(19) 대한민국특허청(KR)
(12) 특허공보(B1)

(51) Int. Cl.[5]
G06F 11/14

(45) 공고일자 1994년09월24일
(11) 공고번호 특1994-0008605

(21) 출원번호	특1991-0009802	(65) 공개번호	특1992-0001347
(22) 출원일자	1991년06월14일	(43) 공개일자	1992년01월30일
(30) 우선권주장	548,720 1990년06월29일 미국(US) 546,306 1990년07월02일 미국(US)		
(71) 출원인 디지탈 이퀍먼트 코오포레이숀 로날드 이. 마이릭
미합중국, 매사추세츠 01754, 메이나드, 파우더밀 로드 111

(72) 발명자 데이비드 비. 로메트
미합중국, 매사추세츠 01886, 웨스트포드, 체리레인 9
피터 엠. 스피로
미합중국, 뉴햄프셔 03062, 나수아, 웨이크필드 드라이브 1
아소크 엠. 조시
미합중국, 뉴햄프셔 03062, 나수아, 세인트 제임스 플레이스 718
아난쓰 라가반
미합중국, 뉴햄프셔 03062, 나수아 비베이 리지 드라이브 26
티루만자남 케이, 렝가라잔
미합중국, 뉴햄프셔 03062, 나수아, #6, 햄릿 드라이브 65

(74) 대리인 나영환, 도두형

심사관 : 오흥수 (책자공보 제3746호)

(54) 언두 로그 사용을 최적화하는 방법 및 장치

요약

내용 없음.

대표도

도1

명세서

〔발명의 명칭〕

언두 로그 사용을 최적화하는 방법 및 장치

〔도면의 간단한 설명〕

제1도는 본 발명을 실시하는 컴퓨터 시스템도.

제2도는 블록과 페이지를 나타내는 디스크의 일부도면.

제3도는 리두 로그를 나타낸 도면.

제4도는 언두 로그를 나타낸 도면.

제5도는 아치브 로그를 나타낸 도면.

제6도는 리두 연산을 수행하는 흐름도.

제7도는 크래쉬 회복을 수행하는 흐름도.

제8도는 아치브 로그를 합병하기 위한 흐름도.

제9도는 더티 블록 테이블을 나타낸 도면.

제10도는 언두 로그 사용을 최적으로 하기 위하여 라이트 어헤드 프로토콜을 실시하는 흐름도.

제11도는 보상 로그 레코드를 나타내는 도면.

제12도는 액티브 트랜잭션 테이블을 나타내는 도면.

제13도는 트랜잭션 개시 연산을 나타내는 흐름도.

제14도는 블록 갱신 연산을 나타내는 흐름도.

제15도는 블록 기입 연산을 나타내는 흐름도.

제16도는 트랜잭션 중단 연산을 나타내는 흐름도.

제17도는 트랜잭션 준비 연산을 나타내는 흐름도.

제18도는 트랜잭션 위임 연산을 나타내는 흐름도.

* 도면의 주요 부분에 대한 부호의 설명

150 : 백업 테이프 140 : 공유 디스크 시스템

110, 120, 130 : 노드 113, 123, 133 : 프로세서,

118, 128, 138 : 메모리 200 : 디스크

212, 214, 216, 218 : 페이지 900 : 더티 블록 테이블

1200 : 액티브 트랜잭션 테이블

[발명의 상세한 설명]

본 발명은 공유 디스크 시스템에서 크래쉬(crash) 회복 분야에 관한 것으로, 특히, 상기한 회복시 로그를 사용하는 방법 및 장치에 관한 것이다.

모든 컴퓨터 시스템은 컴퓨터가 크래쉬될 때 데이터가 손실될 염려가 있다. 디스크와 프로세서 메모리 사이에서 전후로 대량의 데이터를 전송하는 데이터 베이스 시스템과 같은 일부 시스템은 시스템 고장 또는 크래쉬가 발생할 때 특히 데이터 손실 가능성이 크다.

데이터 손실의 일반적인 이유는 휘발성 기억 시스템(예를 들면, 프로세서 메모리)으로부터 영구 기억 시스템(예를 들면, 디스크)으로의 데이터 전송이 불완전하기 때문이다. 가끔, 상기 불완전 전송은 크래쉬가 발생하였을 때 트랜잭션을 수행하기 때문에 발생한다. 트랜잭션은 일반적으로 두 개의 기억 시스템 사이에서 일련의 기록(또는 변화)의 전송을 포함한다.

데이터 손실 및 그 손실로부터 회복에 대한 중요한 개념은 트랜잭션을 "위임(committing)"한다는 생각이다. 트랜잭션은 모든 트랜잭션의 효과가 영구 기억으로 안정될때에 위임된다. 트랜잭션이 위임되기전에 크래쉬가 발생하면 그 회복에 필요한 단계들은 트랜잭션이 위임된 후에 크래쉬가 발생한 경우이 회복에 필요한 단계들과 다르다. 회복(recovery)은 완전한 시스템이 알려진 소망하는 지점에서 다시 시작하게 하는 데이터 베이스에 대한 교정을 수행하는 공정이다.

필요한 회복의 형태는 물론 데이터 손실의 원인에 의존한다. 만일 컴퓨터 시스템이 크래쉬되면 회복 동작은 컴퓨터 시스템의 영구 기억 장치(예를 들면 디스크)의 복원이 최종 위임된 트랜잭션에 의해 발생된 것과 일치되는 상태로 되도록 할 필요가 있다. 만일 영구 기억 장치가 크래쉬되면(소위 매체 고장) 회복 동작은 디스크상에 기억된 데이터를 개조해야 한다.

데이터 베이스 시스템을 회복시키는 많은 방법에는 로그(log)를 사용하고 있다. 로그는 단지, 적어도 데이터 베이스 시스템의 경우에, 데이터 베이스에 대하여 어떠한 변화가 발생하였는지 및 그 변화가 어떤 순서로 발생하였는지를 나타내는 시간 순서의 동작 리스트에 불과하다. 따라서 로그는 리두(read) 또는 언두(undo) 변화에 사용될 수 있는 소망하는 상태로 컴퓨터 시스템이 데이터 베이스에 위치될 수 있게 한다.

그러나 로그는 소위 노드라 하는 다수의 컴퓨터 시스템이 공유 디스크의 수집을 액세스하는 시스템 구서에서는 관리하기가 어렵다. 이런 형태의 구성은 "클러스터" 또는 "공유 디스크" 시스템이라고 부른다. 사기 시스템의 어떤 노드가 어떤 데이터를 액세스하게 하는 시스템은 "데이타 공유" 시스템이라 부른다.

데이터 공유 시스템은 데이터 블록 자신들을 디스크로부터 요구 컴퓨터로 전송시키는 "데이타 쉬핑(data shipping)"을 수행한다. 대조적으로, 분할 시스템이라고 더 잘 알려진 기능 쉬핑 시스템은 연산의 수집을 데이터 분할을 위해 "서버"라고 표시된 컴퓨터에 전송한다. 그 다음 상기 서버는 그 연산을 수행하고 그 결과를 다시 요구 컴퓨터에 역전송한다.

