리눅스에서의 SSD에 대한 IO성능 평가

민항준, 신동군
성균관대학교 정보통신공학부
e-mail : {lucian_min, dongkun}@skku.edu

I/O Performance Analysis of SSD on Linux

Hang-Jun Min, Dong-Kun Shin
School of Information and communication engineering, Sungkyunkwan University

요 약

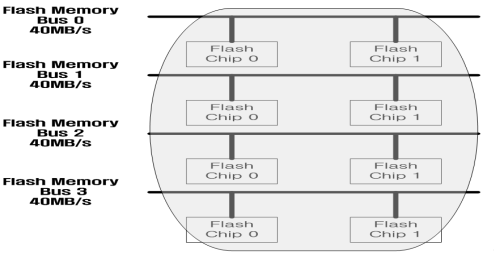
반도체 저장장치의 발달로 인하여 기존의 HDD를 대체할 SSD의 출시와 이를 위한 기술개발 및 성능 향상이 가속화 되고 있다. 이 논문에서는 현재의 운영체제들이 HDD 를 기반으로 최적화하여 제작이 되어 있는데 이러한 최적화 기법이 SSD에서도 유효한지에 대해서 알아보았다. 특히 I/O스케줄러를 변경하여 SSD에서 실험한 결과 SSD의 빠른 응답속도와 대역폭으로 인해 4개의 I/O스케줄러에서 비슷한 성능을 보이는 것으로 측정되었다. 또한 미리읽기기능과 WB에 대해서는 HDD와 마찬가지로 SSD의 성능을 향상 시킬 수 있었다.

1. 서론

최근 반도체 저장장치의 활용도가 매우 커지고 있다. 특히 HDD를 대체하기 위한 낸드 플래시(NAND Flash Memory)로 구성된 SSD[1], [2]의 출시는 무소음, 저전력, 내구성, 빠른 처리속도, 소형화 등의 장점이 있다.

SSD는 HDD와 비교하여 접근시간(Access time)이 매우 빠르고 멀티웨이, 멀티채널로 인해 높은 대역폭을 갖고 있다. 하지만 낸드의 특성을 갖고 있기 때문에 덮어쓰기가 불가능하므로 재 쓰기 또는 갱신 요청 시에는 반드시 삭제 동작이 요구된다. 낸드플래시에서 읽기와 쓰기는 페이지(2 KB)단위로 이루어지고 삭제는 블록(128KB)단위로 이루어지며 이때 읽기는 25us, 쓰기는 200us, 삭제는 2ms가 소요된다. 또한 기존 HDD와의 인터페이스를 맞추고 삭제가 작게 일어나도록 관리하기 위하여 FTL(Flash Translation Layer)[3]을 사용하게 된다.

SSD의 구조는 (그림 1)과 같다.[9]



(그림 1) SSD의 구조

기존의 SATA나 EIDE와 같은 인터페이스의 대역폭은 150 MB/s인데 반해, 낸드는 40MB/s에 불과하다. 그래서 각 버스의 낸드제어기들을 통하여 4개의 버스에 동시에 데이터를 전송한다면 전체 대역폭은 160MB/s가 되므로 기존의 인터페이스 대역폭을 충분히 만족시키게 된다. 또한 버스 내에서도 8개의 칩들이 4쌍으로 묶여있고 I/O처리가 한 쌍으로 이루어지도록 구성이 된다. 따라서 4쌍의 칩에 동시쓰기가 가능하여 쓰기속도는 8배가량 향상되고 4개의 버스를 통해 읽기가 가능하므로 읽기는 4배 정도 향상된다.

이와 같이 SSD의 구조는 HDD와 완전히 다르게 구성이 되어있다. 따라서 현재의 운영체제 하에서 HDD에 최적화 된 기능들이 SSD에서는 어떤 경향을 보이는지에 대해서 이번 논문을 통해서 알아보았다.

각 섹션의 구성은 다음과 같다. 2장에서는 이번 실험을 위한 시스템의 제원 소개와 배경지식을 설명하고 3장에서는 리눅스에서 지원하는 HDD의 최적화 기법에 대한 실험결과를 보여주며 4장에서는 실험의 결론과 앞으로 연구되어야 할 방안에 대하여 논의 한다.

