메타 데이터 영역의 직접 접근을 활용한 고속 파일 검색 기법

김예성[O] 최종욱 이성진 김지홍
서울대학교 컴퓨터공학과

A Fast File Search Technique Using Direct Access of Metadata Area

Yeseong Kim[O], Jongwook Choi, Sungjin Lee, Jihong Kim

요 약

데이터와 파일의 개수가 증가함에 따라 효율적인 파일 검색 기법에 대한 요구가 증가하고 있다. 현재까지 제안된 파일 탐색 기법은 탐색 시간이 매우 오래 걸리거나, 미리 인덱싱 과정을 거쳐야 하기 때문에 사용자가 원하는 시점에 빠르고 효율적인 파일 탐색을 제공하기 어렵다는 단점을 지닌다. 본 논문에서는 디스크의 메타 데이터 영역을 직접 접근함으로써 고속의 파일 검색을 가능하게 해주는 효율적인 검색 기법을 제안하고자 한다. 본 기법은 메타 데이터가 서로 비슷한 영역에 위치한다는 파일 시스템의 특징을 활용함으로써 디스크 내의 데이터를 고속으로 탐색하고, 이를 DB로 구성함으로써 실시간 검색을 지원한다. 또한 파일 시스템 내의 변경 사항을 빠르게 추적하여 DB를 업데이트 할 수 있다는 장점을 지닌다. 본 논문에서 제안한 검색 기법을 다수의 시스템에서 평가한 결과 기존의 기법 대비 약 20~50배의 탐색 시간 향상과 10초 미만에 실시간 검색이 가능한 상태를 구성 할 수 있음을 확인 할 수 있었다.

키워드: 파일 탐색, 파일 검색, 파일 시스템, 메타 데이터

1. 서 론

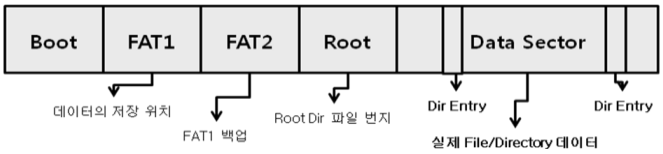
파일 검색은 일반 사용자나 시스템 관리자뿐만 아니라, 보안상 취약한 파일을 구분하는 백신 및 보안 프로그램, 사법 기관에서 사용하는 포렌식 도구와 같은 다양한 응용에서 활용되는 필수적인 기능이다. 최근 저장 장치의 공간이 빠르게 증가하고, 저장되는 파일의 종류와 개수가 늘어남에 따라, 저장 장치 내 파일들의 빠르고 효율적인 검색은 점차 어려워지고 있는 추세이다. 따라서 이런 문제를 해결하기 위한 효과적인 파일 검색 기법에 대한 연구가 매우 시급한 실정이다.

효율적인 파일 탐색을 위해 현재까지 다양한 연구가 진행되어 왔다. 먼저, 파일 검색을 위한 가장 일반적인 방법으로는 파일 시스템 전체를 디렉터리 구조에 따라 순회하면서 검색하는 방식을 꼽을 수 있다. 그러나 이러한 방식은 매우 긴 탐색 시간을 요구한다는 단점을 지닌다. 이러한 검색 시간을 줄이기 위하여, 시스템의 유휴 시간 동안 전체 파일 리스트를 미리 탐색하여 인덱싱을 수행해 놓는 방법이 이용되고 있다. 그러나 긴 탐색 시간을 소요하는 인덱싱 과정이 반드시 선행되어야 한다는 한계를 가지고 있다.

이러한 한계를 극복하기 위하여 새로운 파일 시스템 차원에서, Semantic File System[1]이나 Logic File System[2]과 같은 DB 기반의 파일 시스템이 제안된 바 있다. 이러한 파일 시스템에서는 디렉터리 정보가 담겨있는 메타데이터 구조를 트리 대신 DB로 구성하여 탐색 속도의 향상을 도모하였다. 하지만 이러한 파일 시스템은 기존 파일 시스템의 대체제로 제안되었기 때문에 이식성이 떨어져 실제 많이 사용되지 못한다.

