(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2010-0115057
(43) 공개일자 2010년10월27일

(51) Int. Cl.
G06F 12/00 (2006.01) *G06F 9/06* (2006.01)
(21) 출원번호 10-2009-0033562
(22) 출원일자 2009년04월17일
심사청구일자 2009년04월17일

(71) 출원인
고려대학교 산학협력단
서울 성북구 안암동5가 1
(72) 발명자
유혁
서울특별시 강남구 수서동 738 동익아파트 801-101
박현찬
서울특별시 성북구 안암동2가 121-19호 2층
(74) 대리인
나승택, 조영현

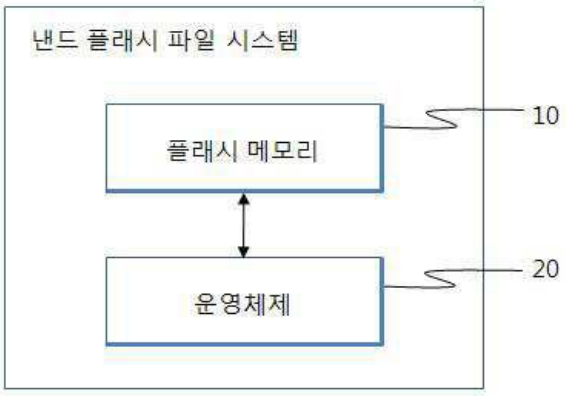
전체 청구항 수 : 총 11 항

(54) 낸드 플래시 파일 시스템

(57) 요 약

본 발명은 낸드 플래시 파일 시스템에 관한 것으로, 더욱 상세하게는 갑작스런 전원의 유실, 시스템 에러 등으로 정상적인 종료 동작을 수행하지 못했을 경우 발생하는 플래시 상태 시 플래시 메모리 공간 분할 기법을 통해 빠른 시간 내에 복구할 수 있는 낸드 플래시 파일 시스템에 관한 것이다.

대 표 도 - 도1a



특허청구의 범위

청구항 1

다수의 블록으로 구분된 플래시 메모리와;

상기 블록의 작업 영역을 다수 개로 세분화하여 할당하되, 현재 작업 중인 영역 정보를 기록 관리하는 운영체제를 포함하는 낸드 플래시 파일 시스템.

청구항 2

제 1항에 있어서,

상기 블록은

시스템 설정 정보를 기록한 파일 시스템 설정값과 저널 블록의 메타 데이터를 기록한 저널블록테이블을 저장하는 슈퍼 블록과;

작업 영역 내에 있는 저널정보를 기록하는 저널 블록과;

데이터와 아이노드가 기록되는 데이터 블록 및;

현재 사용되고 있지 않은 프리 블록을 포함하는 것을 특징으로 하는 낸드 플래시 파일 시스템.

청구항 3

제 2항에 있어서,

상기 모든 블록 내 페이지의 여분 영역(spare area)에

에러교정코드(Error Correction Code)와 버전이 저장되는 것을 특징으로 하는 낸드 플래시 파일 시스템.

청구항 4

제 2항에 있어서,

상기 저널블록테이블(JBT)은

1개 이상의 페이지로 구성되고, 각 페이지의 여분 영역에는 현재의 페이지에 포함된 저널 블록의 개수, 언마운트 정상 수행 여부, 현재의 작업 영역의 시작 블록 번호, 다음 사용될 작업 영역의 시작 블록 번호 중 적어도 하나를 포함하여 기록되는 것을 특징으로 하는 낸드 플래시 파일 시스템.

청구항 5

제 2항에 있어서,

상기 슈퍼블록은

복수 개의 블록으로 구성되고,

슈퍼 블록 데이터의 유실을 막고 유실 시 복구를 위한 복제블록과;

여분의 데이터를 위한 예비블록을 포함하는 것을 특징으로 하는 낸드 플래시 파일 시스템.

청구항 6

제 2항에 있어서,

상기 저널 블록은

저널들이 메인 메모리에 캐싱되고 캐쉬 내의 저널들의 크기가 한 페이지 크기에 도달하는 경우 저장되되, 작업 영역 외부의 아이노드에 대해 삭제가 이루어지면 해당 정보의 저널과 함께 그때까지 캐쉬에 저장된 저널들이 즉각 저장되는 것을 특징으로 하는 낸드 플래시 파일 시스템.

청구항 7

제 1항 내지 제 6항 중 선택된 어느 하나의 항에 있어서,

상기 운영체제는

세분된 다수의 작업영역을 설정하고 관리하는 작업영역 제어모듈과;

파일 시스템의 초기화를 수행하는 초기화 모듈과;

상기 초기화 수행시 크래시가 발생한 경우 크래시를 복구하는 크래시 복구 모듈을 포함하는 것을 특징으로 하는 낸드 플래시 파일 시스템.

청구항 8

제 7항에 있어서,

상기 작업영역 제어모듈은

세분된 다수의 작업영역을 설정하고, 캐쉬에 한 페이지 분량의 저널이 차거나 작업 영역 외부에서 삭제된 아이노드가 있는 경우 또는 현재 작업 영역에 할당된 저널 블록이 남아있지 않고 추가 할당에 실패한 경우 작업영역을 순차적으로 이동하고, 다음 작업영역으로 이동시 저널이 기록된 페이지의 여분 영역에 다음 작업영역의 시작 블록 번호를 기록하는 것을 특징으로 하는 낸드 플래시 파일 시스템.

청구항 9

제 7항에 있어서,

상기 작업영역 제어모듈은

블록 내에 포함된 모든 페이지가 불필요한 데이터를 기록하고 있는 경우 가비지 콜렉션(garbage collection)에 의해 저널 블록과 데이터 블록에 대한 삭제 연산을 수행하되,

슈퍼블록의 저널 블록 테이블 정보를 다음 슈퍼 블록에 기록한 뒤 삭제 연산을 수행하는 것을 특징으로 하는 낸드 플래시 파일 시스템.

청구항 10

제 7항에 있어서,

상기 초기화 모듈은

전체 슈퍼 블록 각각에 대해 첫 번째 페이지의 여분 영역에 저장된 정보를 통해 유효한 슈퍼블록을 추출하고, 상기 유효한 슈퍼블록 중 가장 최신의 저널 블록 테이블을 구비한 슈퍼 블록을 추출하여 언마운트가 정확히 수행된 경우 상기 가장 최신의 저널 블록 테이블을 이용해 전체 저널을 추적하여 초기화를 수행하는 것을 특징으로 하는 낸드 플래시 파일 시스템.

청구항 11

제 7항에 있어서,

상기 크래시 복구 모듈은

언마운트가 정확히 수행되지 않은 경우 크래시가 발생한 것으로 판단하고, 가장 최신의 저널 블록 테이블을 기반으로 저널 정보를 읽고 버전 정보를 이용하여 가장 마지막으로 기록된 저널을 찾아서 해당 저널이 기록된 페이지의 여분 영역에 기록된 작업 영역 정보를 통해 가장 최신의 작업 영역을 파악하고, 작업 영역 내에서 변경되었지만 저널에 기록되지 않은 블록들을 순차적으로 검색하면서 메타데이터 정보를 복구하는 것을 특징으로 하는 낸드 플래시 파일 시스템.