분할 시스템에서, 단일 노드 또는 집중된 시스템에서와 마찬가지로 데이터의 각 부분은 겨우 한 노드의 로컬 메모리에 존재할 수 있다. 또한 분할 시스템 및 집중 시스템은 모두 단일 로그상에서의 기록 동작만을 필요로 한다. 중요한 것은 데이터 회복이 단지 하나의 로그의 내용에 기초하여 진행될 수 있다는 것이다.

반면에, 분배된 데이터 쉬핑 시스템은 분산되며, 따라서 동일한 데이터가 복수 노드의 로컬 메모리에 존재할 수 있으며 그 노드로부터 갱신될 수 있다. 이것은 동일 데이터에 대한 복수 노드 로깅 동작을 초래한다.

복수 로그가 동일 데이터에 대한 동작을 포함하는 문제를 방지하기 위하여, 데이터 공유 시스템은 그 데이터에 대한 로그 기록이 그 데이터에 대한 회복 정보를 기록하는 단일 로그에 역전송될 것을 요구할 수 있다. 그러나 이러한 "리모트" 로깅은 여분의 시스템 자원을 요구하는데, 그 이유는 여분의 메시지가 그 로그에 대한 I/O 기입 이외에 필요한 로그 기록을 포함하기 때문이다. 또한, 로깅 컴퓨터로부터의 인식을 기다리는데 관련된 지연이 필요할 수 있다. 이것은 응답 시간을 증가시킬 뿐만 아니라 수인의 사용자가 동일 데이터 베이스에 대해 동시 액세스하는 능력을 감소시킬 수 있다.

또 하나의 방법은 로그에 대한 기입을 취함으로써 공통 로그의 사용을 동기화하는 것이다. 이것은 조정을 위한 여분의 메시지를 포함하기 때문에 비용이 너무 많이 든다.

상기 단점들은 데이터 공유 시스템이 가끔 분할 시스템이기 때문에 어드레스에 있어서 중요하다. 예를 들어, 데이터 공유 시스템은 워크 스테이션 및 엔지니어링 설계 응용을 위해 중요한 것인데, 그 이유는 데이터 공유 시스템이 워크 스테이션으로 하여금 데이터의 고성능 로컬 처리를 허용하는 연장된 기간동안 데이터를 저장하게 하기 때문이다. 더우기, 데이타 공유 시스템은 복수 노드가 데이터를 동시에 액세스할 수 있고, 일부 로컬 데이터를 관리할 수 있으며, 다른 호스트 컴퓨터 및 워크 스테이션과 함께 다른 데이터를 공유할 수 있기 때문에 본래 고장 허용성이고 부하 조화성이다.

따라서 본 발명의 목적은 리두 기록으로부터 언두 정보를 제거함으로써 리두 로그 관리를 쉽게 하는데에 있다.

본 발명의 다른 목적은 트랜잭션 위임시 언두 정보를 제거함으로써 언두 정보의 관리를 보다 용이하게 하는데에 있다.

본 발명의 또 하나의 목적은 크래쉬 또는 고장이 발생한 경우에 언두 처리를 위해 기억되는야 하는 정보를 최소로 하는데에 있다.

본 발명은 비위임된 트랜잭션의 모든 변화가 제거되고 위임된 트랜잭션으로부터의 변화가 개조되며, 언두 버퍼의 내용을 언두 로그에 기억하는 것이 최소로 될 수 있도록 리두 및 언두 버퍼로부터의 충분한 정보가 유지되게 함으로써 종래의 문제점을 제거한다. 다른 효과는 동작이 실행되지 않을 때 트랜잭션의 카운트 동작을 정확히함으로써 얻어질 수 있다. 특히, 구획으로 분리된 복수의 노드 및 비휘발성 기억 매체를 포함한 데이터 처리 시스템에 있어서, 복수의 노드는 트랜잭션 의해 상기 구획들에 대한 변화를 만들고, 각각의 트랜잭션은 적어도 하나의 노드에 의해 적어도 한 구획에 대해 만들어진 일련의 변화를 포함하며, 각각의 트랜잭션은 그 트랜잭션에 의해 실행된 변화의 기록 및 그 트랜잭션의 완료를 나타내는 표가 신뢰성있게 기억 매체에 기억되는 경우에 위임되어진다. 그렇지 않고 비위임된 경우에는 복수의 노드중 첫 번째의 것이 수개의 요소들을 포함한다. 이들 요소로는 적어도 한 구획의 복사본을 기억하는 메모리와 ; 상기 메모리에 결합되어 그 메모리에서 적어도 한 구획의 복사본에 대해 변화를 일으키는 처리 수단과 ; 상기 메모리에서 적어도 한 구획의 복사본에 대해 처리 수단에 의해 만들어진 일련의 변화 리스트를 포함하는 적어도 하나의 언두 버퍼와(각각의 언두 버퍼는 다른 비위임된 트랜잭션에 대하여 제1노드에 의해 만들어진 변화에 대응한다) ; 메모리에서 적어도 한 구획의 복사본에 대하여 대응하는 노드의 처리수단에 의해 만들어진 일련의 변화 리스트를 포함하는 리두 버퍼와 ; 메모리에 결합되어 적어도 한 구획의 복사본을 다시 기억 매체에 기억시키는 기억 수단과 ; 상기 언두 버퍼 및 리두 버퍼에 결합되어 비위임된 트랜잭션의 모든 변화의 효과를 제거하고 위임된 트랜잭션의 모든 변화의 효과를 개조하는 것을 보장하도록 언두 버퍼와 리두 버퍼의 일부를 기억 매체에 선택적으로 기억시키는 로그 관리 수단을 포함한다.

이하, 첨부 도면을 참조하여 본 발명의 실시예를 설명한다.

A. 시스템 부품

제1도의 시스템(100)은 본 발명의 실시예에 사용될 수 있는 기억 시스템의 일예이다. 시스템(100)은 수개의 노드(110, 120, 130)와 공유 디스크 시스템(140)의 모든 액세싱을 포함한다. 각각의 노드(110, 120, 130)는 이후 설명되는 기억 및 회복 루틴을 실행하는 프로세서(113, 123, 133)를 각각 포함한다. 노드(110, 120, 130)는 또한 적어도 두가지의 기능을 제공하는 메모리(118, 128, 138)를 각각 포함한다. 그 기능들중 하나는 대응하는 프로세서에 대한 로컬 메모리로서 작용하는 것이고 다른 한 기능은 디스크 시스템(140)과 교환되는 데이터를 기억하는 것이다. 데이터 교환에 사용되는 메모리 부분은 캐쉬라고 부른다. 캐쉬는 일반적으로 휘발성 시스템 기억 장치이다.

공유 디스크 시스템(140)은 또한 영구 기억 장치라고 부른다. 영구 기억 장치는 시스템의 일부 또는 전부가 크래쉬 될 때 그 내용이 지속될 것으로 가정되는 비휘발성 시스템 기억 장치이다. 종래에 이 기억 장치는 자기 디스크 시스템이었지만 영구 기억 장치는 또한 광학 디스크 또는 자기 테이프 시스템일 수 있다.