본 논문에서는 현재 널리 쓰이는 파일 시스템을 그대로 사용하면서 파일 검색을 빠르게 할 수 있게 하고, 변경 사항을 추적할 수 있는 방법을 제시하고자 한다. 여기서는 탐색과 인덱싱 시간을 개선하기 위해 디렉터리 트리를 순회하는 방법을 이용하지 않고, 디스크에서 디렉터리 구조를 관리하는 메타 데이터 영역만으로 파일 시스템 전체 구조를 빠르게 파악 할 수 있다는 점에 착안하여 이 영역에 직접 접근하는 방식을 사용한다. 또한 반복적인 검색에 대응하기 위하여 DB를 구성하고 이를 통해 파일명, 파일 크기, 시간 등을 검색하는 방법을 이용한다. 이 DB는 점증적으로, 임의의 시간 간격을 가지고 달라진 두 메타 데이터 영역을 선형 시간 내에 비교하여, 변경 사항을 추적하는 동시에 변경된 부분만 DB에 갱신 할 수 있는 방법도 함께 제시한다. 실험 결과, 기존의 탐색 기법에 비해 20~50배 가량 우수하여, 50만개 파일을 실시간 검색이 가능하게 DB를 구성하는데 걸리는 시간이 10초 내외면 충분함을 확인하였다.

본 논문의 구성은 다음과 같다. 2장에서는 본 연구의 배경 지식 및 관련 연구를 살펴보고, 3장에서는 본 논문이 제시하는 파일 탐색/검색 방법과 변경 사항 추적 방법을 설명하고, 현재 널리 사용되고 있는 NTFS 파일 시스템을 대상으로 세부적인 구현 내용을 설명한다. 그리고 4장에서는 실제로 구현된 기법의 성능을



[그림 1] - FAT 파일 시스템

분석한다. 마지막으로 5장에서는 본 논문의 결론과 향후 연구에 대하여 논의한다.

2. 배 경

2.1. 일반적 파일 탐색 기법

파일 시스템을 탐색하는 가장 일반적인 방법은, 트리 구조로 구성된 파일 시스템에서, Root 디렉터리부터 하위 디렉터리나 파일의 Entry를 하나씩 재귀적으로 순회하는 것이다. 이 방식은 FAT와 같은 고전적 파일 시스템들의 특성에 적합하였다. 당시 디스크의 크기가 일반적으로 작았으므로, 디렉터리 정보를 담은 독립된 메타 데이터 영역을 관리하지 않고, 여유가 있는 공간에 [그림 1]과 같이 분산하여 메타 데이터를 저장한다. 따라서 이러한 파일 시스템에서는 디렉터리 Entry가 흩어져 있어 다른 방법 자체를 사용하기가 어렵다.

이러한 탐색 방법은 Windows 파일 탐색기나, Linux의 find 명령어 등과 같이 최근 파일 시스템에서도 여전히 일반적인 방법으로 사용되고 있다. 그러나, 과거와 달리 최근에는 파일의 개수가 크게 늘어났기 때문에, 어떤 조건의 파일 정보를 검색할 때 이러한 탐색 방법은 더 이상 효율적이지 않게 되었다.

2.2. Indexing 기법을 이용한 파일 검색

이런 검색 시간의 한계를 극복하기 위하여 파일을 미리 인덱싱하는 기법이 등장하였다. Windows 7과 같은 최근 운영 체제에서는 이러한 기능이 내장되어 있으며, 같은 방식을 이용한 Desktop Search[3]와 같은 응용 프로그램도 발표되고 있다. 이 방법에서는 사용자가 선택할 경우, 시스템의 유휴 시간에 파일 시스템 전체에 대하여 DB를 구성하고 검색을 하는 방식을 사용한다.

이를 통해 검색 속도는 빨라지는 효과를 얻을 수 있지만, 인덱싱 자체 과정에 시간이 오래 걸리는 것은 물론, 파일 리스트에 대한 인덱스가 미리 만들어지지 않은 경우엔 사용자가 요청이 있는 시점에 바로 결과를 보여줄 수 없다는 한계를 지니고 있다. 또한 대부분의 인덱싱 기술은 파일 내용을 어떻게 다룰 것인지에 초점이 맞추어 있어, 파일 리스트 자체의 탐색 및 검색에는 효율적인 방법을 제시해주지 못한다.

