디지털산업정보학회 논문지 제16권 제2호-2020년 6월
http://dx.doi.org/10.17662/ksdim.2020.16.2.011

파일시스템의 클러스터를 임의로 할당하여 디스크를 단편화하기 위한 방법*

조 규 상**

An Arbitrary Disk Cluster Manipulating Method for Allocating Disk Fragmentation of Filesystem

Cho Gyu-Sang

〈Abstract〉

This study proposes a method to manipulate fragmentation of disks by arbitrarily allocating and releasing the status of a disk cluster in the NTFS file system. This method allows experiments to be performed in several studies related to fragmentation problems on disk cluster. Typical applicable research examples include testing the performance of disk defragmentation tools according to the state of fragmentation, establishing an experimental environment for fragmented file carving methods for digital forensics, setting up cluster fragmentation for testing the robustness of data hiding methods within directory indexes, and testing the file system's disk allocation methods according to the various version of Windows.

This method suggests how a single file occupies a cluster and presents an algorithm with a flowchart. It raises three tricky problems to solve the method, and we propose solutions to the problems. Experiments for allocating the disk cluster to be fragmented to the maximum extent possible, it then performs a disk defragmentation experiment to prove the proposed method is effective.

Key Words : Disk Cluster Allocation, Manipulating Disk Fragmentation, Disk Defragmentation, NTFS Filesystem

Ⅰ. 서론

단편화(fragmentation)는 기억 장치에 자료가 저장될 때 여러 개의 조각(섹터 또는 클러스터 단위)으로 나뉘는 현상을 말한다. 이런 현상으로 인해서 기억장치의 가능한 사용 공간을 줄이거나, 읽기/쓰기의 속도를 늦어지는 문제가 일어난다. 이런 단편화를 제거하는 것은 물리적 기억장치 상위에 존재하는 파일 시스템에서 파일 내용의 물리적인 저장위치를 다시 정리하여 각 파일의 내용을 연속적으로 놓이도록 재배치하여 접근 속도를 빠르게 해주는 것이다[1]. 이렇게

* 이 논문은 2018년도 동양대학교 학술연구비의 지원으로 수행되었음.
** 동양대학교 컴퓨터학과 교수

하는 것을 디스크 조각모음(defragmentation)이라고 한다[2].

단편화 제거를 위한 디스크 조각 모음은 마이크로소프트 Windows XP부터 사용하기 시작하여 Windows 10에서도 사용하고 있는 유틸리티 프로그램이다[3]. 디스크의 헤더의 이동을 최소화하여 파일의 읽기와 쓰기의 성능을 최적화하여 빠른 전송 속도를 내는 것이 목적이다. 최근에 많이 사용되는 SSD(Solid Stage Drive), USB 메모리, SD 카드 등에서는 드라이브 헤드가 없어서 디스크 조각모음이 적용이 되지 않는다. 그러므로 이 경우는 디스크 단편화와 디스크 조각모음으로 인한 문제는 없다. 디스크 조각모음은 Windows 8부터 마이크로소프트 드라이브 최적화(Microsoft Drive Optimizer)라고 명칭이 변경되었다[4].

디스크의 단편화 여부가 민감하게 적용되는 연구분야들이 있다. 최근에 디지털 포렌식 기술의 발전과 함께 이에 대응하기 위한 방법으로써 안티포렌식 기술들이 심도있게 다뤄지고 있다. 안티포렌식 기술은 스테가노그라피, 아티팩트 지우기, 암호화, 데이터 감추기 등의 여러 가지 방법들로 분류된다[5]. 그 중에서 디스크의 슬랙영역이나 파일이나 디렉토리 구조의 슬랙 영역을 이용한 안티포렌식 방법[6]은 디스크 조각모음에 의하여 데이터가 저장된 클러스터들이 옮겨 질 때 영향을 받는다. 이에 해당하는 방법들 중에서 Berghel등[7]의 방법은 정상적인 파일인 것처럼 위장하여 파일시스템의 슬랙영역에 데이터를 숨기는 방법이고 Huebner등[8]의 방법은 $BadClus속성, $Boot속성, 디렉토리 $Data 속성을 이용하여 비할당 공간과 슬랙 공간을 이용하는 방법이다.

단편화된 디스크에서 디스크 할당문제가 중요하게 작용되는 연구들이 있다. 최근 Karresand등[9]의 연구에서는 Windows의 NTFS의 클러스터 할당방법에 대한 실험을 통해서 사용자 데이터가 저장될 때 데이터의 위치가 어디가 될 것인가를 확률적인 방법을 구현하는 실험을 수행하였다. 이 연구에서는 Windows의 여러 버전의 시스템들에서 파일의 생성, 복사, 파일내용의 확장, 축소 등의 작업을 랜덤하게 수행하여 파일이 생성되는 클러스터 위치를 측정하여 전체 디스크 중에서 중앙부분에 집중적으로 많은 파일들이 할당된다는 것을 결론적으로 구하였다. 이 연구를 통해서 스토리지의 적절한 영역에 우선권을 부여하여 hash-based carving하는데 사용될 수 있어서 디지털 포렌식을 수행하는데 효과적일 것이라고 주장하였다[9].

Bahjat등[10]의 연구에서는 일반적으로 할당되지 않은 공간이나 파일 슬랙영역에서 발견되는 삭제된 파일 단편들에 대한 시간 관계를 알아내기 위한 프레임워크 방법을 제안하였다. 이웃 클러스터의 시간적 상태를 알고 있으면 파일 단편이 저장 매체에 쓰여진 날짜 및 시간 범위를 도출할 수 있다고 가정하여 단편화된 조각을 맞출 수 있는 방법을 제안하였다.

NTFS 파일시스템의 구조를 이용한 데이터 숨기기 방법이 저자에 의하여 제안되었다[6]. 목록이 많아지면 여러 개의 인덱스 레코드에 저장되고 이것들은 B-tree 구조를 사용하여 관리된다. 디렉토리 안의 파일명을 이용하여 파일의 생성, 삭제, 변경 작업을 수행하여 인덱스 레코드 안에 남는 흔적을 이용하여 데이터를 숨기는 방식이다. 이 방법은 인덱스 레코드 안의 목록은 자동적으로 오름차순으로 유지되는 특징이 있다는 점과 B-tree의 노드로 구성된 인덱스 레코드들은 파일목록이 삭제되어 빈 인덱스 레코드가 되어도 런-리스트에서 제거되지 않고 유지된다는 점에 착안하여 기술적인 기교를 적용하여 데이터를 숨기는 방법이다. 이 방법에서 디스크 조각모음의 수행에 의하여 관련 클러스터의 위치에 변동 생기는 경우에도 이 방법에 의하여 숨겨진 데이터들이 강인하게 유지되면서 목적을 달성될 수 있음을 실험적으로 보여야 한다.