또한, 본 발명을 실시하는데 사용되는 역구 기억 장치는 제1도에 도시된 구조로 한정되지 않는다. 예를들어, 영구 기억 장치는 다른 노드에 각각 결합된 수개의 디스크를 포함할 수 있으며, 상기 노드들은 소정형태의 네트워크에 연결된다.

영구 기억 장치의 다른 부분은 "아치브 기억 장치"라고 하는 백업 테이프 시스템(150)이다. 아치브 기억장치는 영구 기억 장치의 데이터가 판독 불능으로 될 경우 영구 기억 장치의 내용을 재구성할 수 있는 정보에 사용되는 시스템 기억 장치를 설명하는데 일반적으로 사용되는 용어이다. 예를 들어, 공유 디스크 시스템(140)이 매체 고장을 갖고 있으면, 테이프 시스템(150)은 디스크 시스템(140)을 복원하기 위하여 사용될 수 있다. 아치브 기억 장치는 빈번하게 자기 테이프 시스템을 포함하지만, 이것은 또한 자기 또는 광학 디스크 시스템을 포함할 수 있다.

데이터 시스템(100)은 시스템의 회복 가능 부분인 블록내에 일반적으로 기억된다. 블록들은 이들이 어떤 노드의 캐쉬에 있을때에만 동작될 수 있다.

제2도는 디스크(200)의 일부상에 있는 수개의 블록(201, 220, 230)의 예를 도시한 것이다. 일반적으로 블록은 영구 기억 장치의 정수개의 페이지를 포함한다. 예를 들어 제2도의 블록 210은 페이지 212, 214, 216 및 218을 포함한다.

B. 로그

전술한 바와 같이, 데이터 베이스 시스템은 회복 목적을 위해 로그를 사용한다. 로그는 일반적으로 영구 기억 장치에 기억된다. 노드가 영구 기억 장치를 갱신할때 노드는 노드 캐쉬의 버퍼내에 상기

갱신을 나타내는 로그 기록을 기억한다.

본 발명의 제기된 실시예는 영구 기억 장치에서 3가지 형태의 로그를 갖지만 각 노드의 캐쉬에는 단지 두가지 형태의 버퍼만을 포함한다. 상기 로그들은 리두 로그(RLOG), 언두 로그(ULOG) 및 아치브 로그(ALOG)이다. 상기 버퍼들은 리두 버퍼와 언두 버퍼이다.

제3도에는 RLOG의 예를, 제4도에는 ULOG의 예를, 제5도에는 ALOG의 예를 도시하고 있다. 리두 버퍼의 구성은 RLOG의 구성과 유사하고, 언두 버퍼의 구성은 ULOG의 구성과 유사하다.

로그 일련 번호(LSN)는 로그에서 기록의 어드레스 또는 관련된 위치를 나타낸다. 각각의 로그는 그 로그에서 기록에 대한 LSN을 타나낸다.

1. RLOG

제3도에 도시된 바와 같이 RLOG(300)는 그 변화중에 발생하는 특정 연산이 반복되게 하는 변화에 대하여 정보를 기록하는데 사용되는 순차 파일의 일예이다. 일반적으로 상기 연산들은 로그 동작이 수행되는 상태로 블록이 복원되면 회복 동작동안 반복되어야 한다.

제3도에 도시된 바와 같이 RLOG(300)는 몇가지의 부속물을 각각 갖는 수개의 레코드(301, 302, 310)를 포함한다. TYPE 부속물(320)은 대응하는 RLOG 레코드의 형태를 식별한다. 다른 형태의 RLOG 레코드 예로는 리두 레코드, 보상 로그 레코드, 및 위임 관련 레코드가 있다. 이들 레코드에 대하여는 후술한다.

TID 부속물(325)은 현재 레코드와 관련된 트랜잭션에 대한 유일한 식별자이다. 이 부속물은 현재의 RLOG 레코드에 대응하는 ULOG에서 레코드를 찾는 것을 돕기 위하여 사용된다.

BSI 부속물(330)은 "이전 상태 식별자"이다. 이 식별자는 이후 상세히 설명된다. 간단히 말해서 BSI는 대응하는 트랜잭션에 의해 수정되기 전에 블록의 형태에 대한 상태 식별자의 값을 나타낸다. BID 부속물은 RLOG 레코드에 대응하는 갱신에 의해 수정된 블록을 식별한다.

REDO DATA 부속물(340)은 대응하는 동작의 성질을 나타내며 재실행될 동작에 대한 충분한 정보를 제공한다. 용어 "갱신"은 본 명세서에서 용어 "동작"과 함께 폭넓게 그리고 상호 교환적으로 사용된다. 동작은 기록 갱신 뿐만 아니라 기록 삽입 및 삭제를 포함하며, 또한 블록 할당과 자유를 포함한다.

LSN 부속물(345)은 RLOG(300)의 상의 현재 레코드를 유일하게 식별한다. 후술되는 바와 같이, LSN 부속물(345)은 제기된 실시예에서 리두 스캔 및 RLOG의 체크 포인팅을 제어하기 위해 사용된다. LSN(345)은 제기된 실시예에서 RLOG 레코드나 블록내에 기억되지 않는다. 그 대신 RLOG의 레코드 위치에 본래부터 존재한다.

본 발명의 한가지 목표는 각각의 노드가 가능한한 다른 노드와는 무관하게 그 회복을 관리하게 하는 것이다. 이를 위하여 별도의 RLOG가 각 노드에 관련된다. 제기된 실시예에서 모즈와 RLOG의 관련은 각 노드에 대한 다른 RLOG의 사용을 포함한다. 대안으로서, 노드들은 RLOG를 공유할 수 있으며 또는 각 노드가 복수의 RLOG를 가질 수 있다. 만일 RLOG가 노드에 전용되면 메시지를 포함하는 동기화가 다른 RLOG 및 노드와 함께 RLOG를 사용할 필요가 없다.

2. ULOG

제4도에서 ULOG(400)는 블록상의 연산이 정확히 비실행되게 하는 정보를 기록하는데 사용된 순차 파일의 일례를 나타낸 것이다. ULOG(400)는 트랜잭션이 시작하였을때에 존재하는 조건으로 블록들을 복원시키는데 사용된다.

RLOG와는 달리 각각의 ULOG 및 언두 버퍼는 다른 트랜잭션과 관련된다. 따라서 ULOG 및 그 대응하는 버퍼들은 트랜잭션 위임으로써 사라지며, 새로운 트랜잭션이 시작될 때 새로운 ULOG가 나타난다. 다른 현상도 발생할 수 있다.

ULOG(400)는 두 개의 필드를 각각 포함한 수개의 레코드(401, 402, 410)를 구비한다. BID 필드(420)는 이 레코드로 로그된 트랜잭션에 의해 수정된 블록을 식별한다. UNDO DATA 필드(430)는 갱신 성질을 나타내며 비실행될 갱신에 대한 충분한 정보를 제공한다.

RLSN 필드(440)는 동일 동작을 나타내는 RLOG 레코드를 식별하며, 상기 동작은 언두이다. 이 부속물은 각각의 ULOG를 유일하게 식별하는 능력을 제공한다.