명 세 서

발명의 상세한 설명

기 술 분 야

[0001] 본 발명은 낸드 플래시 파일 시스템에 관한 것으로, 더욱 상세하게는 갑작스런 전원의 유실, 시스템 에러 등으로 정상적인 종료 동작을 수행하지 못했을 경우 발생하는 플래시 상태 시 플래시 메모리 공간 분할 기법을 통해 빠른 시간 내에 복구할 수 있는 낸드 플래시 파일 시스템에 관한 것이다.

배 경 기 술

[0002] 낸드 플래시 메모리는 비휘발성 특성과 빠른 속도, 작은 크기 등의 이유로 휴대용 전자기기의 저장매체로 널리 사용되고 있다.

[0003] 낸드 플래시 파일 시스템은 낸드 플래시 메모리가 내장된 기기에서 운영체제가 낸드 플래시를 저장 공간으로 운용하기 위한 시스템이다. 이러한 파일 시스템은 낸드 플래시의 지움(erase) 횟수 제한, 덮어쓰기(overwrite) 불가능 등의 특징으로 인해 하드 디스크 드라이브를 대상으로 하는 파일 시스템과 다른 특징을 갖는다.

[0004] 그 중 중요한 특징 중 하나는 파일 시스템의 초기화이다. 파일 시스템을 사용하기 위해서는 실제 파일 시스템 자체가 요구하는 정보를 저장 매체에서 불러오는 동작을 수행하여야하고 이를 플래시 파일 시스템에서의 초기화라고 부른다.

[0005] 초기화 과정에서는 파일 시스템의 메타데이터를 읽어 시스템의 메인 메모리에 파일 시스템을 사용하기 위한 데이터를 구축한다.

[0006] 그런데 만약 파일 시스템이 이전에 비정상적으로 종료된 경우 메타데이터가 유실되거나 기록된 메타데이터와 실제 데이터가 불일치할 수 있다. 이러한 상황을 파일 시스템의 무결성(intergrity)이 깨졌다고 하며 이것을 크래시(crash) 상황이라고 부른다.

[0007] 따라서, 초기화 과정에서는 크래시 상황을 판별할 수 있어야 하며, 무결성 상태로 회복하기 위한 크래시 복구(crash recovery) 기법이 필요하다.

[0008] 일반적으로 JEFS(Journaling Flash File System version)나 YAFFS(Yet Another Flash File System)와 같은 로그 기반 구조의 파일 시스템은 모든 메타데이터가 데이터와 함께 메모리 공간 전체에 분산되어 저장된 로그 기반 구조를 사용하므로 정상적인 초기화 과정에서도 모든 메타데이터를 찾아내기 위해 플래시 전체 공간에 대한 읽기 동작이 수행되기 때문에 특별한 크래시 복구 기법을 가지지 않는다. 이러한 방식은 크래시 상황에서도 특별한 부가적인 복구 기법 없이 초기화를 정상적으로 종료할 수 있지만, 수행 시간이 매우 오래 걸리는 단점이 있다.

[0009] 상기와 같은 단점을 해결하기 위한 빠른 초기화를 기법이 "A fast start-up technique for flash memory based computing systems"의 논문(저자:KS Yim, J Kim, K Koh, 출판:2005년 ACM Symposium on Applied computing 학술대회의 프로시딩)에서 스냅샷(snapshot) 기법이라는 이름으로 제안된 바 있다.

[0010] 상기 논문에서 제시된 기술에 따르면, 스냅샷 기법은 파일 시스템이 종료되는 시점에 메인 메모리 내에 저장되어 있는 파일 시스템 메타 데이터 구조를 플래시 메모리의 미리 약속된 구역에 저장하고, 초기화 시점에 해당 구역만을 읽어들여 초기화를 수행한다. 이 기법은 빠른 초기화 속도를 보이지만, 크래시 상황에서 크래시 복구 수행시간이 명확하게 제시하고 있지 않다. 즉, 고속 결함 복구 기법을 제시하여 기존 플래시 파일 시스템에서의 초기화 기법을 최적화하였지만, 이는 기존 YAFFS에서 읽기 명령의 수행 횟수를 줄이기 위해 사용되는 방법과 다르지 않으며 크래시가 발생한 상황에는 YAFFS와 유사한 수행시간을 보일 것으로 예측할 수 있다.

[0011] 또한, "The Design of efficient initialization and crash recovery for log-based file systems over flash memory"의 논문(저자: Wu, C.H., T.W. Kuo, L.P. Chang 출판: 2006년 ACM Transactions on Storage 학술대회의 프로시딩)에서 제안된 로그 기반의 기법은 파일에 일어나는 쓰기/지움 연산을 모두 로그로 기록하고 초기화 시점에 로그만을 읽어들여 파일 시스템의 메타데이터를 재구축하여 초기화를 효율적으로 수행한다. 여기서, 로그는 파일 시스템의 사용에 따라 계속해서 생겨나기 때문에 특정 블록을 예약해서 사용하지 않고, 앞쪽의 몇 개 블록을 할당해두고 로그를 기록하기 위해 수시로 할당되는 로그 세그먼트 블록들의 번호를 기록하는 방식을 사용한다. 각 로그는 로그 세그먼트 블록 내에 페이지 단위로 기록되며 로그 세그먼트는 임의의 블록에 수시로 할당된다. 그리고 메모리 가장 앞쪽 공간에 로그 세그먼트 디렉토리를 할당하고 이에 로그 세그먼트들의 할당 및 해제 정보를 저장한다.

[0012] 그런데 종래에 개시된 기술은 다음과 같은 문제점들을 가지고 있다.

[0013] 첫째, 스냅샷 기법의 경우 파일 시스템이 정상적으로 종료된 경우 약속된 위치에 저장된 메타데이터들을 바로 읽어와서 초기화를 수행할 수 있기 때문에 매우 효율적인 성능을 보인다. 그러나 크래시가 발생한 상황에서는 JFFS나 YAFFS가 동일하게 거의 전체 플래시 메모리 영역에 대한 읽기를 수행하여야한다. 이때의 초기화 수행성능을 빅오(O) 복잡도 표기 기법으로 표현하면 플래시 메모리의 용량을 n이라 할 때 O(n)으로 정리할 수 있으며, 이것은 플래시 메모리의 용량이 증가하면 할수록 초기화 수행 성능도 선형적으로 비례하여 증가함을 의미한다.

[0014] 이는 최근 플래시 메모리의 사용이 급격하게 증가하면서 용량의 발전 속도 또한, 크게 증가하고 있다는 점을 감안할 때, 플래시 파일 시스템의 초기화 속도가 미래에 플래시 메모리 활용에 있어서 큰 문제가 될 수 있음을 알 수 있다.