이 논문에서는 NTFS 파일시스템에서 디스크 클러

스터의 할당상태를 임의로 설정하고 해제하여 디스크의 단편화를 제어할 수 있는 방법을 제안한다. 이 방법을 이용하여 디스크의 단편화 문제와 연관된 여러 연구들에서 실험을 수행할 수 있다. 적용 가능한 대표적인 연구 사례는 단편화의 상태에 따라서 디스크 조각모음 도구의 성능 측정 실험, 디지털 포렌식 과정에 필요한 단편화된 파일들의 카빙 방법에 대한 실험 환경구축, 디렉토리 인덱스 내에 데이터 숨기기 방법의 강인성 실험을 위한 클러스터 단편화 설정, 윈도우즈의 버전에 따라 파일시스템의 디스크 할당 방법[6-10]을 검증하기 위한 실험 등을 들 수 있다.

이 논문에서 제안하는 임의로 디스크 클러스터를 할당하고 해제하기 위한 방법은 한 개의 파일의 데이터가 한 개의 클러스터를 차지하는 방법을 사용한다. 이 방법을 구현하는데 있어서 여러 난점들이 존재하는데 2장에서 이런 문제 해결의 난점에 대한 존재하는 것을 명시하고 이 문제들에 대한 해결책의 필요성을 제기한다. 3장에서는 제안한 방법에 대한 알고리즘을 플로우차트와 함께 제시한다. 4장에서는 제안한 방법이 범용성을 갖는 방법과 도구로 구현될 수 있도록 특정한 경우에 발생할 수 있는 문제를 제시하고 그것에 대한 해결방안을 기술한다. 5장에서는 제안한 방법에 대한 실험을 수행한다. 이 실험에서는 디스크 클러스터를 최대한으로 단편화시킨 사례를 설정하고 그것에 대한 조각모음을 수행하는 실험을 한다. 6장에서는 이 연구의 기여와 후속 연구에 대하여 언급하면서 결론을 맺는다.

Ⅱ. 문제의 설정 (Problem Statement)

이 연구에서는 NTFS 파일시스템에서 디스크 클러스터의 할당상태를 임의로 설정하고 해제하여 디스크의 단편화를 제어할 수 있는 방법을 제안한다. 이 방법에서는 다음과 같은 문제들이 제기되며 이 문제점을 해결할 수 있는 방법을 제시하는 하는 것이 이 연구에서 수행해야 하는 과제이다.

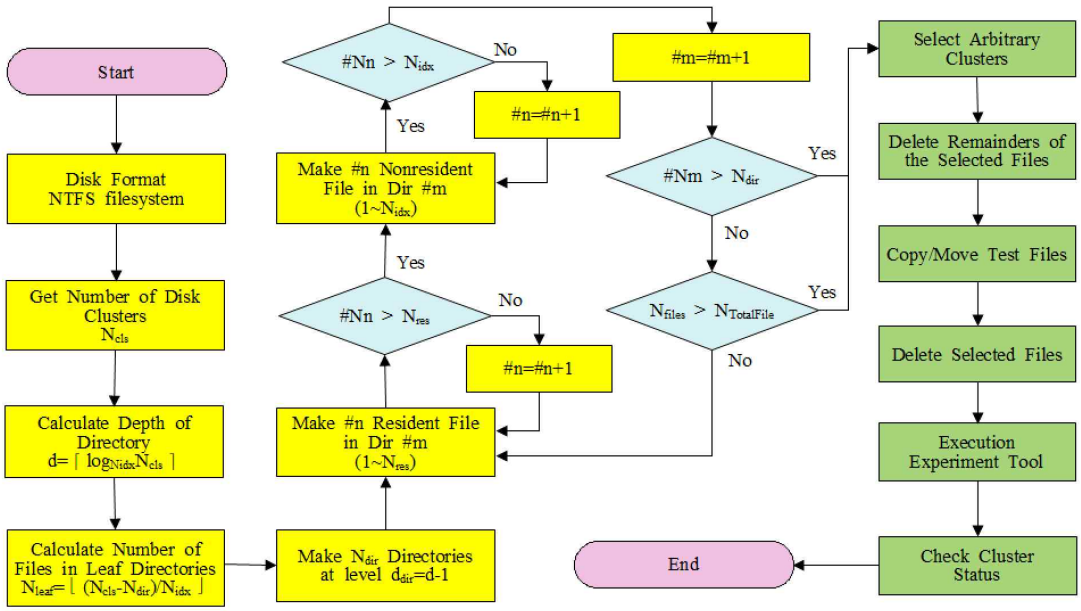
문제 #1: 디스크의 클러스터 당 한 개의 파일을 할당할 수 있는 방법을 제시한다.

문제 #2: 문제 #1에서 많은 파일의 목록(파일의 수)에 따라 B-tree의 노드(인덱스 레코드)가 많아진다. 이 때 인덱스 레코드가 저장되는 클러스터의 위치를 미리 알기 어려운 문제를 해결한다.

문제 #3: 문제 #2를 해결하기 위해 디렉토리 내의 목록 수를 제한하는 과정에서 파일 클러스터와 디렉토리 인덱스 레코드의 클러스터의 생성순서가 뒤바뀌는 특이현상이 발생하는 문제를 해결한다.

문제 #1: 디스크의 클러스터 당 한 개의 파일을 할당하는 것은 클러스터의 할당과 해제를 하는데 있어서 문제를 단순화하기 위한 방법이다. NTFS 파일시스템에서는 $MFT에 각 파일의 정보는 MFT 엔트리에 저장된다. 이 안에는 600~700바이트의 파일의 내용이 저장될 공간이 존재하는데 그 공간에 저장되면 상주(resident) 데이터라고 부른다. 이 용량을 넘어서는 파일의 데이터는 독립적인 클러스터에 저장된다. 이것을 비상주(non-resident)라고 부른다. 클러스터 당 한 개의 파일을 저장하기 위해서는 파일의 데이터가 비상주 속성으로 저장되어야 하며 크기는 4KB이하여야 한다. 이런 특성을 이용하여 파일을 생성함으로써 문제 #1에 대한 해결책을 제시할 수 있다.

문제 #2: 디스크의 클러스터 당 한 개의 파일을 할당하기 위해서는 파일시스템의 시스템 정보를 제외하고 남는 부분의 클러스터의 수만큼 할당할 수 있다. 그 클러스터의 수만큼 파일을 생성해야 하는데 이 수는 디스크의 용량에 선형적으로 비례한다. 그러므로 매우 많은 파일을 만들어야 한다. 디스크의 드



<그림 1> 디스크 클러스터를 임의로 할당을 하기 위한 방법의 플로우차트

라이브의 루트에서나 특정 디렉토리에서는 마찬가지로 많은 파일들을 만들게 되면 파일의 목록을 저장하기 위하여 여러 개의 인덱스 레코드들이 생성된다. 이 인덱스 레코드는 B-tree 구조의 노드로 작용되는데 많은 노드들이 분할과 합병의 과정을 통해서 B-tree의 데이터 구조가 유지될 때 저장되는 클러스터의 위치가 순차적이지 않아서 예상하기 어려운 위치에 할당되는 문제가 발생한다. 이 방법을 해결하기 위한 알고리즘이 3장과 4장에서 제안된다.