3. ALOG

제3도에서, ALOG(500)는 예를 들면 제1도의 공유 데이터 시스템(140)이 고장일때에 매체 회복을 제공하도록 충분한 기간 동안 리두 로그 기록을 기억하는데 사용되는 순차 파일의 일예를 나타낸다. RLOG 버퍼는 ALOG(500)가 발생되는 정보원이며, 따라서 ALOG(500)는 RLOG(300)와 동일한 부속물을 갖는다.

ALOG(500)는 대응 RLOG의 절두부로부터 형성되는 것이 좋다. 절두부는 블록의 영구 기억 장치의 내용을 더 이상 현재의 내용으로 할 필요가 없는 부분이다. 그러나 RLOG의 절두부의 기록은, 만일 블록의 영구 기억 장치의 내용이 이용 불능으로 되고 아치브 기억 장치상의 블록의 내용으로부터 회복될 필요가 있다면 여전히 필요하다.

ALOG(500)는 RLOG(300)와 유사하게 수개의 레코드(501, 502, 510)를 포함한다. TYPE(520), TID(525), BSI(530), BID(535) 및 REDO-DATA 부속물(540)은 RLOG(300)의 동일 명칭을 가즌 부속물들과 동일한 기능을 갖는다. RLOG(300)의 LSN(345)와 마찬가지로 LSN(545)은 ALOG레코드를 식별한다.

C. 상태 식별자(및 라이트-어헤드 로그 프로토콜)

로그 베이스 시스템에서, 로그 기록은 그 블록의 기록 상태가 로그 기록에 의해 표시된 갱신에 적합한 경우에만 블록에 인가된다. 따라서, 블록이 원래의 동작을 수행할때와 동일한 상태에 있을 때 리두를 보정하기 위한 충분 조건은 로그된 트랜잭션을 블록에 인가하는 것이다. 만일 원래 동작이 보정되었으며, 재실행 동작이 또한 보정된다.

블록 상태의 전체 내용을 로그상에 기억시키는 것은 비실제적이다. 따라서, 블록 상태를 위하여 가장 근사한 값 또는 식별자가 창조된다. 제기된 실시예에서 사용되는 식별자는 상태 식별자(SI)이다. SI는 각각의 블록에 대하여 유일한 값을 갖는다. 그 값은 블록상의 어떤 연산의 수행 이전 또는 이후의 어떤 특정 시간에 블록의 상태를 식별한다.

SI는 완전한 상태보다 훨씬 더 적으며 필요할 때 완전한 상태가 개조될 수 있는한 완전한 상태의 위치에 염가로 사용될 수 있다. SI는 블록내에서 "한정 상태 식별자(DSI)"라고 부르는 특정값을 기억함으로써 한정된다. DSI는 포함되는 블록의 상태를 나타낸다.

상태 개조는 회복중에 영구 기억 장치에 기억된 전체 블록을 액세스하고 그 블록의 DSI를 주지함으로써 달성될 수 있다. 이 블록상태는 그 다음 로그 동작을 적절히 적용함으로써 후술되는 바와 같이 갱신되어진다.

ALOG를 사용하는 매체 회복에 대하여 유사한 기술은 이후 설명된다. 로그 레코드가 블록에 인가되는지의 여부를 아는 것은 로그 레코드로부터 어떤 상태로 로그 동작이 적용되는지를 결정하는 것과 관련된다. 본 발명에 따르면, 블록의 DSI는 로그 레코드가 언제 그 블록에 인가되는지를 결정하기 위해 사용된다.

집중된 또는 분할된 시스템에 있어서, 단일 로그상의 물리적인 레코드 순서는 재실행될 동작의 순서를 정하기 위해 사용된다. 즉, 블록상의 동작 B가 블록상의 동작 A에 곧바로 이어진다면 동작 B는 동작 A에 의해 형성된 블록상태에 적용된다. 따라서 동작 A가 재실행되면, 블록에 인가될 다음 로그 레코드는 동작 B이다.

집중된 또는 분할된 시스템과 같은 단일 로그 시스템은 블록 상태를 식별하기 위하여 SI로서 LSN을 자주 사용한다. 블록에 대한 DSI로서 사용되는 LSN은 그 효과가 블록에 반영되도록 로그 순서로 최종 기록을 식별한다. 이러한 시스템에서, 로그 기록의 LSN은 로그 동작 다음의 블록 상태를 식별하는 "이후 상태 식별자(ASI)"의 임무를 수행할 수 있다. 이것은 본 발명에서 이후 설명되는 바와 같이 로그 기록에 사용되는 BSI(이전 상태 식별자)와 대조된다.

DSI를 갱신하고 다음 연산을 준비하기 위하여, 로그 기록을 적용한 후에 블록에 대한 ASI를 결정할 수 있게 할 필요가 있다. BSI 등의 로그 레코드로부터 ASI를 유도하는 것은 유용하며 따라서 ASI는 ASI가 진정으로 기억될 수 있다 하더라도 로그 레코드에 기억될 필요가 없다. 그러나 상기 유도는 정상 동작중에 뿐만아니라 회복 동안에 사용될 수 있는 것이어야 한다. SI는 0에서 시작하여 단조롭게 증가하는 정수 세트이다. 이 기술에서 ASI는 항상 BSI 보다 1이 더 크다.

갱신된 블록을 공유 디스크 시스템(140)과 같은 영구 기억 장치에 다시 기억시킬 때, 라이트 어헤드 로그(WAL) 프로토콜이 사용된다. WAL 프로토콜은 리두 및 언두 버퍼가 블록 이전의 공유 디스크 시스템의 로그에 기이되어야 할 것을 요구한다. 이것은 동작을 반복하거나 비실행하는데 필요한 정보가 영구적인 데이터 복사본을 변화시키기 전에 안정되게 기억되게 한다.

WAL이 이어지지 않고 블록이 그 블록의 최종 갱신을 위한 로그 레코드 이전에 영구 기억 장치에 기입되어져야 한다면, 회복은 특정 조건하에서 발생할 수 없다. 예를 들어, 한 노드에서의 갱신은 비위임된 갱신을 포함한 블록이 영구 기억 장치에 기입되도록 할 수 있다. 그 블록에 대한 최종 갱신이 노드의 RLOG에 기억되어 있지 않고 제2노드상의 다른 트랜잭션이 블록을 추가 갱신하여 위임한다면, 블록의 DSI가 증가된다. 상기 제2트랜잭션의 위임시에 상기 다른 트랜잭션의 로그 동작은 제2노드의 RLOG에 강요된다. 그러나 제2갱신이 다른 노드에 의해 발생되기 때문에 제2트랜잭션에 대한 로그 레코드 기입은 원래 노드상의 비위임된 트랜잭션에 대한 로그 레코드가 기입되어지는 것을 보장하지 않는다.

원래 노드가 크래쉬되고 비위임된 트랜잭션의 로그 레코드가 RLOG에 기입되지 않으면, 그 블록에 대한 ASI-BSI 순서에서 갭이 발생된다. 영구 기억 장치의 블록이 예를 들어 디스크 고장 때문에 이용불능으로 되면 후술되는 바와 같이 ALOG 합병이 공지의 갭이 없는 SI 순서를 요구하기 때문에 회복이 불가능해진다.