2.2. DB를 이용한 파일 시스템

디스크가 가진 파일의 개수가 증가하면서, 디렉터리 트리 구조만으로 빠른 탐색 및 분류가 어렵게 되어, DB를 이용한 새로운 파일 시스템이 제안되고 있다. 최초의 연구는 Semantic File System[1]으로 Query를 받고 그에 대한 결과를 돌려주는 DB를 사용자 레벨에

[그림 2] - NTFS 파일 시스템

구성하여, 고속 파일 검색을 할 수 있는 파일 시스템을 최초로 제안하였다. 후에 나온 대표적인 연구로 Logic File System[2]이 있는데, 이는 파일 시스템의 메타 데이터를 Berkeley DB로 구성하고 propositional logic을 이용하여 탐색을 수행한다. 그 외에, 파일 시스템 구조에 직접 DB를 기록하는 WinFS[4]와 같은 연구가 진행된 바 있다. 이러한 연구들은 파일 시스템에 DB를 사용한다는 아이디어를 제시해주고 있지만, 새로운 파일 시스템을 구성해야 하기 때문에 현실에 바로 적용되기는 힘들다는 제약이 있어, 본 논문이 제시하는 기존 파일 시스템을 활용하는 기법과 그 목적이 다르다고 할 수 있다.

3. 파일 탐색 방법 및 변경 사항 추적

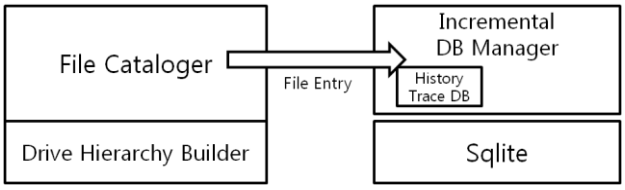
3.1. 연구 동기

본 연구에서 제안하는 기법은 디렉터리 트리를 순회하는 방식을 이용하지 않고, 파일 시스템의 메타 데이터를 직접 접근하여 파일을 탐색함으로써, 전체 파일 시스템에 있는 파일과 디렉터리를 빠르게 검색 할 수 있게 한다. 이러한 방식은 현재 널리 쓰이고 있는 파일 시스템들의 구조적 특성에 기반을 두고 있다.

[그림 2]는 현재 가장 널리 쓰이고 있는 파일 시스템이자 본 기법의 실제 성능을 테스트한 NTFS의 구조를 나타낸다. 여기서는 MFT(Master File Table)라는 지정된 영역에 메타 데이터를 저장하고 있다. MFT도 일종의 파일로 관리되므로 분할되어 다른 영역에 저장되어 있을 수도 있지만, 분할된다 하여도 2~3개 이내인 경우가 거의 대부분이다.

디렉터리 트리 순회를 이용한 탐색 방법과 미리 Indexing을 구성하는 방식 모두 디스크 I/O가 병목으로 작용할 수 밖에 없다. 그러나 같은 디렉토리의 자식들이더라도 메타 데이터가 저장된 곳이 대부분 흩어져 있을 수 밖에 없기 때문에, 트리 순회 방식은 디스크 캐시를 충분히 사용하지 못하게 된다. 그러나 위에서 제시한 바대로 메타 데이터가 몰려있는 디스크 특성에 의해, 순차 읽기 방식은 디스크 캐쉬를 거의 전부 활용할 수 있게 된다. 다시 말해, 메타 데이터 영역을 읽는 과정에서, 순차적 지역성(Sequential Locality)에 의한 성능 향상을 그대로 이용하면서 파일 리스트를 비롯한 파일 시스템 구조를 확보할 수 있다.

EXT3, 4의 경우에는 일부 메타 데이터가 디스크에 분산되어 있어 추가의 오버헤드가 발생 할 수 있으나, 전체 파일의 위치를 저장하고 있는 inode block은 역시 한 군데에 몰려있어 적어도 트리 탐색보다 전체 파일 리스트의 확보 시간이 짧다고 예상된다.



[그림 3] - 본 기법의 전체 구성도

3.2 본 기법의 전체 구성

본 기법은 [그림 3]과 같이 크게 두 가지로 구성된다. 첫 번째 단계로 Drive Heirarchy Builder는 디스크를 분석하여 메타 데이터 영역의 위치를 찾고, File Cataloger는 전체 파일 시스템의 파일과 디렉터리 정보를 하나씩 추출하는 역할을 한다. 이 단계는 디스크의 메타 데이터를 직접 읽는 방식으로 진행된다. 두 번째 단계로 Incremental DB Manager에서 추출된 파일 Entry를 하나씩 DB로 구성하고 반복적인 검색을 할 수 있게 한다. 이 때 이미 DB가 구성된 상황에서는 파일이 변경된 점을 추적하여 갱신을 할 수 있게 한다.