문제 #3: 일반 파일과 마찬가지로 디렉토리들도 $MFT내의 MFT 엔트리에 저장된다. 목록의 수가 적으면 상주 속성인 $INDEX_ROOT 속성에 인덱스 엔트리(파일 목록)가 저장된다. 파일명의 길이가 10-15 문자인 경우는 6개 저장될 수 있는 공간이다. 파일의 목록이 늘어나서 이 공간이 부족하게 되면 $INDEX_ALLOCATION속성으로 확장된다. 이 속성은 비상주 속성으로 인덱스 레코드라고 불리는 4KB 크기의 클러스터를 사용하게 된다. 파일목록이 상주 속성 데이터에서 비상주 속성 데이터로 바뀌는 과정에서 파일의 생성 순서대로 클러스터에 저장되지 않는 특이한 문제가 발생한다. $INDEX_ROOT 속성에 저장된 것이 $INDEX_ALLOCATION속성으로 확장되면서 저장 클러스터의 순서가 변경되어 일련으로 생성한 파일의 목록과 실제 저장되는 위치의 차이가 발생하는 문제가 생긴다. 이런 경우가 발생하면 클러스터의 할당과 해제를 임의로 관리하기 어려운 문제가 생긴다. 이것을 해결할 수 있는 방법을 3장과 4장에서 제시한다.

III. The Proposed Scheme

이 연구에서 제안하는 NTFS 파일시스템의 클러스터 단편화 제어 방법은 다음과 같은 절차로 수행된다.

<표기법>

d: 파일이 생성될 위치의 레벨을 d라고 하고 이것을 리프 레벨이라고 부름

d_{dir}: 서브 디렉토리가 생성되는 가장 깊은 레벨. 이 안에 파일들이 생성된다.

N_{cls}: 디스크 안에 들어 있는 사용되지(점유되지) 않은 클러스트의 전체 수

N_{idx} : 한 디렉토리 안에 갖는 파일의 전체 수 또는 서브 디렉토리의 전체 수

N_{dir} : 디렉토리의 전체 수

N_{files}: 파일의 전체 수

$N_{TotalFiles}$: 생성한 파일의 전체 수

N_{crit}: 디렉토리 인덱스가 레지던트에서 논-레지던트로 학장되는 경계값(critical value)

$Size_{file}$: 파일 내용(데이터)의 크기

$Size_{crit}$: 파일 크기(내용)이 레지던트에서 논-레지던트로 확장되는 경계값

$\lceil x \rceil$: ceiling값. 소수 값을 올림한 것

절차 1: 디스크 준비: 대상 디스크 드라이버를 NTFS 파일시스템으로 포맷한다. 기본적으로 NTFS는 4KB단위로 클러스터를 생성한다.

절차 2: 디스크 전체의 클러스터 개수를 구한다. DeviceIoControl()함수에 인수를 FSCTL_GET_NTFS_FILE_RECORD로 지정하여 구한다. 또는 "fsutil fsinfo ntfsinfo 드라이브명:"을 사용하여 전체 비어있는 클러스터의 수를 구한다.

절차 3: 디렉토리 깊이를 계산한다.

$$d = \lceil \log_{Nidx} N_{cls} \rceil \quad (1)$$

파일은 리프 디렉토리에만 생성된다. 그리고 디렉토리의 목록 정보는 한 클러스터를 차지하도록 설정된다. 그러므로 전체 할당되어야 하는 클러스터의 수는 할당 디렉토리 수와 리프 디렉토리에 생성되는 파일의 수의 합이 되어야 한다.

$$N_{files}+N_{dir} < N_{idx} \quad (2)$$

전체 리프노드에 만들어질 파일의 수는 다음과 같이 계산된다.

$$N_{files} = \lfloor (N_{cls}-N_{dir})/N_{idx} \rfloor \quad (3)$$

절차 4: 디렉토리 만들기

루트 디렉토리를 생성하고 그 안에 서브디렉토리들을 반복적으로 생성한다. 최하단의 디렉토리 레벨은 d_{dir} = d-1이다.

$$makeDir(dirPrefix, d_{dir}, N_{dir}) \quad (4)$$

절차 5: 상주형식 파일 만들기

d_{dir} 레벨에서 상주형식의 파일을 생성한다. 파일데이터의 크기는 600 ~ 700바이트 미만이 되도록 정한다[11]. 파일의 개수는 상주형식으로 디렉토리 안에 1~N_{res}개의 파일을 생성한다.

$$createResFile(filePrefix, ext, d, N_{res}) \quad (5)$$

디렉토리 목록으로 저장되는 인덱스 엔트리의 한 개의 크기는 인덱스 엔트리 헤더 16바이트, $FILE_NAME 속성의 고정요소 64바이트와 이 안에 들어 있는 파일명이 가변 길이로 구성된다. 파일명을 testFile-001.txt 16자(유니코드 2바이트)를 사용하는 경우는 파일명의 길이정보 2바이트를 포함하여 다음과 같이 계산된다. 16 + 64 + 2+ 16*2 = 114바이트가 되므로 6개까지 파일목록이 상주속성의 $INDEX_ROOT에 저장될 수 있다[12].

절차 6: 비상주 형식의 파일 만들기

d_{dir} 레벨에서 파일을 생성한다. 파일의 데이터 크기는 $700B < Size_{file} < 4KB$ 까지 내용을 갖도록 생성한다. 이렇게 하면 한 개의 파일은 한 개의 클러스터에 할당된다.

createFile(filePrefix, ext, d, N_{files}) (6)

디렉토리 안에 생성되는 파일들은 트리구조에서의 리프노드로 간주할 수 있다. 각 디렉토리마다 이 리프노드들(파일들)의 전체의 수는 다음과 같다.

$$N_{files} < N_{idx} - N_{dir} \quad (7)$$

여기서 디렉토리 내에 생성되는 파일은 file#1 ~file#N_{files}까지 N_{files}개의 파일을 생성한다. 이중에서 절차 5에서 생성한 file#1~file#N_{res}의 파일을 파일의 데이터 크기가 800B ~ 4KB 바이트가 되도록 비상주 형식으로 다시 생성해야 한다.