따라서, WAL 프로토콜은 끊어지지 않은 로그 동작 순서를 위한 필요 조건이다. 블록이 하나의 노드 캐쉬로부터 다른 노드 캐쉬로 이동할 때, WAL 프로토콜은 그 블록에 대한 모든 이전 갱신의 RLOG 레코드가 위임 트랜잭션에 의해 변화되도록 강요한다. 상기 "강요"는 노드 캐쉬 또는 버퍼의 기록이 영구 기억 장치에 안정되게 기억되게 한다는 것을 의미한다.

영구 기억 장치에 기입함으로써, WAL 프로토콜은 현재 블록에 대한 최종 갱신을 위하여 로그 레코드를 통한 본래 노드의 RLOG에 모든 기록을 기입하도록 강요한다.

D. 새로운 블록 할당

정상적인 디스크 기억 관리 루틴 동안과 같이 블록이 자유로워지고 추가 사용을 위해 나중에 재할당될 때 그 DSI는 그 동작이 비유일한 상태 식별자를 초래하기 때문에 0으로 설정되지 않는다. 만일 DSI가 0으로 설정되면 수개의 로그 레코드가 동일한 SI를 갖기 때문에 어느 한 블록에 인가되도록 나타난다. 추가적인 정보는 정확한 로그 레코드를 결정하기 위하여 필요하다. 따라서, 이전 할당에서 사용된 DSI 번호 매김(mumbering)은 새로운 할당에서 방해되지 않은 상태로 보존되어야 한다. 새로이 할당된 블록의 BSI는 자유로운 상태인 블록의 ASI인 것이 좋다.

방해되지 않은 SI번호 매김을 달성하기 위한 한가지 용이한 방법은 자유 연산의 결과로서 블록에

DSI를 기억시키는 것이다. 블록이 재할당될 때, 블록은 영구 기억 장치로부터 판독되고 정상적인 DSI 증분은 계속된다. 이것은 갱신 동작과 같은 자유 및 할당을 취급한다. 이러한 해결 방법의 한가지 문제는 새로 할당된 블록을 그 사용 이전에 판독해야 한다는 것이다. 그러나 최소의 I/O 동작에 의해 공간 관리 효율을 증가시키기 위하여 "할당전 판독 페널티"를 제거할 필요가 있다.

본 발명은 모든 비할당 블록에 DSI를 기입하지 않음으로써 효과를 얻는다. 블록들이 이전에 할당되지 않기 땜에 초기 DSI는 항상 0으로 설정된다. 할당되지 않은 블록의 DSI만이 기억된다. 이 DSI들은 영구 기억 장치에서 자유 공간의 부기(bookkeepoing)를 위해 시스템에 의해 이미 유지되어 있는 기록들을 사용하여 기억된다. 일반적으로 상기 부기 정보는 공간 관리 블록의 수집으로 기록된다.

상기 공간 관리 정보로서 각각의 할당되지 않은 블록에 대해 초기 SI를 기억시킴으로써, 초기 SI는 블록에 기억될 필요가 없으며, 따라서 할당전 판독 패널티를 제거할 수 있다. 재할당시, 할당 연산을 위한 BSI는 상기 이전 자유 블록의 초기 SI가 된다.

물론 이 절차를 정확히 동작시키기 위하여 공간 관리 정보를 포함한 블록은 영구 기억 장치에 주기적으로 기입되어야 하며, 자유 블록의 존재가 상기 부기에 의해 알려질때까지 하나의 노드가 다른 노드에 의해 자유로워진 블록들을 재할당하도록 허용되어서는 않된다. 따라서, 자유 블록에 대하여 초기 SI를 유지하는 것은 자유 공간 부기 정보의 추가적인 판독 또는 기입을 일으키지 않는다.

비록 자유 블록에 대한 SI 정보의 부가가 이 시스템에서 필요로 하는 공간 관리 정보의 양을 증가시키지 않는다 하더라도 시스템 효율이 왜 너무 많이 손상되어서는 안되는가에 대하여는 두가지 이유가 있다. 첫 번째는 대부분의 자유 공간이 "이전 할당 불가"로서 특징지어지기 때문이며 따라서 초기 SI는 이미 제로값을 갖는다. 두 번째는 이전에 사용된 자유 공간이 대부분의 데이터 베이스에서 작기 때문이며, 그 이유는 데이터 베이스가 일반적으로 성장하기 때문이다. 초기 SI가 재할당된 블록에 대해서만 개별적으로 기억되기 때문에 SI의 증가된 기억 장치는 작아야 한다.

대안으로서, 이전 할당 불가 블록은 재할당된 블록과 구분되어야 한다. 재할당된 블록의 SI는 이 블록들이 할당될 때 영구 기억 장치로부터 판독된다. 그러나 이것은 비록 페널티가 전술한 이유에 대하여 경미하다하더라도 할당 이전 판독 페널티를 만든다.

E. 회복

1. 블록 변형(Block Version)

회복시 로그가 어떻게 사용되는지를 이해하기 위하여 크래쉬 이후에 이용할 수 있는 블록의 다른 변형을 이해할 필요가 있다. 이 변형들은 이용가능한 현재 변형을 형성하기 위하여 얼마나 많은 로그상의 얼마나 많은 로그 갱신이 필요한가에 특징이 있다. 이것은 광범위한 또는 국부적인 회복 동작이 어떻게 시행되는가에 대한 명백한 충격(impact)이다.

회복을 위하여는 3가지의 블록이 사용된다. 블록의 변형은 만일 그 블록에서 수행되는 모든 갱신이 그 변형에 반영된다면 "현재"이다. 고장 이후 현재 변형을 갖는 블록은 리두 회복을 필요로 하지 않는다. 그러나, 예상하지 못한 시스템 고장을 처리할 때, 사용자는 갱신이 발생할때마다 영구 기억 장치에 대한 캐쉬를 통하여 항상 기입함이 없이 모든 블록이 현재가 되도록 보장할 수 없다. 이것은 가격이 많이 들며 거의 행하여지지 않는다.

블록의 변형은 하나의 노드의 로그만이 그 블록에 아직 인가되지 않은 갱신을 갖는 경우 "1-로그"이다. 고장이 발생할때에는 하나의 노드만이 회복에 관여할 필요를 갖는다. 이것은 회복중의 광범위한 조정 및 추가적인 설비 비용을 요하지 않기 때문에 바람직하다.

블록의 변형은 하나 이상의 로그의 노드가 그 노드에 아직 이가되지 않은 갱신을 갖는 경우에 "N-로그"이다. 회복은 일반적으로 1-로그 블록보다는 N-로그 블록에서 더 어렵지만 이 변형은 블록이 항상 1-로그로 되도록 매체 회복을 제공할 때 비현실적으로 되는데 그 이유는 블록이 노드를 변화시킬 때마다 기억을 달성하도록 이 변형이 블록을 기입하기 때문이다.