[0015] 둘째, 로그 기반 기법의 경우 초기화에서는 스냅샷 기법과 유사한 효율적인 성능을 보이지만 로그를 페이지 단위로 기록하여야 하는 한계로 인해 페이지 용량만큼 로그가 생겨날 때까지 캐싱해두기 때문에 언마운트가 제대로 이루어지지 않으면 캐싱된 정보가 소실되며 이로 인한 크래시 현상을 복구해주어야 한다.

[0016] 이때 이미 기록된 로그 정보를 활용할 수 있기 때문에 스냅샷보다는 오류복구에 걸리는 시간이 더 작을 것으로 예상되지만, 메타데이터가 유실된 데이터가 어느 공간에 위치하고 있는지 알 수 없기 때문에 플래시 메모리 내의 사용되지 않은 공간을 모두 검색하여야 한다. 따라서, 이 기법 또한 플래시 메모리의 용량에 따라 크래시 복구 수행시간이 O(n)의 형태로 선형적으로 증가할 수 있다.

[0017] 상기와 같이 현재 제안된 플래시 파일 시스템들은 공통적으로 크래시 복구의 수행시간이 플래시 메모리의 용량에 따라 선형적으로 비례하여 증가하므로 플래시 메모리를 사용하는 기기들의 초기화 수행 시간을 증가시키는 문제점으로 작용하며, 이로 인해 대용량 플래시 메모리에 적용시 문제가 될 수 있다.

[0018]

발명의 내용

해결 하고자하는 과제

[0019] 상기와 같은 문제점을 해결하기 위해 안출된 것으로서 본 발명의 목적은 특정 시점에는 플래시 메모리의 한정된 작업 영역 내에서만 쓰기 작업이 이루어지도록 구성하여 비정상적인 종료로 인한 크래시 발생시 낸드 플래시 용량에 무관하게 빠른 복구가 가능한 낸드 플래시 파일 시스템을 제공하는 데 있다.

과제 해결수단

[0020] 도 1a는 본 발명의 바람직한 실시예에 따른 시스템 구성도이고, 도 1b는 도 1a의 상세 블록도이다.

[0021] 도 1을 참조하면, 본 발명의 낸드 플래시 파일 시스템은 다수의 블록으로 구분된 플래시 메모리(10)와 상기 블록의 작업 영역을 다수 개로 세분화하여 할당하되, 현재 작업 중인 영역 정보를 기록 관리하는 운영체제(20)를 포함하는 것을 특징으로 한다.

[0022] 여기서, 상기 플래시 메모리의 블록은 시스템 설정 정보를 기록한 파일 시스템 설정 값과 저널 블록의 메타 데이터를 기록한 저널블록테이블을 저장하는 슈퍼 블록(110)과 작업 영역 내에 있는 저널정보를 기록하는 저널 블록(120)과 데이터와 아이노드가 기록되는 데이터 블록(130) 및 현재 사용되고 있지 않은 프리 블록(140)을 포함하는 것을 특징으로 한다.

[0023] 그리고 상기 모든 블록 내 페이지의 여분 영역(Spare area)에 에러교정코드(Error Correction Code)와 버전이 저장되는 것을 특징으로 한다.

[0024] 또한, 상기 저널블록테이블(JBT)은 1개 이상의 페이지로 구성되고, 각 페이지의 여분 영역에는 현재의 페이지에 포함된 저널 블록의 개수, 언마운트 정상 수행 여부, 현재의 작업 영역의 시작 블록 번호, 다음 사용될 작업 영역의 시작 블록 번호 중 적어도 하나를 포함하여 기록되는 것을 특징으로 한다.

[0025] 그리고 상기 슈퍼블록(110)은 복수 개의 블록으로 구성되고, 슈퍼 블록 데이터의 유실을 막고 유실 시 복구를 위한 복제블록과 여분의 데이터를 위한 예비블록을 포함하는 것을 특징으로 한다.

[0026] 또한, 상기 저널 블록(120)은 저널들이 메인 메모리에 캐싱되고 캐쉬 내의 저널들의 크기가 한 페이지 크기에 도달하는 경우 저장되되, 작업 영역 외부의 아이노드에 대해 삭제가 이루어지면 해당 정보의 저널과 함께 그때까지 캐쉬에 저장된 저널들이 즉각 저장되는 것을 특징으로 한다.

[0027] 상기 슈퍼블록(110)은 모든 작업 영역에 포함되는 저널블록의 메타데이터를 저장 관리하는 블록으로 전체 작업 영역에 하나(슈퍼블록이 복수 개로 구성될 경우 블록은 복수 개로 구성될 수 있음)만 존재하고, 각 작업 영역에는 저널블록(120), 데이터 블록(130) 및 프리 블록(140)이 혼합되어 존재할 수 있다.

[0028] 한편, 상기 운영체제(20)는 세분된 다수의 작업영역을 설정하고 관리하는 작업영역 제어모듈(210)과 파일 시스템의 초기화를 수행하는 초기화 모듈(220)과 상기 초기화 수행시 크래시가 발생한 경우 크래시를 복구하는 크래시 복구 모듈(230)을 포함하는 것을 특징으로 한다.

[0029] 여기서, 상기 작업영역 제어모듈(210)은 세분된 다수의 작업영역을 설정하고, 캐쉬에 한 페이지 분량의 저널이 차거나 작업 영역 외부에서 삭제된 아이노드가 있는 경우 또는 현재 작업 영역에 할당된 저널 블록이 남아있지 않고 추가 할당에 실패한 경우 작업영역을 순차적으로 이동하고, 다음 작업영역으로 이동시 저널이 기록된 페이지 여분 영역에 다음 작업영역의 시작 블록 번호를 기록하는 것을 특징으로 한다.

[0030] 그리고 상기 작업영역 제어모듈(210)은 블록 내에 포함된 모든 페이지가 불필요한 데이터를 기록하고 있는 경우 가비지 콜렉션(garbage collection)에 의해 저널 블록과 데이터 블록에 대한 삭제 연산을 수행하되, 슈퍼블록의 저널 블록 테이블 정보를 다음 슈퍼 블록에 기록한 뒤 삭제 연산을 수행하는 것을 특징으로 한다.

[0031] 또한, 상기 초기화 모듈(220)은 전체 슈퍼 블록 각각에 대해 첫 번째 페이지의 여분 영역에 저장된 정보를 통해 유효한 슈퍼블록을 추출하고, 상기 유효한 슈퍼블록 중 가장 최신의 저널 블록 테이블을 구비한 슈퍼 블록을 추출하여 언마운트가 정확히 수행된 경우 상기 가장 최신의 저널 블록 테이블을 이용해 전체 저널을 추적하여 초기화를 수행하는 것을 특징으로 한다.