현재까지 생성한 파일의 수가 다음의 조건을 만족하면 절차 5로 이동하고 그렇지 않으면 절차 7을 수행한다.

$$N_{TottalFiles} < N_{files} \quad (8)$$

절차 7: 클러스터의 할당 해제

최하위 디렉토리에 들어 있는 리프노드 파일들에 대하여 삭제할 파일을 선택한다. 이 파일들을 삭제하는 것으로 클러스터의 할당을 해제한다. 디스크의 단편화 상태와 정도는 삭제할 파일의 간격의 선택으로 달성된다.

delete(filePrefix, ext, num) (9)

절차 8: 대상파일의 복사

할당 해제된 클러스터의 빈 공간에 들어갈 수 있는 대상 파일을 복사한다. NTFS의 클러스터 할당방식은 최적할당(best fit)인 것을 고려하여 대상 파일(fragFiles)을 복사한다. 대상 파일이 여럿인 경우는 클러스터의 할당과 파일의 복사과정을 순차적으로 반복한다.

copyFiles(fragFiles) (10)

절차 9: 잔여 클러스터의 할당 해제

절차 7에서 수행하고 남은 파일들에 대하여 삭제 명령을 수행하여 클러스터의 할당을 해제한다. 잔여 클러스터의 할당을 해제하는 작업은 실험의 목적에 맞게 남은 파일들에 대한 선택을 해야 한다.

delete(filePrefix, ext, num) (11)

절차 10: 실험의 수행

절차 9를 수행하고 난 뒤의 단편화된 파일들에 대하여 실험(예: 디스크조각 모음)을 수행한다.

execExperi(fragFiles) (12)

절차 11: 디스크조각 결과리포트

대상 디스크에 대한 실험수행의 결과를 리포트한다.

reportCluster(disk) (13)

예를 들어, 최대의 단편화를 설정하여 디스크 조각 모음 실험을 수행하는 경우에는 파일명 “filePrefix+num.txt”의 형태로 번호를 사용하여 삭제 대상을 정한다. 홀수 번호의 파일을 먼저 삭제한다. 그 후에 대상파일을 복사하여 파일을 단편화한 후에 나머지 짝수 번호의 파일들을 삭제하면 디스크에는

조각화된 실험파일 파일만이 남는다. 이것에 대하여 디스크 조각모음을 수행한다. 이와 유사한 방식으로 파일카빙이나 안티-포렌식 실험의 경우에 디스크 단편화를 설정한 후에 필요한 실험을 수행할 수 있다.

Ⅳ. 단편화 작업의 문제점과 해결책

디스크 클러스터의 단편화를 조작하기 의도대로 수행되도록하기 위해서는 클러스터 당 한 개의 파일을 할당하고 파일의 번호와 클러스터 번호를 연계하면 쉽게 위해서 원하는 클러스터를 할당/해제를 할 수 있다(2장의 문제 #1). 그러나 NTFS 파일시스템의 고유한 특성으로 인하여 클러스터 할당/해제의 조작이 어려운 경우가 발생한다. 이런 문제가 발생하는 이유와 해결방법을 다음과 같이 제안한다.

4.1 상주 속성과 비상주 속성

NTFS에서 파일과 디렉토리 정보가 파일시스템에 $MFT에 MFT 엔트리가 저장된다. 이것은 크기는 1KB이다. 이 안에는 파일에 관한 기본 속성들이 들어있고 가변크기의 파일명이 저장된다. $DATA 속성에 파일의 내용을 저장하는데 600~700바이트 정도가 가능하다. 이것을 상주(resident) 속성이라고 부른다.

디렉토리의 경우도 마찬가지로 기본 속성들이 들어 있고 인덱스 엔트리를 저장하기 위해서 $INDEX_ROOT와 $INDEX_ALLOCATION 속성을 갖는다. $INDEX_ROOT는 상주(resident) 속성으로 저장된다. 이 안에는 파일목록의 개수가 5~6개 정도(파일명의 길이에 따라 가변적) 이 안에 저장된다. 더 많은 목록이 저장되는 경우는 $INDEX_ALLOCATION에 목록이 저장된 클러스터의 정보(런-리스트)가 저장되는데 이것을 비상주(non-resident) 속성이라고 부른다. MFT엔트리의 내부가 아닌 외부의 클러스터에 파일목록을 저장하고 그 위치를 $INDEX_ALLOCATION속성에 저장한다. 목록의 숫자가 점차 늘어나면 B-tree방식으로 클러스터들의 데이터 구조를 관리한다. 이 때 클러스터의 이름을 인덱스 레코드(index record)라고 부른다.

이 연구에서는 파일 한 개당 한 개의 클러스터를 할당하는 방식을 사용한다. 이 방식은 디스크를 다루는데 있어서 8개 섹터로 구성된 4KB로 포맷된 클러스터가 접근할 수 있는 최소단위이기 때문에 이것을 할당의 기준으로 선택한다.

파일당 한 개씩 $MFT안에 기록되는 MFT 엔트리에 비상주 방식으로 데이터를 저장한다. 비상주 속성으로 데이터가 저장될 때 $MFT 파일에서 벗어난 위치의 외부의 클러스터에 저장하게 된다. 파일의 데이터 크기를 700바이트 이상 4K바이트 이하로 저장하면 자연스럽게 한 개의 클러스터에 저장된다. 새로 포맷된 디스크에서는 순차적으로 클러스터를 할당하기 때문에 파일의 번호가 되는 순서에 맞춰서 클러스터의 번호가 증가한다. 이렇게 저장하게 되면 파일에 해당하는 클러스터의 위치가 선형적으로 구성되기 때문에 클러스터의 할당과 해제를 하기 쉬워진다. 이런 특성을 이용하여 파일과 클러스터를 다루어서 문제 #1에 대한 해결책을 제시할 수 있다.

4.2 파일목록의 수의 중가에 따른 비상주 속성의 확장 문제

파일의 목록이 증가하면 파일의 목록을 저장하고 있는 클러스터의 수도 증가하게 된다. 실험을 통해서 이 문제를 확인할 수 있다. <그림 2>에서 한 디렉토리에만 파일 생성한 경우에는 “1st Cls.” 위치에 첫 번째 디렉토리 인덱스 정보가 할당된다. 파일의 수가 늘어나면 되면 “1st Cls.”에 이어서 담을 수 있는 “2nd Cls.”로 확장하여 저장한다. 계속적으로 파일의 수가 증가하면 “3rd Cls.”. “4th Cls.”등으로 지속적으

로 확장된다.

이 실험에서는 836개의 파일을 생성하였을 때에 “Direcotry Idx. Cls”라고 표시된 클러스터가 디렉토리 인덱스가 저장된 곳이다. 인덱스 레코드를 저장한 클러스터들이 지속적으로 확장해서 여러 개의 클러스터에 연속으로 저장되는 경우도 발생한다. 이 중에서 “Last Cls.”도 이 경우에 해당한다. 결론적으로 한 디렉토리 안에 많은 파일을 생성하는 경우는 파일들의 목록을 여러 개의 인덱스 레코드 클러스터에 저장하게 되고 이것의 위치가 불규칙적이며 여러 개의 연속된 클러스터를 사용한다는 문제점이 발생한다(문제#2).

파일의 목록이 늘어나고 줄어듦에 따라서 B-tree의 노드(인덱스 레코드)의 수가 변하게 된다. 이 때 인덱스 레코드가 저장되는 위치가 선형적으로 계산되지 않는다는 점에서 어려움이 발생한다. 어떤 경우는 연속적인 클러스터에 할당하기도 하고 어떤 경우는 분산되어 저장하기도하는 특징을 보이기 때문에 다루기 어렵다. 이 문제를 해결하기 위해서 이 논문에서는 인덱스 엔트리가 한 개의 클러스터 내에만 저장되고 더 이상 확장하지 않도록 인덱스 엔트리 수의 조절을 통해서 이 문제를 해결하고자 한다.