3.3. File Cataloger: 메타 데이터에서 파일 리스트 추출

먼저 Drive Heirarchy Builder는 메타 데이터 영역의 위치를 찾는다. 파일 시스템에 따라 다를 수 있지만, 일반적 으로 디스크의 0번 Sector로부터 MBR(또는 GPT)을 읽어 파티션을 확보하고, 각 파티션으로부터 Boot Sector를 읽어 메타 데이터 영역이 어디에 위치하는지를 확인하는 과정으로 수행 할 수 있다. NTFS의 경우에는 MFT의 위치가 여기에 해당한다.

다음으로 MFT는 일종의 파일이므로 분할되어 있는 위치를 고려하여 전체 파일에 대한 메타 데이터를 가져오는데, 메타 데이터는 Record라 불리는 2 Sectors (1024 Bytes) 하나 하나가 각 파일에 대응된다. 이 Record는 순차적으로 연결되어 있는 구조로 저장되어 있으며, 이 안에 통상 Attribute라고 불리는 단위로 파일에 대한 각종 정보-파일명, 생성, 접근, 수정 시간, 파일 크기 등을 저장하고 있다. 따라서 MFT 파일을 순차적으로 읽으면서 전체 파일 시스템의 구조에 대한 리스트를 확보 할 수 있다. 이 때 지워진 파일의 흔적이 남아 있는 경우도 확인할 수 있으므로 이를 별도로 저장하여 후에 파일을 복구하는데 이용할 수도 있다.

3.4. DB Manager: 파일 시스템 구조의 DB화 및 검색

다음으로 반복 검색을 위해, 전체 파일 시스템의 구조를 읽는 동시에, 추출한 각 파일 정보를 DB로 구성한다. 본 연구에서는 가볍고 유니코드를 지원하는 DB인 sqlite[5]를 이용하여 개발 하였다. 또한 파일명에 대해서는 실시간 검색을 할 수 있도록 인덱스를 생성한다. 물론 파일명 외에 시간이나 파일 크기로도 검색을 할 수 있고, 확장자 등으로도 검색을 할 수 있으므로 파일의 효율적인 분류도 가능하게 된다. 따라서 반복하여 검색을 수행 할 경우라도 디스크를 재탐색 하지 않고 DB로부터 결과를 얻을 수 있다.

더불어, 보다 효율적인 구성을 위해 DB를 디스크가 아니라 메모리에 구성하도록 하였다. 파일 시스템에 저장되어 있는 메타 데이터 정보에는 검색에 사용되지 않는 다른 데이터도 함께 포함되어 있기 때문에, DB의 크기는 메타 데이터 영역에 비해 작아지게 된다. 실제로 파일이 50만개일 때 메타 데이터 영역의 크기가 약 500MB라 할지라도, DB는 인덱싱을 생성한 후에도 30~50MB 정도로 구성된다. 따라서 메모리에 충분히 올라갈 정도의 크기가 되어 Disk I/O를 줄일 수 있다. 또한 인덱싱한 데이터를 추후에 사용해야 할 경우에는 파일로 저장해 두었다가 불러올 수 있도록 하였다.

3.5. 점증적인 DB 갱신과 변경 사항 추적

DB가 한번 구성되고 임의의 시간 후에 다시 파일 시스템의 구조를 읽었을 때, 탐색 시간이 짧더라도 DB를 재구축하면 단 1초라도 비효율적이므로, 파일을 하나씩 추출하면서 동시에 DB를 갱신하는 것이 좋다. 따라서 변경된 부분만을 빠르게 찾는 방법이 필요하다.

이를 위해 DB를 구축할 때 메타 데이터가 저장되어 있는 Record 순서에 맞추어 Index를 매겨 DB에 저장하게 한다. 그리고 이후에 다시 파일 시스템의 구조를 갱신할 때, 먼저 이전에 저장되어 있던 목록을 ($Index_0$, $Metadata_0$) pair를 순서대로 가져온 후, 현재 읽고 있는 ($Index_1$, $Metadata_1$) pair 엔트리와 다른 점이 있는지 순차적으로 하나씩 비교를 한다. 그러면 다음 [표 1]과 같이 변경 사항을 판정할 수 있다.