[0032] 즉, 슈퍼 블록이 복수 개로 존재할 경우 각 블록의 첫 번째 페이지의 여분 영역에 저장된 에러교정코드(ECC)가 유효한지 여부에 따라 유효한 슈퍼블록을 추출하고, 상기 추출된 유효 슈퍼블록 중 버전 정보에 따라 가장 최신의 저널 블록 테이블을 구비한 슈퍼블록을 추출하게 된다.

[0033] 그리고 상기 크래시 복구 모듈(230)은 언마운트가 정확히 수행되지 않은 경우 크래시가 발생한 것으로 판단하고, 가장 최신의 저널 블록 테이블을 기반으로 저널 정보를 읽고 버전 정보를 이용하여 가장 마지막으로 기록된 저널을 찾아서 해당 저널의 여분 영역에 기록된 작업 영역 정보를 통해 가장 최신의 작업 영역을 파악하고, 작업 영역 내에서 변경되었지만 저널에 기록되지 않은 블록들을 순차적으로 검색하면서 메타데이터 정보를 복구하는 것을 특징으로 한다.

효 과

[0034] 본 발명을 통해 기대할 수 있는 효과는 플래시 파일 시스템의 크래시 복구 성능의 향상이다.

[0035] 특히, 크래시 복구에 소요되는 시간이 플래시 메모리의 용량에 비례하던 기존 기법의 한계를 극복하고 이에 비례하지 않는 수행시간을 갖는 기법을 제안함으로써 앞으로 플래시 메모리의 용량이 증가하더라도 크래시 복구 수행 시간은 거의 증가하지 않는 탁월한 효과가 발생한다. 이는 실제 플래시 메모리가 장착된 기기를 사용할 때 기기의 초기화 시간이 크래시 복구 수행 성능에 크게 영향을 받지않도록 하는 효과를 가진다.

발명의 실시를 위한 구체적인 내용

[0036] 이하, 본 발명의 바람직한 실시예에 대하여 도면을 참조하여 상세하게 설명하기로 한다.

[0037] 플래시 파일 시스템의 크래시 복구를 효율적으로 수행하기 위해 플래시 메모리의 블록에 대해 작업영역을 설정해 사용하는 기법을 설계하였다.

[0038] 본 발명에서 해결하고자 하는 종래 크래시 복구 기법의 문제점은 전체 플래시 공간에 따라 수행시간이 선형적으로 증가한다는 점이다.

[0039] 이를 해결하기 위해 파일 시스템이 사용하는 블록의 영역을 세분화하여 현재 작업 중인 영역이 어디인지 기록해두는 작업 영역 설정 방식을 이용하여 크래시 복구 수행시간을 매우 작은 상수값으로 한정시키는 기법을 제안한다.

[0040] 예를 들어, 0~9까지 10개의 블록에 대해 현재 작업 중이라면 이 정보를 약속된 공간에 기록하고, 이후 해당 공간을 모두 사용하거나 그 외 설정된 조건이 만족되면 이에 대한 메타데이터를 기록한 후 10~19까지로 작업영역을 이동하고 이 정보를 업데이트 한다.

[0041] 이러한 방식으로 크래시 복구시 탐색해야 하는 공간의 크기를 제한하여 결함 복구 수행시간을 플래시 메모리 용량과 무관하게 단축시킬 수 있다.

[0042] 본 발명은 전체 플래시 메모리에 대해 이루어지는 변경 내용을 하나의 작업 영역 내에 기록하는 구조의 설계, 작업 영역의 할당 및 변이에 관한 정보를 유실하지 않는 설계, 이러한 구조에서의 효율적인 크래시 복구 기법 제공을 특징으로 한다. 이러한 특징에 대해 하기의 설계 내용에서 보다 세부적으로 기술하기로 한다.

[0043] 1) 블록의 분류

[0044] 도 1b를 참조하면, 플래시 메모리 내의 각 블록은 슈퍼블록, 저널 블록, 데이터불록, 프리 블록의 4가지로 구분된다.

[0045] 상기 슈퍼블록은 플래시 메모리의 첫 블록부터 2개 이상의 블록으로 구성되고 파일 시스템 설정값과 저널블록의 메타데이터가 저장된다. 파일 시스템 설정값은 각 블록의 첫 페이지에 기록되며, 현재 파티션의 크기나 첫 번째 블록 번호 등 파일 시스템이 생성될 때 결정되는 정보들이 저장된다. 상기 파일 시스템 설정값은 파일 시스템의 설정에 따라 변경될 수 있다. 다만, 일반적인 파일 시스템 정보들 외에 한 작업 영역에 속하는 블록의 개수, 슈퍼블록의 크기 등은 반드시 포함되어야 한다.

[0046] 그리고 저널블록의 메타데이터는 저널 블록 테이블(Journal Block Table, 이하 "JBT"라고 함)에 저장된다.

[0047] 상기 JBT는 저널 블록들의 번호가 저장된 공간이며 1개 이상의 페이지로 구성되고, 각 페이지의 여분 영역에는 현재의 페이지에 포함된 저널 블록의 개수, 언마운트 정상 수행 여부, 현재의 작업 영역의 시작 블록 번호, 다음 사용될 작업 영역의 시작 블록 번호가 기록된다. 상기와 같은 정보를 통해 JBT의 크기와 언마운트가 정상적으로 수행되었는지 그리고 현재와 다음 작업 영역의 위치를 알 수 있다.

[0048] 상기 저널 블록은 저널이 기록되는 공간이다. 저널은 쓰기/지움 등의 업데이트 정보가 저장되며 현재 작업 영역 내에 있는 저널 블록에 페이지 단위로 기록된다. 저널들은 메인 메모리에 캐싱되며 캐쉬 내에서 중복되는 저널들을 정리하는 동작을 수행한다. 이와 동시에 각 저널들은 저널이 포함된 파일의 메타데이터 구조 내에서 시간 순으로 연결된 링크드 리스트 형태로 유지된다. 캐쉬 내 저널들의 크기가 한 페이지의 크기에 도달하면

실제 저널 블록에 기록된다.

[0049] 크래시가 발생하더라도 작업 영역 내에서 변경이 이루어진 내용은 작업 영역을 스캔함으로써 파악할 수 있지만 다른 영역에 있던 아이노드가 삭제된 정보는 해당 정보가 기록되지 않으면 다시 복구할 수 없으므로 만약 작업 영역 외부의 아이노드에 대해 삭제가 이루어진다면 삭제 정보가 유실되지 않도록 해당 정보의 저널과 함께 그 때까지 캐쉬에 저장되어있던 저널들을 즉각 기록한다.

[0050] 이와 같이 저널을 기록하면 정보의 유실을 막는 효과가 발생하지만 한 페이지 내에 남는 공간이 생겨서 자원의 사용 효율이 떨어질 수 있는 트레이드-오프(trade-off) 관계에 있다. 그리고 저널이 저장되는 각 페이지의 여분 영역에는 다음으로, 저널이 기록될 작업 영역의 번호가 기록된다.