2. 배경지식 및 실험환경

리눅스에서는 느린 I/O의 성능을 개선하기 위해서 I/O스케줄러를 이용한다. I/O스케줄러는 I/O를 요구 큐에 삽입하여 수집, 정렬, 병합 등의 과정을 거쳐 실제 보조저장장치로 전달되는 I/O의 수를 줄이고자 하는데 그 목적이 있다. 리눅스에서 제공되는 I/O스케줄러로는 No-op, CFQ, Deadline, Anticipatory[4][5]가 있는데 2.6.18 이 후로 CFQ가 기본 I/O스케줄러로 사용되고 있다.

시스템	Intel Quad core 2.4GHz, 4GB memory
OS	Fedora Linux 2.6.26
응용프로그램	Postmark, Bonnie++, HD Tach
파일시스템	Ext2, Ext3, Reiserfs
I/O 스케쥴러	No-op, CFQ, Deadline, Anticipatory

(표 1) 시스템 제원

제조사	삼성	웨스턴 디지털
모델	SSD 2.5″ 64GB	HDD 2.5″ 80GB
인터페이스	SATA	SATA(5400RPM)
주 전송률	300MB/sec	300MB/sec
최대 순차 읽기	100MB/sec	150MB/sec
최대 순차 쓰기	80MB/sec	93MB/sec

(표 2) SSD와 HDD의 사양

파일시스템[6], [7], [8]은 실제 보조기억장치에 데이터가

저장될 때 사용되는 자료구조, 접근권한, 메타데이터관리 등을 하는데 있어서 필수적인 요소로서 현재 Linux에서는 여러 가지 파일시스템을 제공하고 있다. 예를 들면, Ext2, Ext3, Reiserfs, XFS, JFS, FAT, NTFS, HPFS 등을 제공하고 있고 현재 Ext3가 기본 파일 시스템으로서 사용되고 있다. 특히 몇몇 파일시스템의 경우 데이터의 신뢰성과 일관성 등을 향상시키기 위하여 저널링 기능을 제공한다.

이번 논문을 진행하면서 사용된 시스템의 사양은 (표 1)과 같다. SSD와 HDD의 제원은 (표 2)와 같다.

작업부하	파일 크기	파일 개수	트랜잭션 개수	총 읽기/쓰기
SS	9k ~ 15k	10,000	100,000	630M/755M
SL	9k ~ 15k	100,000	100,000	600M/1.8G
LS	100k ~ 3M	1,000	10,000	9.7G/12G
LL	100k ~ 3M	4,250	10,000	9G/17G

(표 3) Postmark 작업부하 분류표

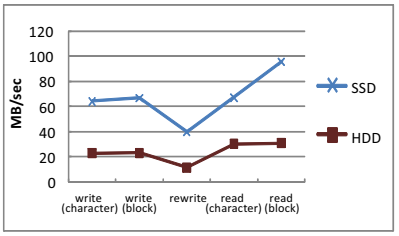
실험 진행을 위하여 벤치마크를 이용하였다. 특히 Postmark에서는 서버와 클라이언트구조에서 발생하는 파일의 작업부하(Workload)[7]와 비슷하도록 설정하여 실험을 진행하였다. (표 3)은 작업부하를 분류한 것을 나타내는데 SS/SL은 랜덤성이 크고 LS/LL은 순차성이 크도록 설정하였다.

3. 성능분석

3.1 Bonnie++

Bonnie++은 표준 C라이브러리의 getc(), putc()등의 함수를 이용하여 보조기억장치의 순차읽기와 쓰기에 대한 성능측정을 위하여 사용되는 벤치마크이다.

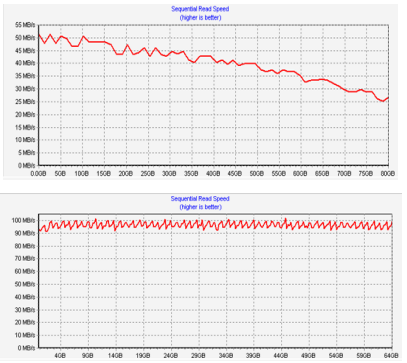
(그림 2)는 SSD와 HDD의 순차읽기와 쓰기에 대한 성능을 Bonnie++을 이용하여 측정한 결과이다. 순차읽기와 쓰기능력에 있어서는 SSD가 HDD에 비해 약 2~3배가량 빠른 처리속도를 낸다는 것을 알 수 있다. 또한 SSD에서 블록읽기가 문자읽기보다 약 1.5배 빠른 속도를 보여 주고 있는데 SSD의 경우 섹터보다 큰 페이지와 블록단위로 처리되기 때문에 블록단위 접근이 훨씬 SSD에 적합하다는 것을 알 수 있고 재 쓰기의 경우 낸드의 특성상 제자리 쓰기가 가능하지 않으므로 성능이 저하되었음을 알 수 있다.