1KB크기의 $MFT엔트리에 저장되고, 파일명의 길이에 따라 저장될 수 있는 파일 내용의 최대 크기가 정해진다. 비상주 속성의 경우는 한 클러스터에 4KB 크기까지 저장된다. 목록의 수가 많아지면 여러 클러스터에 걸쳐서 목록이 저장되는데 $INDEX_ALLOCATION에 이 정보가 들어 있다.

디렉토리 내의 파일의 목록정보 저장에 사용되는 인덱스 엔트리의 한 개의 크기는 인덱스 엔트리 헤더(IEH) 16바이트, $FILE_NAME 속성의 고정요소(FNA) 64바이트와 이 안에 들어 있는 파일명이 가변길이로 구성된다. 파일명(FNL)을 testFile-01.txt 15자(유니코드 2바이트)를 사용하는 경우는 파일명의 길이정보(FNI) 2바이트를 포함하여 다음과 같이 계산된다. 16 + 64 + 2+ 15*2 = 112바이트가 되므로 6개까지 파일목록이 상주속성의 $INDEX_ROOT에 저장될 수 있다[12].

디렉토리 안에 6개 보다 많은 파일목록이 생기는 순간에 비상주 형식으로 4KB 크기의 인덱스 레코드에 목록을 저장한다. 한 개의 인덱스 레코드 안에 들어갈 수 있는 파일의 목록 수는 다음과 같이 계산할 수 있다. 이 경우에는 인덱스 레코드 헤더(IRH, 64바이트)와 인덱스 엔트리 끝 표식(End mark of index entry, EIE, 16바이트)를 추가로 고려해야 한다.

$$TotalIR = IRH+(IEH+FNA+FNI+FNL)*N_{idx}+EIE$$
$$4096 = 64+(16+64+2+15*2)*n+16 \quad (14)$$

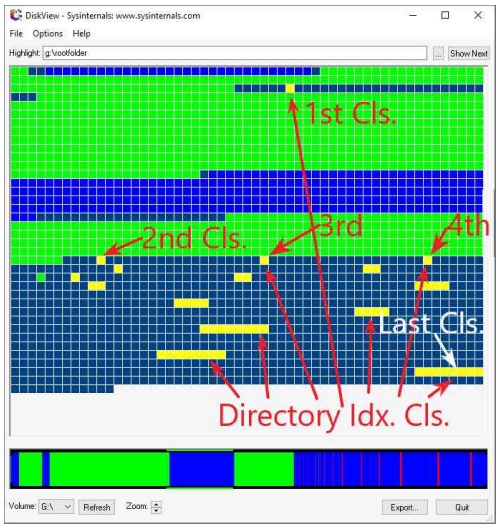
$$N_{idx}=\lfloor (4{,}096-80)/112 \rfloor = \lfloor 35.85 \rfloor = 35 \quad (15)$$

파일명이 15자인 경우는 4KB 크기의 인덱스 레코드 안에는 최대 35개의 인덱스 엔트리가 저장된다. 파일명이 길이에 따라 가변적으로 계산된다는 점을 반드시 고려해야 한다.

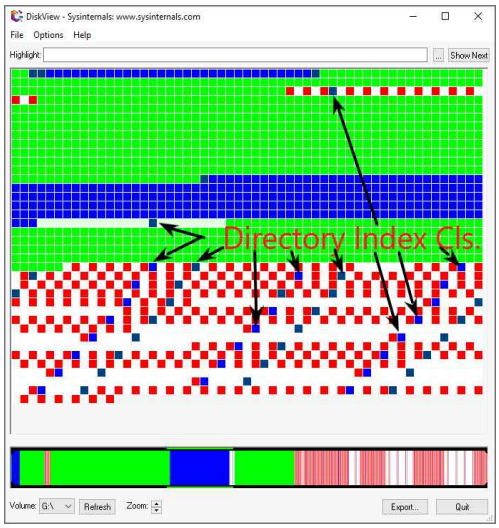
이 연구의 사례에서는 디렉토리 당 파일 N_{idx}을 35개씩 저장하여 35개의 클러스터를 할당하고 디렉토리 인덱스 엔트리 정보를 1개 클러스터에 할당하여 총 36개의 클러스터 세트를 사용하여 문제#2를 해결하였다.

4.3 클러스터 할당 순서 변경 문제

4.2절의 문제 #2의 해결방안을 수행하는데 있어서 또 다른 문제가 야기된다. 한 디렉토리 안에 들어가는 파일의 목록 정보의 총 합이 4KB크기를 넘지 않도록 파일의 목록 수를 조절한다. 파일명 testFile-01.txt(15자)형식은 한 디렉토리에 N_{idx}=35개



<그림 2> 한 디렉토리에만 파일생성한 경우-여러 클러스터에 디렉토리 인덱스 저장



<그림 3> 파일 클러스터 -인덱스 클러스터 순서 섞이는 현상

를 넘지 않으면 이 조건을 만족한다.

그러나 이 방법에 있어서도 또 다른 문제가 발생한다. 파일이 차례대로 할당되는 과정 중에 인덱스 레코드의 중간에 할당되는 경우가 발생한다(<그림 3> 참조)

앞에서 TestFile#1_01부터 시작하여 6개의 파일의 클러스터가 먼저 차지하고 그 후에 디렉토리의 인덱스 클러스터 DirectoryIndex#1가 생성된다. 이것은 파일목록이 저장된 인덱스 레코드가 상주 속성인 $INDEX_ROOT에 저장되어 있다가 파일의 목록이 6개를 넘는 순간에 저장공간이 부족하여 비상주 속성인 $INDEX_ALLOCATION으로 변경되기 때문이다. 다음과 같은 순서로 발생한다.

TestFile#1_01 → TestFile#1_02 → TestFile#1_03 → TestFile#1_04 → TestFile#1_05 → TestFile#1_06 → DirectoryIndex#1 → TestFile#1_07 → ···· → TestFile#1_35 → TestFile#2_01 → TestFile#2_02 → TestFile#2_03 → TestFile#2_04 → TestFile#2_05 → TestFile#2_06 → DirectoryIndex#2 → TestFile#2_07 → ···· → TestFile#2_35 ····· TestFile#n_m →

이 문제를 해소하기 위하여 다음과 같은 절차로 파일과 디렉토리를 생성한다.

절차1: 최초로 루트 디렉토리를 생성한다. 그 안에 서브디렉토리를 생성하는데 디렉토리는 d_{dir} 레벨까지 생성한다.

$$d = \lceil \log_{Nidx} N_{cls} \rceil \quad (16)$$

$$d_{dir} = d-1 \quad (17)$$

절차 2: 디렉토리 내에 파일을 순차적으로 N_{crit}미만 개를 생성하고 파일의 크기(내용)가 $Size_{crit}$ 미만인 파일을 생성한다.