[0051] 그리고 상기 데이터 블록은 데이터와 아이노드가 기록되는 블록이고, 이외의 블록들은 모두 현재 사용되지 않는 프리 블록으로 간주한다.

[0052] 도 2는 본 발명의 바람직한 실시예에 따른 저널 블록을 C언어의 구조체 형태로 도시한 것이다.

[0053] 도 2를 참조하면, _u16은 기호가 붙지않은 정수형(unsigned integer) 형태의 16비트 자료형을 나타내고, _u32는 같은 형태의 32비트 자료형을 나타낸다.

[0054] 한 저널의 크기는 총 16바이트이며, 해당 저널이 표현하는 아이노드의 ID와 한 연산이 시작되는 파일 내의 오프셋, 연산의 사이즈, 연산의 종류를 나타내는 플래그, 그리고 페이지 주소가 기록된다.

[0055] 한 저널이 나타낼 수 있는 최대 크기는 65,535바이트이며, 이는 2048 바이트 크기의 페이지에서 32개 페이지에 해당한다. 프래그의 옵션으로는 파일의 생성, 삭제, 쓰기, 내용 제거(truncate), 0으로 채우기 등이 포함되며 기타 다른 옵션이 추가될 수 있다.

[0056] 이러한 저널이 하나의 페이지에 모이면 저널 페이지 형태가 된다. 도 1은 2048바이트 크기의 페이지를 가정하여 128개의 저널이 포함된 형태를 표현하였고, 페이지 여분 영역에는 저널의 길이, 페이지의 버전, 그 외 에러 교정 코드(ECC, Error Correction Code)가 기록된다.

[0057] 도 3은 본 발명의 바람직한 실시예에 따른 JBT 내의 페이지 형태를 C언어의 구조체 형태로 도시한 것이다.

[0058] 도 3을 참조하면, 2048바이트 크기의 페이지에서 총 512개의 4바이트 주소가 기록되며, 각각은 하나의 저널 블록 주소를 나타낸다. JBT가 기록되는 페이지의 여분 영역에는 현재 페이지에 기록된 저널 블록의 개수, 해당 시점의 작업 영역이 시작되는 페이지 주소, 그리고 다음 작업 영역이 시작되는 페이지 주소, 파일 시스템이 정상적으로 종료되었는지 나타내는 언마운트 플래그 등이 기록된다.

[0059] 2) 블록의 관리

[0060] 상기와 같이 분류된 블록들은 운영체제에 의해 할당되고, 할당되는 규칙은 다음과 같다.

[0061] 슈퍼 블록은 파일 시스템이 생성될 때 정적으로 파티션의 가장 첫 블록부터 할당되며 그 크기는 2의 배수 개의 블록이다.

[0062] 도 4는 본 발명의 바람직한 실시예에 따른 슈퍼 블록의 상세 구성도이다.

[0063] 도 4는 슈퍼 블록이 4개로 할당된 레이아웃을 간략하게 도시한 것으로, 0번 블록은 일반적인 슈퍼 블록이며 2번 블록은 슈퍼 블록 데이터의 유실을 막고 유실시 복구를 위한 복제 블록이다.

[0064] 그리고 1번 3번 블록은 0번 2번 블록에 대한 예비 블록이다. 플래시 메모리의 특성상 이미 데이터가 쓰여진 페이지에는 다시 데이터를 쓸 수 없고 반드시 삭제 연산을 수행하여야 한다.

[0065] 따라서 슈퍼 블록에 데이터를 기록하는 동작시 삭제 연산으로 인해 지연되는 것을 예비 블록을 구비함으로써 방지할 수 있다. 예를 들어, 0번 블록에 모든 데이터가 쓰여졌다면 슈퍼 블록에 대한 쓰기 연산은 1번 블록에 수행된다. 그리고 파일 시스템의 유휴 시간에 0번 블록에 대한 지움 연산을 수행하고 이후에 0번 블록이 예비 블록이 된다.

[0066] 파일 시스템이 초기화되는 시점에는 0번 블록부터 차례대로 블록의 두 번째 페이지를 검사하여 JBT가 발견되는 블록이 현재 사용중인 슈퍼 블록이라 판단할 수 있다. 첫 번째 페이지에는 파일 시스템의 설정 정보를 기록한 파일 시스템 설정값(FS-configurations)이 저장된다.

[0067] 저널 블록은 한 번에 연속된 N개 블록이 할당된다. 블록은 임의의 위치에 있을 수 있고, 모든 저널 블록을 소진한 경우, 추가로 N개의 블록을 현재 작업 영역 내에서 할당하여 저널 블록으로 활용한다.

[0068] 할당이 이루어지면 슈퍼 블록 내의 JBT를 업데이트한다. 한 번에 N개를 할당하는 이유는 슈퍼 블록에 너무 잦은 연산이 이루어지지 않으며, 크래시 상황에서 다음 저널 블록을 쉽게 찾을 수 있도록 하기 위함이다.

[0069] 데이터 블록은 현재 작업 영역 내에서 최대한 순차적으로 할당된다. 만약 작업 영역 내에 충분한 블록이 없다면 해당 시점까지의 저널 정보를 기록하고 다음 작업 영역으로 이동한다.

[0070] 도 5는 본 발명의 바람직한 실시예에 따른 블록들이 할당된 상태를 도시한 것이다. 도 5를 참조하면, 현재 작업 영역의 크기는 6개 블록이며 슈퍼 블록은 JBT를 통해 저널 블록들을 가리키고 있으며 데이터 블록은 각 작업 영역 내에서 순차적으로 할당되고 있음을 알 수 있다. 현재 작업 영역 3까지는 사용되지 않았지만 미리 저널 블록을 할당해둔 상태이다.

[0071] 모든 블록 내 페이지의 여분 영역에는 공통적으로 에러 교정 코드(ECC)와 버전이 기록된다. 상기 에러 교정 코드(ECC)는 플래시 메모리에서 발생하는 1비트 에러를 교정하기 위하여 모든 플래시 기반 저장 시스템에서 공통적으로 사용되는 정보이고, 버전은 페이지 상호 간의 기록된 순서를 알기 위해 1씩 증가시키며 기록하는 전역 정보이다. 이를 통해 어느 페이지가 가장 최신 정보를 기록하고 있는지 알 수 있다.

[0072] 3) 작업 영역의 관리

[0073] 작업 영역은 파일 시스템이 생성될 때 그 크기가 정해지며 순차적으로 이동한다. 크기는 블록의 개수로 지정되며 대략 32~64MB 가량이 되도록 설정한다. 이 크기가 클수록 크래시 복구 소요시간이 길어지므로 플래시 메모리의 사양에 따라 적당한 값을 설정해줄 필요가 있다. 현재 플래시 메모리의 성능을 볼 때 32~64MB 이내라면 크래시 복구 수행 성능이 약 1-2초 이내가 될 것으로 예상된다. 이후 플래시 메모리 성능의 발전에 따라 이 값은 변경될 수 있다.