2. 리두 회복

주의를 하지 않으면 어떤 블록들은 시스템 크래쉬(매체 고장과는 반대임)가 발생할때에 N-로그로 된다. 그러나 본 실시예는 시스템 크래쉬 회복 동안 모든 블록들이 1-로그 블록에 있도록 보장한다. 이것은 N-로그 블록이 회복 동안 노드 사이에서 복잡한 조정을 요구할 수 있기 때문에 유리하다. 비록 갱신이 분배된 동시에 제어를 사용하여 정상적인 시스템 동작중에 처음부터 순서가 정해졌기 때문에 상기한 조정이 가능하다 하더라도, 상기 동시 제어는 회복중에 제거되어야할 오버 헤드를 요구한다.

모든 블록들은 이들이 하나의 캐쉬로부터 다른 캐쉬로 이동하기 전에 더티 블록이 영구 기억 장치에 기입될 것을 요구함으로써 리두 회복에 대하여 1-로그로 되는 것을 보장할 수 있다. 더티 블록은 그 블록이 영구 기억 장치로부터 판독되기 때문에 캐쉬에서의 변형이 갱신되어지는 블록이다.

이 법칙에 따르면 블록이 새로운 노드 캐쉬에 들어갈때에 요구 노드는 항상 깨끗한 블록을 갖는다. 또한 회복중에서 블록을 변화시키도록 최종 노드의 로그상의 레코드 만이 블록에 인가될 필요가 있다. 다른 노드의 모든 다른 동작들은 영구 기억 장치의 블록 상태 이미 포획되어 있다. 따라서 모든 블록들은 리두 회복 동안 1-로그이며, 따라서 리두 회복은 분배된 동시 제어를 요구하지 않는다.

이 기술은 복수의 로그의 블록에 대한 레코드를 포함하지 않는다는 것을 의미하지 않는다. 이 기술은 단지 한 노드의 레코드만이 영구 기억 장치의 블록의 변형에 적용되는 것을 보장한다.

더우기, 비록 1-로그 리두 회복이 매체 회복을 수행하기 위하여 시스템 크래쉬 되었다고 가정하더라도 복수 로그에서의 리두 동작은 블록이 캐쉬 사이에서 이동할때마다 아치브 기억 장치에 각각의 블록을 기입하는 것을 피하기 위하여 인가될 수 있다. 따라서, 어떤 환경하에서는 N-로그 블록에 대한 회복을 제공하기 위하여 모든 로그에 걸쳐서 로그 동작을 순서정할 필요가 있다. 그러나 이것은 순

차 SI에 의해 달성될 수 있다.

제6도는 전술한 RLOG 및 SI를 사용하여 리두 연산을 행하기 위한 기본 단계를 나타낸 흐름도(600)이다. 흐름도(600)로 표시된 리두 연산은 단일 블록에 인가된 단일 RLOG 레코드를 사용하여 단일 노드에 의해 실행된다.

먼저, 로그 레코드에 의해 식별된 가장 최근의 블록 변형이 영구 기억 장치로부터 검색된다(단계 610). 검색된 블록에 기억된 DSI가 로그 레코드에 기억된 BSI와 동일하면(단계 620), 로그 레코드에 표시된 동작이 블록에 인가되고 새로운 블록 상태를 반영하도록 DSI가 증가된다(단계 630). 그렇지 않으면 갱신이 블록에 인가되지 않는다.

제6도에서 설명된 리두 연산은 BSI와 ASI가 회복시 결정될 수 있기 때문에 가능하다. 따라서 사용자는 로그 레코드가 재실행될 필요가 있는 각각의 로그에 대하여 결정할 수 있으며, 이 결정은 다른 로그의 내용과 무관하게 행할 수 있다. 블록 DSI와 로그 레코드 BSI 사이에서 행해져야 할 유일한 비교는 동일성 여부이다.

제6도에서 설명된 리두 연산은 시스템 크래쉬로부터의 회복시에 사용될수 있다. 크래쉬 회복 절차의 한예는 제7도에서 흐름도(700)로 표시하였다. 단일 노드는 다른 노드와 무관하게 상기 크래쉬 회복 절차를 실행할 수 있다.

첫번째 단계는 가장 최근의 체크 포인트에 의해 표시된 제1RLOG기록을 노드가 판독하는 것이다(단계 710). 체크 포인트는, 후술하는 바와같이, 인가될 필요가 있는 가장 오래된 갱신에 대응하는 레코드를 포함하는 RLOG에서 지점을 나타낸다.

그다음, 그 로그 레코드에서 지정된 동작이 그 로그 레코드에서 식별된 블록에 인가되었는지의 여부를 확인하기 위하여 제6도에 도시된 리두 연산이 실행된다(단계 720).

만일 리두 연산을 실행한 후에 더 이상의 레코드가 없으면(단계 730), 크래쉬 회복을 완료하고, 만일 다른 레코드가 있으면 다음 레코드를 RLOG로부터 검색하고(단계 740), 제6도에 도시한 리두 연산을 반복한다.(단계 720).

로그 레코드와 관련된 SI가 단조롭게 증가하는 ASI이면, 로그 레코드가 어떤 상태로 블록에 인가되었는가에 대한 시험은 이 ASI가 블록의 DSI보다 더큰 첫 번째의 것인가에 대한 시험이다. 이것은 1-로그 회복에 대해서만 충분한 것인데, 왜냐하면 이 경우에 오직 하나의 로그만이 브록내에서 DSI보다 더큰 ASI와 함께 레코드를 갖기 때문이다.

그러나 제기된 실시예에서 각각의 로그 레코드는 로그 동작이 실행되기전의 블록 상태와 정확히 동일한 상태를 포함한다. 전술한 바와같이, 이것은 "이전 상태 식별자(BSI)"이다.

3. 매체 회복을 위한 복수 로그 리두

매체 회복은 크래쉬 회복과 동일한 특성을 많이 갖는다. 예를들면, 로그 레코드가 인가되는 변형이 안정되게 기억될 필요가 있다.

여기에서는 또한 중요한 차이점이 있다. 먼저, ALOG레코드가 인가되는 블록의 안정된 변형은 아치브 기억 장치에 기억된 최종 변형이다.

매체 회복은 아치브 기억 장치로부터의 복원 블록을 포함하고 전술한 바와같이 블록들이 캐쉬 사이에서 이동하는 매 시간 마다 아치브 기억 장치에 기입되지 않기 때문에 N-로그이다. 따라서 시스템 크래쉬에 대한 N-로그회복을 피하기 위한 기억 장치에 블록을 기입하는 기술을 매체 회복을 위하여 사용될 수 없다.

매체 회복의 관리는 ALOG를 합병하지 않고서는 어렵다. 만일 ALOG가 합병되지 않으면 회복은 적용가능한 로그 레코드를 탐색하는 상수를 포함한다. ALOG의 합병시에는 BSI를 사용하면 실질적인 장점을 갖는다.

제8도는 복수 ALOG의 합병과 관련된 N-로그 매체 회복에 대한 절차(800)를 도시한 것이다. 합병은 모든 로그 레코드중에서 전체의 순서 정함에 기초하지 않지만 동일 블록에 대한 로그 레코드 사이에서 순서 정하는 것에 기인하는 부분적인 순서 정함에 기초한다. 때때로 각각의 블록에 그 동작들이 인가될 수 있는 레코드를 갖는 복수의 ALOG가 있다. 제8도의 흐르도(800)로부터 명백한 바와같이 이것은 매체 회복중에 그 동작들이 먼저 인가되어는 무형의 것이다.