(그림 2) Bonnie++ 실험 결과

3.2 HD Tach

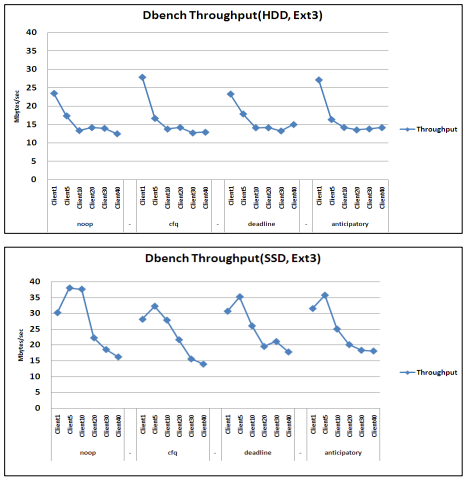
HD Tach에서는 디바이스의 최대용량에 도달할 때까지 순차읽기를 수행한다. (그림 3)을 보면HDD에서 읽기용량이 늘어나면서 처리능력이 점진적으로 감소하고 있음을 확인할 수 있다. 반면 SSD에서는 처리량이 증가해도 읽기능력이 크게 변화되지 않고 있음을 알 수 있다. 이는 읽기용량 증가에 의해 HDD의 평균탐색 및 접근시간이 16.5ms인데 반해 SSD는 0.1ms에 불과하고 SSD의 I/O처리 대역폭 또한 HDD에 비해 크기 때문이다.



(그림 3) HD Tach 실험 결과

3.3 Dbench

Dbench는 Net-bench를 시뮬레이션한 벤치마크로서 여러 개의 응용프로그램이 실행된 것처럼 가상 환경을 만들어준 후 사용자가 설정한 수의 클라이언트들을 생성시킨다. 그 후에 클라이언트들이 실제로 웹을 통해 접속한 것처럼 여러 가지 작업을 요청하게 되는데 이때 발생하는 I/O에 대한 작업처리량(Throughput)을 측정해 준다.



(그림 4) Dbench 실험 결과

(그림 4)는 HDD와 SSD에서 각각 클라이언트를 늘리면서 실험한 결과이다. 이 결과를 보면 HDD의 경우 클라이언트가 하나일 때 가장 좋은 성능을 보여준다. 반면 SSD에서는 클라이언트가 10이 될 때까지도 좋은 성능을 보여주고 있고 또한 HDD에 비해 성능이 완만하게 저하되는 것을 확인할 수 있다. 이는 HDD에 비해 탐색시간과 I/O대역폭이 HDD보다 뛰어나기 때문에 위의 결과가 발생함을 알 수 있다. 또한 흥미로운 것은 SSD에서 CFQ에서 클라이언트의 수가 10이 될 때까지도 가장 좋은 성능을 보여주고 있음을 알 수 있다. 이것은 CFQ가[5] 프로세스마다 공정성(Fairness)을 부여한 I/O스케줄러로서 클라이언트의 수가 늘어남에 따라 I/O대기큐를 할당하게 되므로 더욱 효율적으로 I/O의 처리가 가능하고 빠른 탐색시간과 대역폭으로 인해 많은 I/O처리가 가능하기 때문이다.

또한 클라이언트의 수가 40이상으로 늘어나게 되면 SSD의 성능이 어느 정도 변화가 없는 상태가 되는데 이것은 I/O의

처리속도보다도 CPU의 컨택스트 스위칭(Context Switching) 등에 의한 오버헤드로 인해 성능이 감소하기 때문이다.

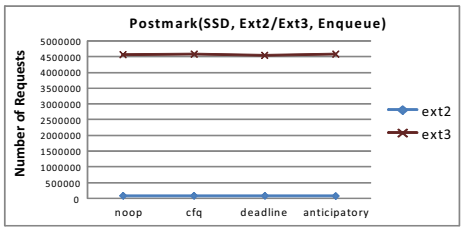
3.4 Postmark

Postmark는 이메일과 같이 크기가 다양하게 변할 때 I/O의 성능을 측정하는 벤치마크이다.