[0074] 작업 영역의 관리는 운영체제가 수행한다. 작업 영역의 이동은 순차적으로 이루어진다. 예를 들어, 현재 작업 영역이 128번 블록부터 1024개 블록이라면 다음 작업 영역은 1152(=128+1024)번 블록부터 시작한다.

[0075] 이렇게 연속적으로 작업 영역을 이동하게 함으로써 만약 가장 최근의 작업 영역 정보가 유실된다 하더라도 이를 검색하는 수행시간을 최소화시킬 수 있다.

[0076] 이동이 이루어지는 시점은 하나의 저널 페이지가 기록될 때, 즉 캐쉬에 한 페이지 분량의 저널이 차거나 작업 영역 외부에서 삭제된 아이노드가 있을 경우이다.

[0077] 이때 만약 현재 작업 영역으로 이동한 이후 작업 영역 크기의 20% 이상을 사용했다면 다음 작업 영역으로 이동하고 기록될 저널이 기록된 페이지의 여분 영역에 다음 작업 영역의 시작 블록 번호를 기록한다. 이를 통해 저널들을 추적함으로써 가장 최근의 작업 영역을 파악할 수 있다.

[0078] 여기서 20%는 임의의 수치이며 플래시 메모리의 크기, 작업 영역의 크기 등 여러 가지 변수에 따라 변경될 수 있다. 만약 한 작업 영역 내의 블록을 모두 할당하여 사용한 이후에 다음 작업 영역으로 이동한다면 사용자의 이용 패턴에 따라 특정 영역에만 많은 양의 데이터가 기록될 수 있고, 이것은 균등한 지움 횟수를 유지(wear-leveling)하여야 하는 플래시의 특성을 저해할 수 있다.

[0079] 따라서 최대한 플래시 전체 공간에 대해 균일한 I/O가 수행될 수 있도록 하기 위해 작업 영역을 수시로 변경하여 준다. 다만, 작업 영역 변경에 대한 정보를 JBT에 업데이트하는 약간의 오버헤드가 존재하기 때문에 너무 자주 작업 영역을 변경하는 것 또한 성능에 나쁜 영향을 끼칠 수 있다.

[0080] 4) 가비지 콜렉션(garbage collection) 기법

[0081] 저널 블록과 데이터 블록의 해제는 블록 내에 포함된 모든 페이지가 불필요한 데이터를 기록하고 있는 경우에 가비지 콜렉션에 의해 이루어진다. 저널 블록 내의 데이터는 갱신된 정보가 다른 위치에 저장되거나 파일의 삭제로 인해 저널 정보가 더이상 유효하지 않은 경우에 필요없어지고, 데이터 블록은 내부에 포함된 데이터가 삭제 또는 수정된 경우이다.

[0082] 상기와 같은 저널 블록과 데이터 블록들을 수집하고 실제로 삭제하는 가비지 콜렉션은 파일 시스템이 유휴 상태일 때 수행된다. 가비지 콜렉션 대상 블록이 발견되고 해당 정보가 저널에 기록이 완료된 이후 해당 블록에 대해 지움 연산을 수행한다.

[0083] 슈퍼 블록의 경우 JBT 정보는 언제나 갱신되기 때문에 하나의 슈퍼 블록 내의 페이지를 모두 사용하면 다른 슈퍼 블록에 정보를 기록하여야 한다. 이때는 데이터가 유실되는 것을 막기 위해 우선 다음 슈퍼 블록에 JBT 정보를 기록한 이후 이전 슈퍼 블록을 삭제한다. 이러한 동작을 통해 모든 슈퍼 블록들 내에 유효한 슈퍼 블록은 단 하나로 유지된다.

[0084] 5) 초기화 및 크래시 복구 기법

[0085] 이하, 상기와 같이 설계된 작업 영역 기법을 사용한 플래시 파일 시스템에 대한 초기화 기법과 크래시 복구 기법에 대해 도 6을 참조하여 기술하기로 한다.

[0086] 파일 시스템의 초기화는 운영체제의 초기화 모듈에서 수행되고, 가장 먼저 수행되는 것은 유효한 슈퍼 블록과 가장 마지막에 저장된 저널 블록 테이블(JBT)을 찾는 것이다. 이를 위해 우리는 복수 개의 슈퍼 블록을 순차적으로 탐색한다. 각 블록의 첫 번째 페이지는 파일 시스템 정보이므로 해당 페이지의 데이터 영역과 여분 영역(spare area) 모두를 읽어 메인 메모리에 로딩한다. 두 번째 페이지부터는 JBT이므로 순차적으로 각 페이지의 여분 영역을 읽어 언마운트 플랙을 확인한다. 만약 언마운트 플랙이 발견되고, 그 다음 페이지에 아무 데이터가 없다면 그것이 유효한 JBT라 볼 수 있다. 만약 유효한 JBT가 없다면 이는 크래시가 발생한 상황으로, 가장 최신의 버전 값을 저장하고 있는 JBT를 토대로 크래시 복구를 수행하여야 한다.

[0087] ① 정상적 상황의 초기화 기법

[0088] 정상적으로 언마운트가 수행된 경우 초기화 모듈은 가장 최신의 JBT를 이용해 전체 저널을 추적할 수 있다. 저널들은 N개씩 연속적으로 할당되어 있으므로 만약 N개 내에서 사용되지 않은 저널이 있는 경우 생략할 수 있다.

[0089] 저널 내에는 파일 시스템의 업데이트 정보가 모두 기록되어 있기 때문에 전체 저널의 정보를 읽어들이면 전체 파일 시스템 메타 데이터 정보를 구축할 수 있다. 이러한 정상적 상황의 초기화 기법은 로그 기반 기법에서 유효한 슈퍼 블록과 정상적인 언마운트 여부를 파악하는 기법이 없는 단점을 보완할 수 있다.

[0090] ② 크래시 복구 기법

[0091] 만약 크래시가 발생하여 정상적으로 언마운트가 수행되지 않았다면, 운영체제의 크래시 복구 모듈은 먼저 가장 높은 버전이 기록된 가장 최신의 JBT를 기반으로 저널 정보를 읽는다. 이때 더이상 할당된 저널 블록이 없는 것을 확신할 수 있기 때문에 버전 정보를 이용하여 가장 마지막으로 기록된 저널과 작업 영역에 대한 정보를 찾는다.

[0092] 다음으로, 작업 영역 내에서 변경되었지만 저널에 기록되지 않은 정보를 찾고, 저널에 기록되어 있지 않은 블록들을 순차적으로 검색하면서 메타데이터 정보를 복구한다. 한 블록 내의 페이지는 언제나 순차적으로 기록되기 때문에 첫 페이지의 여분 영역을 읽음으로써 블록이 사용 중인지 여부를 알 수 있고 이를 통해 검색을 최적화할 수 있다. 이와 같이 작업 영역 내의 모든 블록 정보를 파악하면 크래시 복구를 종료한다.