사례에서는 fileName-##.txt 형식으로 15자를 사용한다.

절차 3: N_{crit}=6 보다 큰 파일을 생성한 직후에

fileName-01.txt ~ fileName-06.txt의 파일 내용을 700byte ~ 4KB바이트 미만(700 < $Size_{file}$ < 4KB)으로 다시 갱신한다.

절차 4: 파일명의 번호가 N_{idx}에 도달할 때까지 파일일을 생성하고 파일의 내용을 700 < $Size_{file}$ < 4KB 크기가 되도록 저장한다.

절차 5: 리프노드의 최대 파일수 N_{idx}에 도달할 때까지 반복한다.

이 절차를 수행하고 나면 다음과 같은 순서로 파일과 디렉토리가 저장된다. 디렉토리 정보가 파일과 섞이지 않기 때문에 파일과 디렉토리에 대한 조작이 쉬워져서 특정 클러스터에 대한 작업을 의도적으로 수행할 수 있다.

DirectoryIndex#1 → TestFile#1_01 → TestFile#1_02 → TestFile#1_03 → TestFile#1_04 → TestFile#1_05 → TestFile#1_06 → TestFile#1_07 → · ··· → TestFile#1_35 → DirectoryIndex#2 → TestFile#2_01 → TestFile#2_02 → TestFile#2_03 → TestFile#2_04 → TestFile#2_05 → TestFile#2_06 → TestFile#2_07 → ···· → TestFile#2_35 ····· TestFile#n_m →

Ⅴ. 실험

5.1 실험 환경

제안한 방법을 구현하기 위하여 실험환경을 다음과 같이 구축한다. 실험의 수행시간을 절약하기 위하여 디스크 장치를 작은 파티션으로 구성하고 NTFS 파일시스템으로 포맷한 직후에 fsutil 명령어로 얻은 디스크 장치의 정보는 다음과 같다.

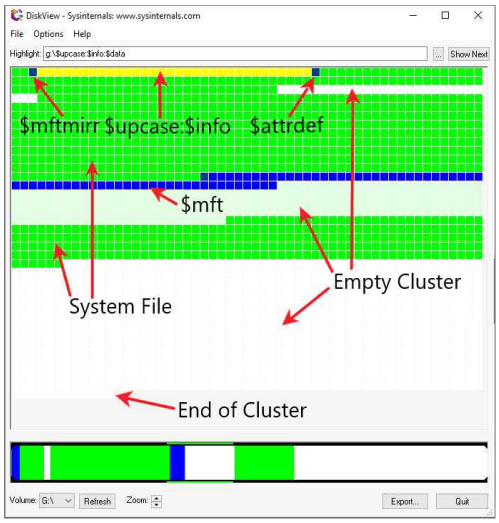
c:\>fsutil fsinfo ntfsinfo g:

NTFS Version	:	3.1
Total Sectors	:	16,383
Total Clusters	:	2,047
Free Clusters	:	1,077
Total Reserved Clusters	:	0
Reserved For Storage Reserve	:	0
Bytes Per Sector	:	512
Bytes Per Physical Sector	:	4096
Bytes Per Cluster	:	4096
Bytes Per FileRecord Segment	:	1024
Clusters Per FileRecord Segment	:	0
Mft Valid Data Length	:	256.00KB
Mft Start Lcn	:	0x00000000000002a
Mft2 Start Lcn	:	0x0000000000000002
Mft Zone Start	:	0x00000000000002a0
Mft Zone End	:	0x00000000000003c0
MFT Zone Size	:	1.13 MB

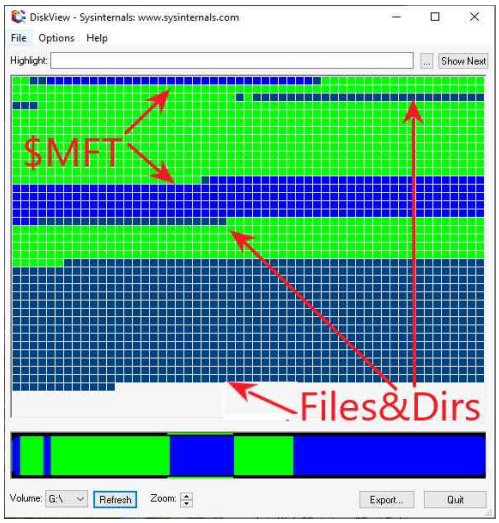
첫 번째 실험은 파일 당 한 개의 클러스터에 할당되도록 구성한 실험이다. 디렉토리 안의 파일명이나 디렉토리명의 목록의 수가 영향을 주기 때문에 이것에 의하여 클러스터가 할당된다. 다음과 같이 리프 디렉토리의 수N_{dir}와 그 안에 들어갈 파일의 수 N_{leaf}를 계산하고 전체 클러스터에 할당되는 수 N_{cls}를 다음과 같이 계산할 수 있다.

파일명의 구성(길이): dataHide-○○.txt (15자)
루트 디렉토리의 수: N_{root}=1
한 디렉토리 내의 최대 파일 수: N_{idx} = 35
리프 디렉토리 내에 생성되는 전체 파일 수: N_{leaf}
전체 클러스터에 할당되는 수는 $N_{cls}=N_{leaf} + N_{dir}$

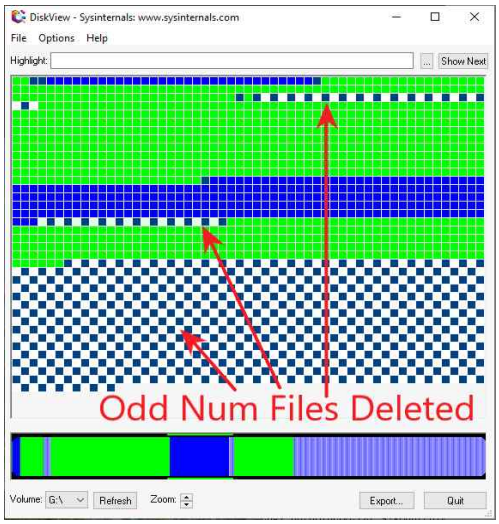
N_{cls}=1,077
$d = \lceil \log_{Nidx} N_{cls} \rceil = \lceil \log_{35} 1077 \rceil = 2$