Postmark는 파일의 크기와 트랜잭션의 크기들을 사용자가 임의로 설정을 할 수 있으므로 다양한 작업부하들을 만들 수 있도록 한다. 이 논문에서는 Postmark를 이용하여 HDD에 최적화된 기법들을 SSD상에서 실험해 보았다. 또한 I/O 스케줄러와 파일시스템을 변경해 가면서 실험을 하였기 때문에 I/O스케줄러와 파일시스템이 SSD에서 어떤 경향을 보이는 지 간접적으로 확인할 수 있었다.

3.4.1 I/O 스케줄러의 유용성 실험

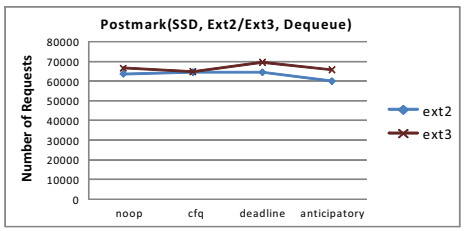
I/O스케줄러는 느린 I/O의 처리를 좀 더 빠르게 하기 위하여 I/O들에 대하여 수집, 정렬, 병합을 실시하여 HDD에서 I/O를 처리하는 수를 줄이도록 고안되었다.



(그림 5) Ext2/Ext3에서의 대기큐에 입력된 I/O개수

(그림 5)는 LL영역에서 Ext2와 Ext3의 인큐(Enqueue)된 I/O개수를 나타낸다. 인큐는 대기큐에 파일시스템으로부터 입력된 I/O의 개수를 나타내고, 디큐(Dequeue)는 대기큐에서 I/O스케줄러에 의해 정렬, 병합 등이 행해진 후 실제 디바이스로 전송된 I/O의 개수를 나타낸다. (그림 6)은 Ext2와 Ext3의 디큐된 I/O개수를 나타낸다.(그림 5와 그림 6의 결과는 HDD에서도 동일한 수준으로 측정되었다.)

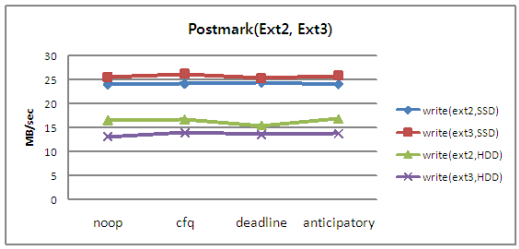
Ext3에서 인큐의 I/O개수가 Ext2에 비해서 상당히 많이 발생한 이유는 Ext3가 저널기능을 가진 파일 시스템이기 때문이다.



(그림 6) Ext2/Ext3에서의 I/O개수

(그림 5)와 (그림 6)을 보면 Ext3에서 대기큐로 들어온 I/O가 상당히 효율적으로 정렬 및 병합이 되어 실제 디바이스로 전송된 것을 확인할 수 있다. Ext3에서 더욱 많은 수의 I/O가 발생했음에도 불구하고 I/O요구 작업처리량은 Ext2에 비해 저하되지 않고 서로 비슷하다는 것을 (그림 7)에서 확인할 수 있다.

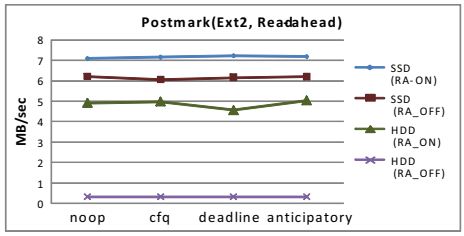
(그림 5,6,7)을 통해서 알 수 있듯이 SSD에서 I/O스케줄러의 성능차이는 미미하다. I/O의 요구수가 조금씩 다르더라도 SSD에서의 빠른 응답속도로 인해 각각의 I/O스케줄러에 영향을 받지 않고 안정된 처리능력을 보여줌을 알 수 있다.



(그림 7) Ext2 와Ext3의 작업처리량 비교

3.4.2 미리읽기 기능의 성능 측정

미리읽기기능은 데이터를 읽을 때 현재영역 이후의 값이 사용될 것으로 예상하고 미리 데이터를 더 읽어서 페이지 캐쉬에 저장하는 기법으로서 HDD의 접근시간을 줄여 읽기에 대한 응답속도를 빠르게 한다. (그림 8)에서 RA-ON은 미리읽기기능을 사용하는 것을 의미하고 RA-OFF는 사용하지 않는 것을 의미한다.