[0093] 또한, 본 발명에 따른 기법의 성능을 수행시간 모델로 나타내면 하기의 <수학식 1>과 같이 표현할 수 있다.

수학식 1

[0094] $$T_w = L(T_d + T_s) + B_{fw}T_s + M_f(T_d + T_s) + M_dT_s$$

[0095] 상기 모델에서 T_d와 T_s는 각각 한 페이지 내의 데이터 공간과 여분 영역을 읽는데 걸리는 시간을 의미하고, L은 저널의 전체 개수, M_f는 크래시로 인해 유실된 파일의 개수, M_d는 유실된 데이터 페이지의 개수, B_{fw}는 하나의 작업 영역 내에서 사용되지 않은 블록을 의미한다.

[0096] 분석 결과에서 가장 특징적인 것은 빈 블록을 확인하기 위한 시간을 크게 줄였다는 점이다. 작업 영역의 크기는 플래시 메모리의 용량과 무관하게 거의 고정된 값이므로, B_{fw} 또한 용량에 비례하여 증가하지 않고 한계를

가지는 값이라 추정할 수 있다. 이는 설계에 따라 32~64MB 이하의 값으로 예상된다.

[0097] 따라서, 종래의 로그 기반 기법과 비교할 때 사용되지 않은 블록을 걸러내는데 소용되는 시간이 획기적으로 줄어들게 되고, 이외의 시간은 동일하다고 할 수 있으며 유실된 아이노드와 데이터의 크기는 저널을 저장하기 위한 캐쉬 크기와 비례한다. 단 작업 영역을 관리하기 위해 보다 자주 저널 데이터를 저장하기 때문에 더 좋은 성능을 보일 것으로 예상할 수 있다.

[0098] 이상에서 설명한 본 발명의 상세한 설명에서는 본 발명의 바람직한 실시예를 참조하여 설명하였지만, 본 발명의 보호범위는 상기 실시예에 한정되는 것이 아니며, 해당 기술분야의 통상의 지식을 갖는 자라면 본 발명의 사상 및 기술영역으로부터 벗어나지 않는 범위 내에서 본 발명을 다양하게 수정 및 변경시킬 수 있음을 이해할 수 있을 것이다.

도면의 간단한 설명

[0099] 도 1a는 본 발명의 바람직한 실시예에 따른 시스템 구성도이고, 도 1b는 도 1a의 상세 블록도이다.

[0100] 도 2는 본 발명의 바람직한 실시예에 따른 저널 블록을 C언어의 구조체 형태로 도시한 것이다.

[0101] 도 3은 본 발명의 바람직한 실시예에 따른 JBT 내의 페이지 형태를 C언어의 구조체 형태로 도시한 것이다.

[0102] 도 4는 본 발명의 바람직한 실시예에 따른 슈퍼 블록의 상세 구성도이다.

[0103] 도 5는 본 발명의 바람직한 실시예에 따른 블록들이 할당된 상태를 도시한 것이다.

[0104] 도 6은 본 발명의 바람직한 실시예에 따른 초기화 및 크래시 복구 과정을 개략적으로 도시한 것이다.

[0105] *도면의 주요부분에 대한 부호의 설명*

[0106] 10 : 플래시 메모리 110 : 슈퍼 블록

[0107] 120 : 저널 블록 130 : 데이터 블록

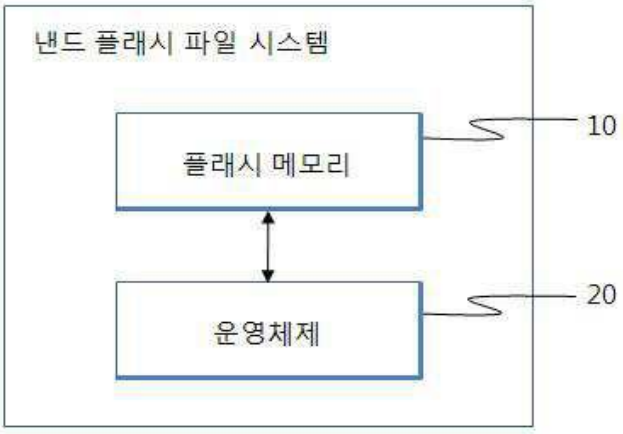
[0108] 140 : 프리 블록 20 : 운영체제

[0109] 210 : 작업영역 제어모듈 220 : 초기화 모듈

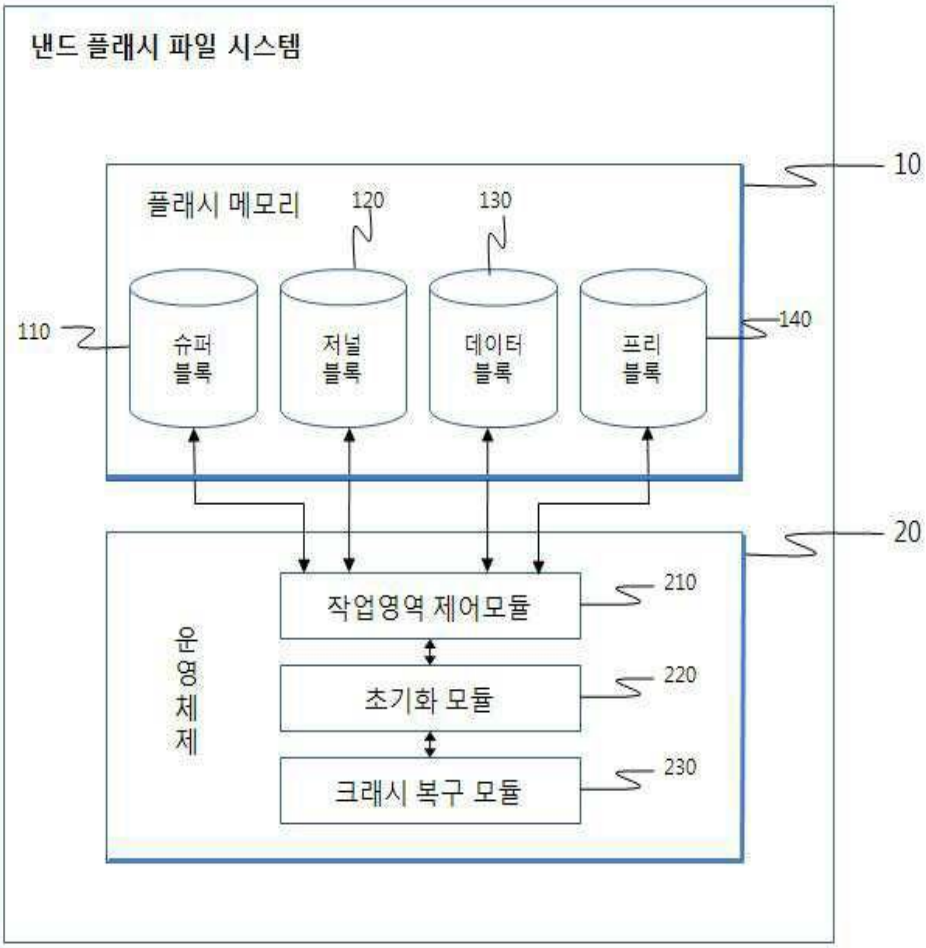
[0110] 230 : 크래시 복구 모듈

도면

도면1a



도면1b


낸드 플래시 파일 시스템
플래시 메모리
120
130
10
110
슈퍼 블록
저널 블록
데이터 블록
프리 블록
140
20
운영 체제
작업영역 제어모듈
210
초기화 모듈
220
크래시 복구 모듈
230