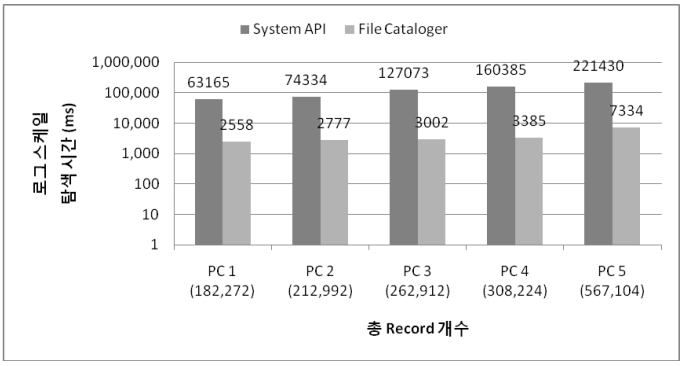
변경 없음	($Index_0$=$Index_1$) & ($Metadata_0$ = $Metadata_1$)
수정	($Index_0$=$Index_1$) & ($Metadata_0$ ≠ $Metadata_1$)
생성	($Index_0$>$Index_1$)
삭제	($Index_0$<$Index_1$)

[표 1] - 변경 사항 판정 방법

이 방법을 이용하면 이전 파일 개수 O와 현재 얻어온 파일 개수 N이 있을 때 선형 시간 내 (Θ(O+N))에, 변화된 항목을 확인 해가며 DB를 갱신 할 수 있다. 덧붙이면, 삭제 후 새로 생성된 파일이 수정으로 판단될 가능성이 존재하므로 생성 시간으로 한번 더 비교해 주는 것이 정확하다. 이 변경된 정보를 시간에 따라 별도의 DB([그림 3]의 History Trace DB)에 저장하면 추후에 변경 사항을 추적하는데 이용 할수 있다.

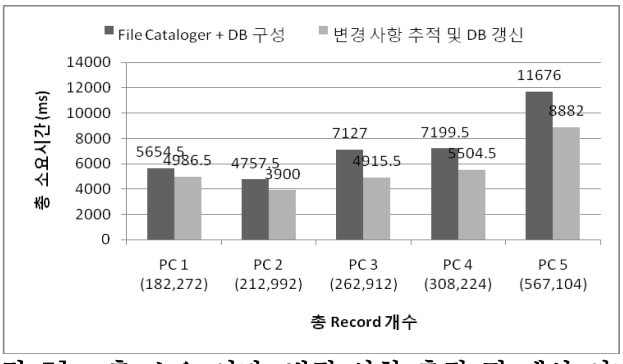
4. 실험 결과

성능을 분석하는데 사용된 시스템은 현재 널리 사용되고 있는 500 GB의 용량에 7200 RPM의 회전수와 16 MB의 버퍼를 가진 하드 디스크, Intel Core 2 Quad Q9400의 CPU, RAM 4 GB가 장착된 PC 환경 하에서 테스트를 하였다. 총 다섯대의 컴퓨터에서 주 드라이브(C드라이브)에 대해 정보를 수집하였으며, 디스크 버퍼 캐시에 따라 영향을 받을 수 있으므로 충분한 시간 간격을 두고 반복하여 성능을 측정하였다.



[그림 4] - File Cataloger와 System API의 탐색 시간 비교

먼저, [그림 4]는 검색을 위한 DB를 구성하는 시간을 제외하고 순수히 파일 탐색하는 시간에 대하여, System API을 사용한 기존의 디렉터리 순회 탐색 방법과 본 기법에서 사용하는 File Cataloger의 탐색 시간을 비교한 것이다. 사용자가 사용하는 한 드라이브에 있는 파일의 개수가 일반적으로 50만개를 넘지 않으므로, 본 기법을 사용하면 약 6초 내외에 모든 파일에 대한 탐색이 완료되는 것을 확인할 수 있다. 기존의 기법과 비교하여 약 20~50배 가량의 속도 향상이 있는데, 트리 기반 탐색을 이용하지 않고 바로 메타 데이터에 접근하는 것이 전체 파일 탐색 시간에 큰 효과를 가져다 주는 것을 확인 할 수 있다.



