 수 있다.

non-RSD 드라이버도 NCQ depth를 낮춤으로써 최악 응답시간을 줄일 수는 있다. 하지만 이는 지나치게 throughput을 희생하게 된다. throughput과 응답시간은 trade-off 관계에 놓여있다. 반면, RSD 드라이버의 경우 throughput을 일정수준으로 유지하면서 최악 응답시간을 줄일 수 있다. {RSD-OFF,Depth=16}과 {RSD=90, Depth=31)의 결과가 이 관측 결과를 뒷받침한다. 이 경우 둘의 최악 응답시간은 비슷하지만, 후자의 경우가 10% 정도 높은 throughput을 보인다. 결과적으로 응답

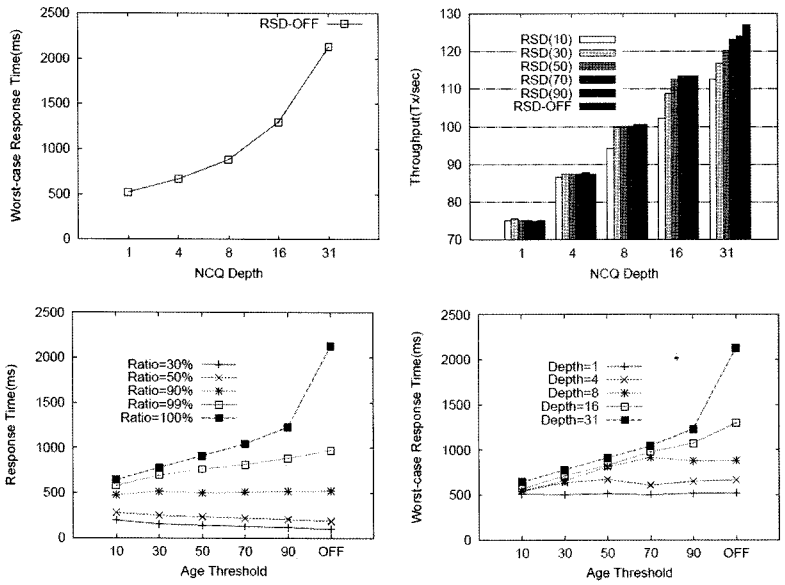

그림 3 RSD 메커니즘 실험 결과

시간과 throughput의 균형을 맞추는 문제에 있어서 RSD가 non-RSD보다 더 효과적으로 문제를 해결할 수 있다고 여겨진다.

5. 결 론

NCQ는 간과해서는 안되는 중요한 기능이다. 가까운 시일내에 디스크의 리소스가 풍부해지면서[9] 더 많은 기능을 제공할 수 있게 될 것이고, 그 결과 freeblock scheduling[10]과 같은 기능들도 실제로 디스크내에 구현될 가능성이 있다.

이러한 진보에도 불구하고 현재의 OS와 디스크 간의 인터페이스는 지나치게 좁아서, 최신의 디바이스 특성들이 효과적으로 이용될 수 있는 기회를 빼앗고 있다. 본 논문에서는 기대 불일치 현상이 가져오는 요청의 굶주림 현상을 분석하고, 이에 대해 간단하고 적은 비용이 드는 RSD 메커니즘을 제안하였다. 이 해결책은 특별한 하드웨어의 추가나 펌웨어의 수정 없이도 가능한 소프트웨어적인 접근 방법이다.

하지만 이것은 차선책에 불과하다. 궁극적으로는 S-ATA 2의 스펙에 NCQ를 통제할 수 있는 인터페이스를 제공할 것을 제안한다. 이 인터페이스를 통해 NCQ 스케줄링 알고리즘을 외부로 추출할 수 있게 하고, 공평한 정책에 영향을 줄 수 있도록 한다면 소프트웨어적인 접근보다 더 좋은 결과를 가져올 수 있을 것이며 호스트 OS의 부담을 크게 덜어줄 수 있을 것이라 확신한다. 물론 이를 위해서는 벤더가 적극적으로 의견을 수용하여 디스크 펌웨어를 개선하는 방법밖에는 없다.

지난 십수년간, 기술의 급진보에도 불구하고 OS와 디스크의 인터페이스는 거의 변한 것이 없다. 인터페이스의 변화에 대해 벤더와 OS진영의 합의가 이루어지기까지는 아직도 많은 시간이 필요할 것으로 보인다. 그 때까지는 둘 간의 정보차이를 인식하고 메꿀 수 있는 방법을 찾는 것이 현실적이고 효과적이라 여겨진다.

참 고 문 헌

[1] Gregory R. Ganger, "Blurring the Line Between Oses and Storage Devices," Technical Report CMU-CS-01-166, 2001.

[2] Mike Mesnier and Gregory R. Ganger and Erik Riedel, "Object-Based Storage," IEEE Communications Magazine, Vol.41, No.8, pp. 84-90, 2003.

[3] G. Sivathanu and S. Sundararaman and E. Zadok, "Type-Safe Disks," Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI'06), 2006.

[4] Dong In Shin and Young Jin Yu and Heon Young Yeom, "Shedding Light in the Black Box: Structural Modeling of Modern Disk Drives," 15th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2007.

[5] L. Huang and T. Chiueh, "Implementation of a rotation latency sensitive disk scheduler," SUNY, Stony Brook Technical Report ECSL-TR81, 2000.

[6] Knut Grimsrud, "SATA-IO: Features Moves SATA Into Smaller Form Factors," Intel Developer Forum (IDF) Techinical Talk, 2007.

[7] S. Chen and J. A. Stankovic and J. F. Kurose and D. Towsley, "Performance Evaluation of Two New Disk Scheduling Algorithms for Real-Time Systems," Journal of Real-Time Systems, Vol.3, pp. 307-336, 1991.

[8] Muthian Sivathanu and Vijayan Prabhakaran and Florentina I. Popovici and Timothy E. Denehy and Andrea C. Arpaci-Dussseau and Remzi H. Arpaci-Dusseau, "Semantically-Smart Disk Systems," 2nd USENIX Conference on File and Storage Technologies(FAST), 2003.

[9] Erik Riedel and Garth A. Gibson and Christos Faloutsos, "Active Storage for Large-Scale Data Mining and Multimedia," Proceedings of the 24th international Conference on Very Large Databases(VLDB'98), 1998.

[10] Christopher Lumb and Jiri Schindler and Gregory R. Ganger and Erik Riedel and David F. Nagle, "Towards Higher Disk Head Utilization: Extracting "Free" Bandwidth From Busy Disk Drives," Proc. of the 4th Symposium on Operating Systems Design and Implementation, 2000.