복수의 ALOG를 합병하여 단일 통로로 아치브 기억 장치의 백업 데이터 베이스에 인가되도록 하는 것은 보다 신속하고 보다 효율적이다. 이것은 SI가 적절히 순서 정해지면 실행될 수 있다. 이것은 전술한 바와 같이 SI가 공지의 순서로 순서 정해지는 이유이며 제기된 실시예는 단조롭게 증가하는 SI를 사용한다.

어떤 ALOG에서 시작하여, 첫 번째 로그 레코드가 액세스된다(단계 810). 그다음 블록 ID와 BSI가 그 레코드로부터 검색된다(단계 820). 다음에, 블록 ID에 의해 식별된 블록이 인출된다(단계 830).

일단 식별된 블록이 인출되면, 그 DSI가 판독되어 ALOG 레코드의 BSI와 비교된다(단계 840). 만일 ALOG의 BSI가 블록의 DSI보다 더 적으면 로그된 동작이 이미 블록내에 포함되어 있으므로 그 레코드는 무시되고 리두 연산은 필요해진다.

만일 ALOG 레코드의 BSI가 블록의 DSI와 동일하면, 로그된 동작은 그 동작을 블록에 인가함으로써 재실행된다. 이것은 SI의 동일성이 블록의 현재 변형에 로그된 동작이 인가되었음을 의미하기 때문이다.

그다음 블록의 DSI가 증가된다(단계 860). 이것은 로그 동작의 인가에 의해 블록의 새로운(그다음)

변형이 창조된다는 사실을 반영한다.

ALOG 레코드의 BSI가 블록의 DSI보다 더 크면, 이것은 로그 레코드에 대응하는 동작을 인가할 적당한 시간이 아니고 대신 다른 ALOG에 기록된 동작을 인가할 적당한 시간임을 의미한다. 따라서, 상기 ALOG의 판독은 일시 정지되어야 하고 다른 ALOG의 판독이 개시된다(단계 87).

만일 다른 ALOG가 미리 정지되어 있으면(단계 880), 제어는 로그가 정지되었을때의 로그 레코드의 블록 ID와 BIS를 추출하도록 단계 820으로 진행한다.

상기 단계들이 완료된후에, 또는 다른 ALOG가 이전에 일시 정지되어 있지 않으면, 다른 ALOG 레코드가 남아 있는지 여부를 판단한다(단계 890). 만일 남아 있으면, 다른 레코드가 인출된다(단계 810). 남아있지 않으면, 절차(800)는 종료된다.

ALOG가 일시 정지될때에는 현재의 ALOG를 진행하는 블록에 대한 레코드를 포함하는 적어도 하나의 다른 ALOG가 있어야 한다. 대기중인 로그 레코드와 함께 일시 정지된 ALOG는 첫 번째 항목(정해진 순서에서)이 다른 입력 스트림(즉, 다른 ALOG)에서의 항목보다 더 늦게 비교되는 입력 스트림으로서 간주된다. 처리는 다른 ALOG를 사용하여 계속된다.

일시 정지된 ALOG의 현재 레코드는 BSI가 다른 ALOG상의 동작을 방해함이없이 미래의 소정 시간에 인가될 수 있어야 한다. 이것이 발생하면 일시 정지된 ALOG는 일시 정지에서 해제된다.

모든 ALOG는 그 동작이 그 블록에 대하여 순서 정해지 SI와 일치되는 순서로 행하여 지기 때문에 동시에 일시 정지되지 않는다. 따라서, ALOG의 합병은 항상 가능하다.

4. 리두 관리

a. 안전 지점 결정

많은 체크 포인팅 기술은 보다 효율적으로 리두 회복을 행하도록 본 발명과 함께 사용될 수 있다. 예를들어, 더티 블록 테이블은 회복 관리 정보를 각각의 더티 블록과 관련되도록 만들어 질 수 있다. 이 정보는 RLOG 및 ALOG를 관리할 때 두가지의 중요한 기능을 제공한다. 먼저, 회복 관리 정보는 RLOG 주사와 절두(truncation)를 제어하는 "안전지점(safe point)"을 결정하는데 사용된다. 두 번째로 회복 관리 정보는 잠재적인 언두 로그 뿐만아니라 RLOG에 대한 WAL 프로토콜을 실행하는데 사용될 수 있다.

안전 지점 결정은 얼마나 많은 RLOG가 리두 회복을 수행하기 위하여 스캔되어야 하는지를 결정하는데에 중요하다. 상기 리두 스캔을 위한 RLOG의 시작점은 "안전 지점"이라고 부른다. 안전 지점은 두가지 의미에서 안전하다. 첫 번째로 리두 회복은 안전 지점을 앞서는 레코드들을 안전하게 무시할 수 있는데 그 이유는 상기 레코드들이 모두 영구 기억장치내 블록의 변형에 이미 포함되어 있기 때문이다. 둘째로, 무시된 레코드들은 더 이상 필요 없기 때문에 RLOG로부터 절두될 수 있다.

상기 두 번째 특징은 언두/리두 로그의 결합 형태에 대하여는 적용되지 않는다. 예를들어, 만일 언두 레코드를 발생했던 긴 트랜잭션이 있었다면, 언두 레코드에서의 동작이 영구 기억 장치에 기입된 리두 레코드에서의 동작을 앞설수 있기 때문에 체크 포인트 이전에는 절두가 불가능하다. 이것은 절두를 방해한다.

더티 블록 테이블(900)은 제9도에 도시하였다. 더티 블록 테이블(900)의 현재 복사본은 휘발성 기억장치에 기억되며 체크 포인팅 공정의 일부로서 RLOG의 영구 기억 장치에 주기적으로 기억된다. 더티 블록 테이블 엔트리(910, 911, 912)는 회복 LSN 필드(920)와 블록 ID 필드(930)를 포함한다.

회복 LSN 필드(920)에서의 값들은 최선행 RLOG 레코드를 식별하며, RLOG 레코드의 동작을 영구 기억 장치의 블록의 변형에 포함되지 않는다. 따라서 LSN 필드(920)의 값은 재실행될 필요가 있는 제1 RLOG 레코드이다.

블록 ID 필드(930)의 값은 회복 LSN에 대응하는 블록을 식별한다. 따라서, 더티 블록 테이블(900)은 블록 더티를 만드는 RLOG 레코드의 매 더티 블록 LSN과 관련된다.

더티 블록 테이블(900)에서의 또 하나의 엔트리는 최종 LSN 엔트리(950)이다. 이 엔트리의 값은 각각의 블록에 대하여 그 블록에 대한 최종 갱신을 설명하는 RLOG 및 ULOG 레코드의 LSN이다. LSN은 로그에서의 위치를 결정하기 위해 필요하기 때문에 DSI대신 사용된다.