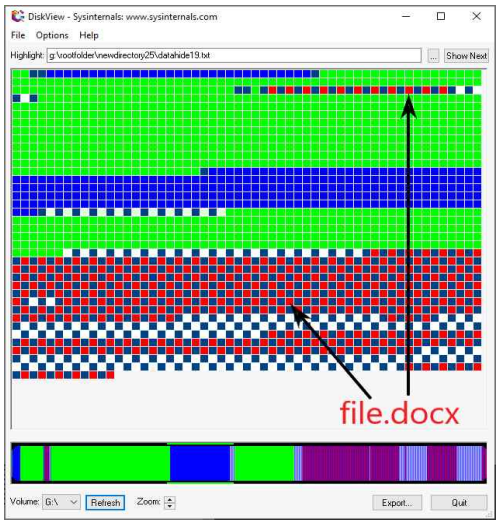
<그림 4 (a)> 포맷직후 빈 디스크



<그림 4 (b)> 디렉토리와 파일생성 후



<그림 4 (c)> 홀수번 파일 삭제 후



<그림 4 (d)> 파일 복사 후 조각화 상태

$d_{dir} = d-1 = 1$

$N_{dir} = N_{root} + N_{idx}^{ddir} = 1 + 35^{1} = 36$

$N_{leaf} = INT((N_{cls} - N_{dir})/N_{idx})*N_{idx}$

$+ (N_{cls} - N_{dir}) \bmod N_{idx}$

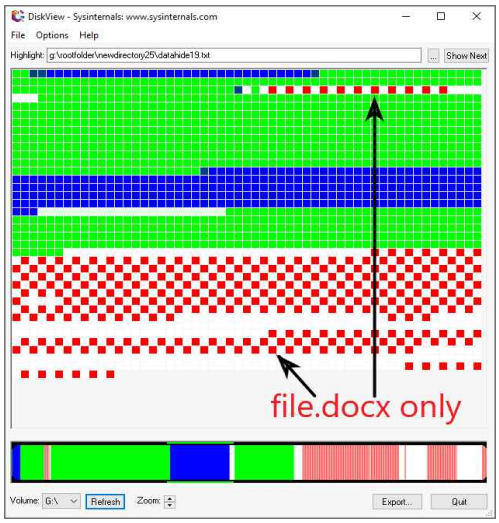
$= INT((1077-36)/35) + (1077-36) \bmod 35$

$= 1,015 + 27 = 1,042$

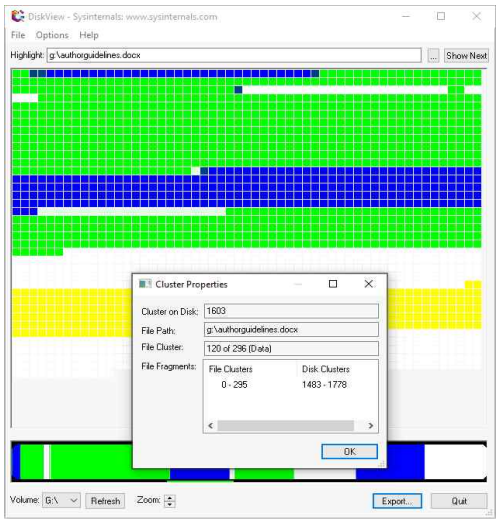
5.2 최대의 조각모음 실험

이 실험에서는 NTFS 파일시스템으로 포맷한 디스크에 파일을 복사하여 최대한으로 단편화한 후에 디스크조각모음을 수행하려고 한다.

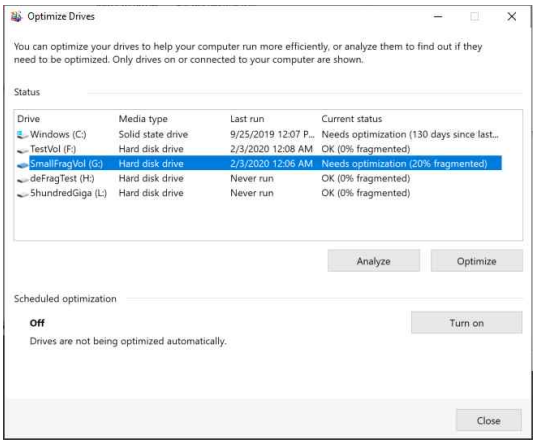
<그림 4(a)>는 DiskView.exe 도구로 표시된 디스



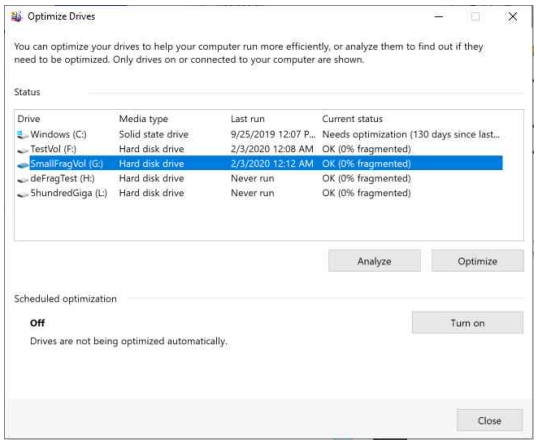
<그림 4 (e)> 짝수 파일과 디렉토리 삭제 후 단편화 상태



<그림 4 (f)> 디스크조각 모음 후의 상태



<그림 4 (g)> 디스크조각 모음 전의 단편화 정도: 20%

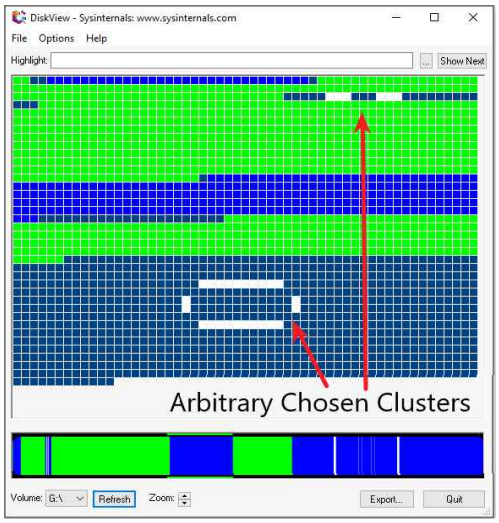


<그림 4 (h)> 디스크조각 모음 후의 단편화 정도: 0%

크 포맷직후의 디스크 상태를 나타내고 있다. 작은 용량으로 파티션으로 하였기 때문에 흰 부분으로 표시된 "Empty Cluster"영역이다. 초록색 부분은 "System File"부분이다. 앞쪽에는 연속적으로 "$mftmirr", "$upcase:$info", "$attrdef"가 구성된다. 파란색으로 표시된 "$mft" 부분은 MFT엔트리가 저장되는 부분으로 포맷직후에는 <그림4 (a)>과 같은 형태 이지만 파일이나 디렉토리가 늘어나면 이 영역은 증가한다.

<그림 4(b)>는 디스크 용량의 한계까지 파일과 디렉토리를 최대로 생성한 후의 디스크 클러스터 점유상태를 나타내고 있다. "Files&Dirs"는 생성된 디렉토리와 파일의 점유상태를 나타내고, "$MFT"는 이에 따라 증가된 $MFT 영역을 나타내고 있다.

<그림 4(c)>는 리프디렉토리 내에 생성한 파일들 중에서 홀수번 파일들을 삭제한 후의 상태를 나타낸



<그림 5> 임의로 선택된 클러스터의 해제

것이다. 파일마다 한 개의 클러스터를 점유하고 있었기 때문에 삭제 후의 모습에서 삭제되지 않은 짝수번 파일들과 삭제된 빈 클러스터 공간들이 교차적으로 표시되고 있다.