도면2

```
struct journal {
        __u32 inodeId;
        __u32 offset;
        __u16 size;
        __u16 flag;
        __u32 pageAddr;
};

// size range : 0~65535 (32 pages for 2048 bytes page)
// flag options : {create, delete, write, truncate, fill zero, reserved(s)}

struct journalPage {
        struct journal journals[128];  //for 2048 bytes page
};

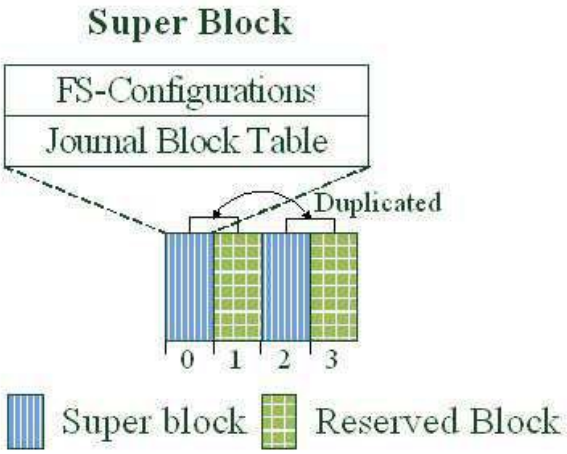
struct journalPageSpare {
        __u32 length;
        __u32 version;
        //... ECCs and file system specific datas
};
```

도면3

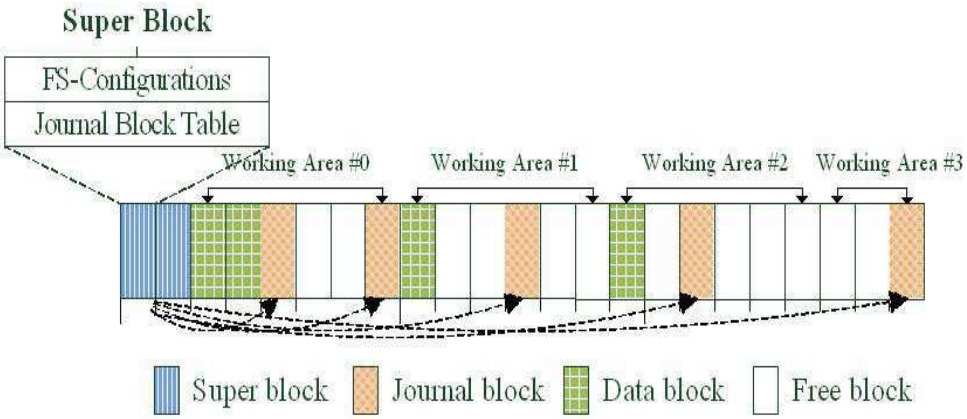
```
struct journalBlockTablePage {                    //for 2048 Bytes page
        __u32 journalBlockAddr[512];
};

struct journalBlockTablePageSpare {   //for 64 Bytes spare area
        __u32 journalBlockNums;
        __u32 currWorkArea;
        __u32 nextWorkArea;
        __u32 unmountFlag;
        //... ECCs and reserved
};
```

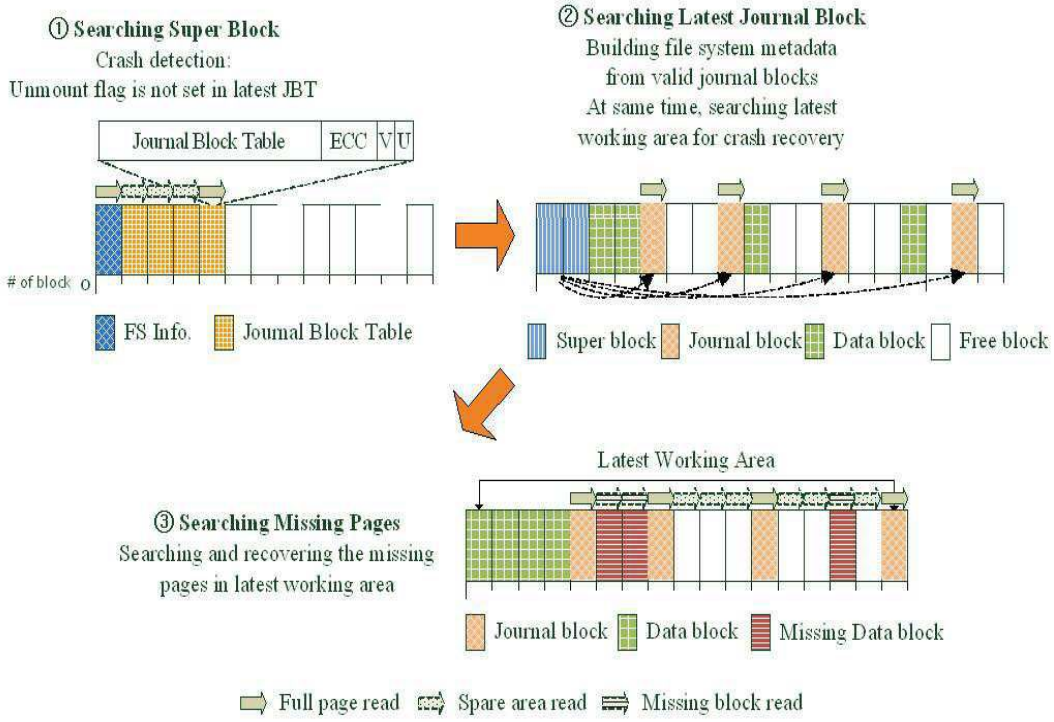
도면4


Super Block
FS-Configurations
Journal Block Table
Duplicated
0
1
2
3
Super block
Reserved Block

도면5


Super Block
FS-Configurations
Journal Block Table
Working Area #0
Working Area #1
Working Area #2
Working Area #3
Super block
Journal block
Data block
Free block

도면6


① Searching Super Block
Crash detection:
Unmount flag is not set in latest JBT
Journal Block Table
ECC
V
U
# of block
0
FS Info.
Journal Block Table
② Searching Latest Journal Block
Building file system metadata
from valid journal blocks
At same time, searching latest
working area for crash recovery
Super block
Journal block
Data block
Free block
③ Searching Missing Pages
Searching and recovering the missing
pages in latest working area
Latest Working Area
Journal block
Data block
Missing Data block
Full page read
Spare area read
Missing block read