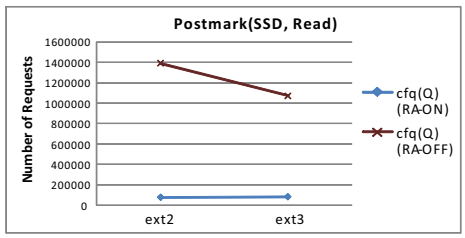


(그림 8) SSD와 HDD에서의 미리읽기기능의 I/O성능비교

HDD의 미리읽기기능은 I/O성능향상에 있어서 상당한 영향을 끼친다. 그림 8에서 보면 미리읽기기능 설정이 안되어 있는 경우 I/O성능이 약 10배 가량 저하됨을 알 수 있다.

반면 성능이 현저하게 감소했던 HDD와 달리 SSD는 약 12% 정도의 성능 저하가 발생했음을 또한 그림 8을 통해 알 수 있다.

미리읽기기능을 사용하지 않았을 때 HDD에 비해서 SSD의 성능은 큰 격차로 차이가 나지 않았는데 그 이유를 분석한 결과는 다음과 같다.



(그림 9) 미리읽기기능의 I/O개수

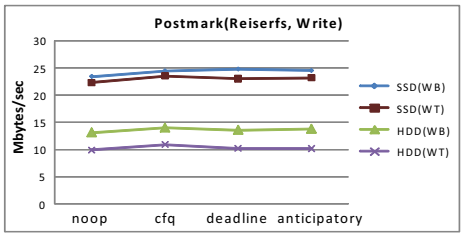
미리읽기기능을 사용하지 않게 되면 페이지 캐시에 저장된 데이터가 없으므로 필요한 만큼 많은 I/O 를 발생하게 된다. (그림 9)는 미리읽기기능을 사용했을 때와 사용하지 않았을 때의 I/O개수를 각각 Ext2와 Ext3에서 측정한 것으로서 미리읽기기능을 사용하지 않게 되면 약 10~15배가량 많은 I/O가 발생하고 있음을 확인할 수 있다.

미리읽기기능을 사용하지 않아서 I/O의 요구량이 현저하게 많아지게 되면 HDD의 경우에는 데이터 탐색시간이 늘어나므로 성능이 I/O의 개수에 비례하여 성능이 저하되는 것에 반해 I/O의 요구량이 많아졌음에도 불구하고 SSD의 성능은 많이 저하되지 않았다. 그 이유는 SSD의 빠른 데이터 접근시간과 대역폭에 의한 I/O처리량이 HDD에 비해 매우 높음을 실험결과를 통해 알 수 있다.

3.4.3 WB(Write-back)과 WT(Write-through)

WB방식은 쓰기요청 발생시 디스크에 직접쓰기보다 캐쉬에 먼저 쓴 후 주기적으로 플러쉬(Flush)를 통해 디스크에 실제 쓰기를 행하는 기법으로써 쓰기가 빈번한 프로세스에서는 성능의 향상을 꾀할 수 있다.

반면 WT방식은 WB와 달리 데이터의 일관성과 신뢰성을 향상시키기 위해서 캐쉬에 쓰는 것뿐 만 아니라 디스크의 내용까지도 갱신시키는 기법이다.

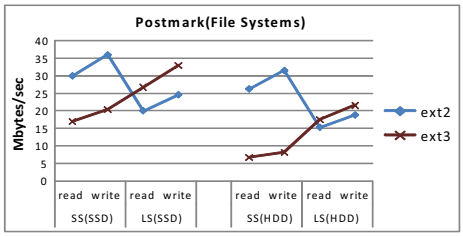


(그림 10) SSD와 HDD에서의 WB와 WT의 성능측정

HDD에서 WB기법을 사용하여 쓰기의 성능을 측정한 결과 약 25%의 성능향상이 있음을 (그림 10)을 통해 알 수 있다. 반면 SSD에서는 약 5%의 성능향상이 있는데 SSD의 빠른 검색속도와 처리속도가 WT방식의 성능을 보완해주고 있음을 알 수 있다.

3.4.4 Postmark

파일시스템은 보조기억장치에 데이터가 저장될 때 자료구조와 접근권한 및 I/O크기 등을 결정해주는 역할을 한다.