<그림 4(d)>는 1.15MB의 file.docx 파일을 복사한 후의 조각화 상태를 나타내고 있다. <그림 4(e)>는 짝수파일과 관련 디렉토리들을 삭제한 후에 file.docx 파일만 남은 조각화 상태를 나타내고 있다. <그림 4(f)>는 조각화된 file.docx에 대하여 디스크 조각모음을 수행하고 난 뒤의 상태를 나타내고 있다. 296개의 클러스터에 연속으로 저장된 것을 확인할 수 있다. <그림 4(g)>는 디스크 조각모음을 실행하기 전에 단편화정도가 20%로 계산된다. <그림 4(h)>는 디스크 조각모음을 실행 후에 단편화정도는 0%가 된 것이다.

5.3 임의로 선택한 클러스터에 대한 조작

제안한 방법으로 임의의 원하는 클러스터의 할당을 해제하는 방법은 디렉토리 안에 만들어진 개별의 파일을 삭제하는 것으로 수행된다. 파일마다 할당된 정보를 얻기 위해서 Sysinternal의 DiskView.exe 도구를 사용한다. 이 도구에서 클러스터 위치를 더블클릭하면 할당된 클러스터의 경우는 클러스터의 위치에 대한 기본적인 정보를 얻을 수 있다.

특정 위치에 대한 선택은 생성된 파일의 번호를 통하여 가능하다. 한 개의 파일당 한 개의 클러스터를 차지하도록 구현되어 있기 때문에 특정 위치에 대한 선택은 파일의 번호를 통하여 가능하다. <그림 5>는 임의로 선택된 부분의 할당을 해제하는 과정을 나타낸 것인다. 특정 위치에 연속된 클러스터를 선택할 수 있고 규칙적이거나 반복된 위치에 대하여 선택할 수 있다.

VI. Conclusions

이 논문에서는 NTFS로 포맷된 디스크 장치에서 어떤 클러스터의 사용상태를 임의로 선택하여 할당하거나 해제할 수 있는 방법을 제안하였다. 이 방법에서의 가장 큰 특징은 파일의 생성, 복사, 삭제의 가장 간단한 형태의 파일 명령에 의하여 제어할 수 있다는 점이다. 특정 클러스터 위치에 어떤 파일을 할당하거나 해제하는 것은 매우 어려운 과정이지만 제안한 방법으로 특별한 도구없이 구현할 수 있게 되었다. 제안한 방법에서 발생하는 문제 3가지에 대한 해결방안을 제시하여 제안한 방법을 구현하는데 있어서의 난점을 해결하였다.

디스크 조각모음의 실험의 사례의 구현을 통해서 제안한 방법의 효과적으로 구현됨을 입증하였다. 또한 임의의 위치에 대한 클러스터의 사용상태의 해제의 예시를 통하여 다양한 사례에 적용할 가능성을 제시하였다.

제안한 방법에서는 기본적인 파일시스템이 점유하고 있는 위치를 변경할 수 없다는 점과 디스크의 파티션의 크기에 따라 첫 번째 파일 저장되는 위치가

달라진다는 점들이 난점으로 존재한다. 이것에 대해서는 “fsutil fsinfo ntfsinfo”에서 정보를 얻을 수 있기 때문에 그 정보를 이용하여 해결할 가능성이 있다. 또한, 디스크 뷰(DiskView.exe)의 시각적인 기능을 활용하여 클러스터의 정보를 얻는 방법이 효과적으로 사용할 수 있다.

추후과제로써 디스크 뷰와 같은 GUI 도구를 제작하는 것이 필요하다. 디스크 상에 배치된 모든 파일들에 대하여 “클러스터 번호”, “파일의 경로”, “파일에 구성된 클러스터의 번호”, “파일의 단편화된 정보” 등을 제공하고 있다. 이 도구는 단지 디스크 상태를 보이기 위한 기능만을 갖고 있다. 이 논문에서 제안한 방법을 바탕으로 클러스터에 대한 상태를 변경하기 위한 기능을 제작하여 클러스터에 대해서 직접적으로 할당을 위한 쓰기와 해제를 위한 지우기 기능을 구현하는 것을 수행할 필요가 있다.

Acknowledgment

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (NRF-2019R1F1A1058902)

참고문헌

[1] Wikipedia, File System Fragmentation, https://en.wikipedia.org/wiki/File_system_fragmentation

[2] Microsoft, Inside Windows NT Disk Defragmenting, http://mirrors.arcadecontrols.om/www.sysinternals.com/Information/DiskDefragmenting.html

[3] Microsoft Drive Optimizer, https://en. ikipedia.rg/wiki/Microsoft_rive_ptimizer

[4] Conlan, K., Baggili, I. and Breitinger, F., “Anti-forensics: Furthering digital forensic science through a new extended, granular taxonomy,” Digital Investigation, Vol. 18, 2016, pp. S66-S75.

[5] Hassan, N. A., and Hijazi, R., Data Hiding Techniques in Windows OS, Elsevier, 2017.

[6] Cho, G.-S., “A New NTFS Anti-Forensic Technique for NTFS Index Entry,” The J. of Korea Institute of Information, Electronics, and Communication Technology, Vol.8 No.4, 2015, pp. 327-337.

[7] Berghel, H., Hoelzer, D., and Sthultz, M., “Data Hiding Tactics for Windows and Unix File Systems,” Advances in Computers, vol. 74, 2008, pp. 1-17.

[8] Huebner, E., Bem, D. and Wee, C. K., “Data Hiding in the NTFS File System,” Digital Investigation, Vol. 3, Issue 4, 2006, pp. 211-226.

[9] Karresand, M., Axelsson, S., and Dyrkolbotn, G., "Using NTFS Cluster Allocation Behavior to Find the Location of User Data," Digital Investigation, Vol. 29, 2019, pp. S51-S60.

[10] Bahjat, A. and Jones, J., “Deleted file fragment dating by analysis of allocated neighbors,” Digital Investigation 28, 2019, pp. S60-S67.

[11] Carrier, B., File System Forensic Analysis, Addison-Wesley, 2005.

[12] Cho, G. S., “A Maximum Data Allocation Rule for Anti-forensic Data Hiding Method in NTFS Index Record,” Int. J. of Internet, Broadcasting and Communication, Vol.9, No.3, 2017, pp. 17-26.

■ 저자소개 ■

조 규 상
Cho, Gyu-Sang

1996년 3월~현재
동양대학교 컴퓨터학과 교수
1997년 2월 한양대학교 전자공학(공학박사)
1989년 2월 한양대학교 전자공학(공학석사)
1986년 2월 한양대학교 전자공학(공학사)

관심분야 : 디지털포렌식, 데이터하이딩, 인공지능보안
E-mail : cho@dyu.ac.kr

논문접수일 : 2020년 4월 13일
수 정 일 : 2020년 5월 2일
게재확정일 : 2020년 5월 12일