[그림 5] - 총 소요 시간, 변경 사항 추적 및 갱신 시간

다음으로 [그림 5]는 파일을 탐색하여 DB를 구성하고 인덱스를 생성하는 시간까지 포함하여 측정한 시간과, 일정 시간후 다시 탐색을 수행하여 변경 사항을 추적하고 DB를 갱신할 때 소요된 시간에 대한 테스트 결과이다. 이 과정을 거치면 실시간 검색이 가능한 상태가 된다. 5~10초 이내로 구성이 완료되므로, 유휴 시간에 미리 인덱싱을 하는 방식을 이용하지 않아도, 충분히 빠른 시간내에 검색을 할 수 있는 상태를 만들 수 있는 것을 확인하였다. 또한 변경 사항을 추적하여 DB를 갱신하는 것이 새롭게 DB를 만드는 것보다 10%~30%정도 빠른 것도 확인 할 수 있다.

[표 2]는 메타 데이터의 Record 수와, 삭제된 파일을 제외한 정상 파일 수, 메모리에 구성된 DB의 크기를 보여준다. 디스크에서 읽은 메타 데이터의 총 크기는 Record 개수에 각각의 크기인 1024를 곱하여 알 수 있으므로, 약 1/10 정도로 DB 크기가 줄어들어 메모리에 충분히 올라 갈 수 있는 크기임을 알 수 있다.

	Record 개수	*정상 파일 개수*	*DB 크기(MB)*
PC 1	182,272	172,662	19
PC 2	212,992	211,286	26
PC 3	262,912	260,493	30
PC 4	308,224	297,767	31
PC 5	567,104	558,363	58

[표 2] - 총 Record 개수 대비 정상 파일 개수, DB 크기

5. 결 론

기존의 파일을 검색하는 방법은 디렉터리 트리 순회를 기반으로 하기 때문에 시간이 오래 걸리는 것은 물론, 반복된 검색에 대응하지 못한다는 단점을 지니고 있었다. 또한 미리 인덱스를 만든다 하여도 빠른 응답이 필요한 상황에서는 이용할 수 없는 문제점을 지니고 있었다. 기존에 제시된 방법들은 대부분 파일 시스템을 직접 수정하거나 새로운 파일 시스템 모델을 제시하여 해결하려고 하였다.

본 논문에서 제시한 방법은 기존에 널리 사용되는 파일시스템 구조를 그대로 유지하면서, 메타 데이터 영역만을 순차적으로 접근하는 방식으로 빠르게 파일을 검색 가능한 상태로 DB화 시킬 수 있다. 또한 DB도 점증적으로 빠른 갱신이 가능하다. 실험 결과, 제안한 파일 탐색 기법은 기존의 기법 대비 20~50배 우수한 성능을 제공함을 확인할 수 있었다. 현재 PC에서 널리 사용되고 있는 파일시스템이 오랜 기간 동안 이러한 구조를 유지하고 있고 앞으로도 그럴 것으로 생각되므로, 후의 파일시스템에도 적용될 수 있을 것으로 기대한다.

앞으로 본 논문의 접근 방법을 확장하여 NTFS 뿐만 아니라 EXT 시리즈와 같은 다른 파일 시스템에도 적용해볼 계획이다. 이를 통하여 제안한 기법이 실제로 얻을 수 있는 다른 이득은 어느 정도이고 차이점은 어떤 것이 있을지, 그러한 구조에 맞는 다른 접근 방법은 무엇이 있을지 연구를 계속 진행해 보려 한다.

6. 참고 문헌

[1] K. Gifford, P. Jouvelot, A. Sheldon, and W. O'Toole Jr., “Semantic File Systems,” In Proceedings of the 13th ACM Symposium on Operating Systems Principles (SOSP '91), pp. 16-25, 1991.

[2] Y. Padioleau, B. Sigonneau, and O. Ridoux, “LISFS: A Logical Information System as a File System,” In Proceedings of the 28th International Conference on Software Engineering (ICSE '06), pp. 803-806, 2006.

[3] Google Desktop, http://desktop.google.com/

[4] T. Rizzo, “WinFS 101: Introducing the New Windows File System,” Microsoft Corp., 2004.

[5] M. Owens, “The Definitive Guide to SQLite,” Apress, 2006.