(그림 11) SSD와 HDD에서 파일시스템을 변경하며 실험한 결과

Ext2의 경우 SS영역에서 성능이 좋게 나온 반면 LS, LL영역에서는 오히려 저널기능으로 많은 쓰기가 발생하는 Ext3에 비해 성능이 저하되었음을 알 수 있다. 이것은 SS의 작업부하특성이 랜덤성이 강하고 I/O의 개수가 작기 때문에 상대적으로 I/O발생개수가 작은 Ext2에서 성능이 우수하게 나왔다. 또한 LS영역에서는 Ext3의 경우 쓰기 I/O 발생시 4KB단위로 잘라내어 전송하게 되는데 이 때 블록단위로I/O작업을 처리하는 SSD에 더욱 최적화되어 성능이 향상되었음을 (그림 11)을 통해 확인할 수 있다.

HDD에서도 비슷한 결과가 나왔는데 HDD역시 I/O발생시 이를 섹터와 블록단위로 처리하는 블록디바이스이기 때문이다. 하지만 SSD에서 LS, LL영역에서 더욱 좋은 성능을 보인 이유는 SSD는 페이지(2KB)단위로 I/O를 처리하기 때문에 4KB로 나뉘어 I/O가 입력될 때 좀 더 최적화가 되어 처리가 가능하기 때문이다.

4. 결론

이번 논문에서는 SSD와 HDD의 단순성능비교와 함께 미리읽기기능, WB/WT, I/O스케줄러의 유용성에 등에 대해서 살펴보았다. 이번 실험을 통해 HDD에 적용했던 최적화 기법들을 SSD에서 적용했을 때 SSD에서도 다소간의 성능향상이 있음을 확인할 수 있었다. 특히 미리읽기기능은 HDD를 최적화하기 위해 고안된 기법이기에 SSD에서는 성능향상이 이루어지지 않을 것으로 예상했었지만 약 12%정도의 성능이 향상되는 것을 확인할 수 있었다. 그리고 미리읽기기능 실험을 통해 SSD의 I/O처리속도가 HDD와 비교하여 현저하게 빠르다는 사실을 알 수 있기 때문에 이와 같은 특성은 SSD가 최적화될 때 고려해야 할 사항이라고 할 수 있다.

또한 I/O 스케줄러의 경우에도 많은 쓰기가 발생되는 저널링에서 I/O의 정렬, 병합, 수집 등을 통해 디바이스로 보내지는 I/O의 수를 현저하게 줄임으로써 성능향상을 위해 꼭 필요하다는 것을 실험을 통해 보였다.

SSD는 HDD가 갖고 있었던 기계적인 부분을 제거함으로써 I/O처리에서 지연되는 컴퓨팅시간을 상당하게 줄일 수 있었다. 그리고 위의 실험 결과로서 SSD의 빠른 접근시간과 대역폭에 부응하는 I/O 스케줄러와 파일시스템, 그리고 운영체제의 최적화가 더해진다면 더욱 빠른 I/O처리가 가능할 것이다.

추가 연구할 사항으로는 SSD를 위한 I/O스케줄러의 개발과 파일시스템의 개발, 그리고 운영체제의 I/O 패스 개선으로 인한 성능 향상이 다음 연구주제들이다.

참고문헌

[1] 배영현. “고성능 플래시 메모리 SSD설계 기술”, 정보과학회지, 제 25권, 제 6호, 페이지 18-28, 2007.

[2] Samsung Electronics. “Solid State Drive Data Sheet,” http://www.samsung.com/global/business/semiconductor/products/flash/Products_FlashSSD.html

[3] S. W. Lee, D. J. Park, T. S. Chung, W. K. Choi, D. H. Lee, S. W. Park, H. J. Song. “A log buffer based flash translation layer using fully associative sector translation,” ACM Transactions on Embedded Computing Systems, Volume 6, 2007

[4] Jens Axboe, “Linux Block I/O-present and Future,” Proceedings of the Linux Symposium, Volume 1, 2004

[5] Steven L. Pratt, Dominique A. Heger., “Workload Dependent Performance Evaluation of the Linux 2.6 I/O Schedulers,” Proceedings of the Linux Symposium, Volume 2, 2004

[6] Redhat Inc. “Redhat’s New Journaling File System: ext3,” 2001 http://www.redhat.com/support/wpapers/redhat/ext3/ext3.pdf

[7] M. Cao, T. Y. Ts’o, B. Pulavarty, S. Bhattacharya, A. Dilger, A. Tomas, “State Of The Art: Where We Are With The Ext3 File System,” Ottawa Linux Symposium, 2005

[8] M. Tim Jones, “Anatomy Of Linux Journaling File Systems,” IBM developer works, 2008