최종 LSN(950)은 RLastLSN 955(RLOG에 대하여)와 ULastLSN 958(ULOG 각각에 대하여)의 리스트를 포함한다. 상기 리스트는 WAL 프로토콜을 실행하도록 블록이 영구 기억 장치에 기입될 때 얼마나 많은 RLOG 및 ULOG가 강요되어야 하는지를 각각 나타낸다. 따라서 WAL 프로토콜의 실행은 영구 기억 장치의 블록내에 포함된 모든 동작들이 안정되게 기억된 RLOG 및 ULOG 레코드를 갖고 있음을 의미한다.

RLastLSN(955)과 ULastLSN(958)은 그 임무가 단지 RLOG와 ULOG에 대한 WAL 프로토콜을 실행시키는 것이기 때문에 체크 포인트에 포함되지 않는다. 따라서, 제기된 실시예에서, 상기 엔트리는 체크 포인트 정보에 의해 기억되는 것을 피하기 위하여 회복 LSN으로부터 분리된다.

노드 캐쉬의 모든 블록에 대한 최선행 LSN은 로컬 RLOG에서 리두 스캔에 대한 안전 지점이다. 리두 회복은 안전 지점으로부터 전방으로 로컬 RLOG를 판독하고 후술되는 레코드에서 그 동작을 재실행함으로써 시작된다. 재실행을 필요로 하는 모든 블록들은 상기 스캔중에 조우되는 재실행될 필요성이 있는 모든 동작들을 갖는다.

전술한 바와같이, 1-로그 가정은 각각의 RLOG를 분리하여 관리하는 것을 가능하게 한다. 노드는 그 자신의 RLOG만을 처리하면 되며, 따라서 하나의 노드의 동작은 블록이 어떤 다른 노드의 캐쉬에서 더러워지는 이유가 되지 못한다. 그러므로, 각각의 블록과 관련된 단순한 회복 LSN(RLOG로 명명되지 않은것)을 유지하는 것으로 충분하며, 여기에서 회복 LSN이 로컬 RLOG의 레코드를 식별한다는 것을

알수 있다.

b. 체크 포인팅

체크 포인팅의 목적은 안전 지점의 결정이 전술한 바와같이 시스템 크래쉬를 확실히 존속시키기 위한 것이다. 체크 포인팅은 안전 지점으로 하여금 재실행에 필요한 로그의 일부를 이동 및 수축시키게 하는 블록들을 관리하기 위하여 전략(strategy)과 결합될 수 있다. 체크 포인팅에는 여러 가지의 다른 기술이 있다. 그중 하나는 이후 설명되는데 이 방법은 좋은 기술로 생각되지 않는다.

본 발명을 실시하는 양호한 기술은 "퍼지" RLOG 체크 포인팅의 형태이다. 이것은 트랜잭션 또는 연산이 완료되었는지 여부에 관계없이 체크 포인팅이 실행될 수 있기 때문에 "퍼지"라고 부른다.

체크 포인팅된 정보로부터 더티 블록 테이블(900)의 변형의 회복은 리두스캔이 시작되는 곳의 결정을 가능하게 한다. 더티 블록 테이블(900)내의 블록들만이 재실행될 필요성을 갖는데 그 이유는 이 블록들만이 영구 기억 장치에 기억되지 않은 동작들을 갖기 때문이다. 전술한 바와같이, 더티 블록 테이블(900)은 재실행될 필요가 있는 최선행 로그 트랜잭션을 나타낸다.

RLOG를 통한 시스템 크래쉬 회복과 ALOG를 통한 매체 회복은 일반적으로 상이한 안전지점을 가지며 이에 따라 절두되어 진다. 특히, RLOG의 절두부는 매체 회복을 위하여 계속 필요할 수 있다. 만일 그렇다면, 절두부는 ALOG의 일부가 된다.

ALOG 절두는 RLOG 체크 포인트를 사용한다. RLOG 체크 포인트는 체크 포인트의 시간으로서 RLOG의 절두를 허용하는 안전 지점을 결정한다. 이것은 영구 기억 장치내 데이터의 모든 변형이 상기 안전 지점보다 더 최근의 것이기 때문이며 그렇지 않으면 그 지점은 안전하지 못하다.

ALOG를 절두하기 위하여, 영구 기억 장치상의 블록들은 아치브 기억 장치에 먼저 백업된다. 백업이 완료되었을 때 아치브 체크 포인트 기록은 예를들면 아치브 기억 장치내의 동일한 위치에 기입되어 현재 아치브 체크 포인트의 결정이 개선된 RLOG체크 포인트를 식별한다.

ALOG는 아치브 체크 포인트에서 매체 회복이라고 명명된 RLOG 체크 포인트에 의해 식별된다. 모든 영구 기억 장치 블록들은 RLOG 체크 포인트가 실행된후에 아치브 기억 장치에 기입되며, 따라서 상기 체크 포인트의 안전지점 이전에 만들어진 모든 변화를 반영한다. 블록 백업중에 몇 개의 추가적인 RLOG 체크 포인트가 취해질 수 있다. 이것들은 관련된 로그 레코드들이 모두 아치브 기억 장치내의 블록들의 상태에 포함되는 것을 보장하지 않기 때문에 ALOG 절두에 영향을 주지 않는다. 실행될 필요는 없지만 ALOG에 남아 있는 동작들은 적용 불능으로 검출되며 매체 회복 공정중에서 무시된다.

체크 포인트는 RLOG에 기입된다. RLOG에 기입된 최종 체크 포인트를 찾기 위하여, 그 위치는 노드에 대한 글로발 정보 영역의 대응하는 노드의 영구 기억 장치에 기입된다. 가장 최근의 체크 포인트 정보는 일반적으로 회복중에 액세스된 첫 번째 정보이다. 대안으로서, 사용자는 최종 체크 포인트에 대한 RLOG의 후부를 탐색할 수 있다.

체크 포인트는 시스템이 리두 로그의 크기에 대한 뚜렷한 제어를 가지며, 따라서 리두 회복에 필요한 시간을 갖는다는 순수 RLOG의 중요한 장점을 제공한다. RLOG가 ULOG와 결합되면 안전 지점은 전술한 이유 때문에 로그 절두를 위해 사용될 수 없다.

또한, RLOG로부터 언두 정보를 제거하면 시스템이 영구 기억 장치에 블록을 기입함으로써 로그 절두를 제어할 수 있다. RLOG 절두는 긴 트랜잭션의 중단을 요구하지 않는다. 이것은 절두 로그가 언두 정보를 포함할 때에는 사실이 아니다.

시스템은 블록들을 영구 기억 장치의 해당 위치에 다시 기입함으로서 RLOG에 대한 제어를 실행한다. 사실상, 이러한 블록의 기입은 가끔 체크 포인트의 일부로 간주된다. 블록들은 또한 더 늦은, 즉 RLOG보다 더 뒤에 있는 회복 LSN을 갖는 영구 기억 장치에 기입될 수 있다. 이것은 RLOG에 대한 안전 지점을 로그의 후부에 더 가깝게 이동시킨다. 연산이 새로 기입된 블록에 포함되는 로그 레코드는 리두 회복을 위하여 더 이상 필요하지 않으며 따라서, 절두가 가능하다.

매체 회복은 시스템 크래쉬 회복과 동일한 기본 활용례를 따른다. 블록의 변형은 아치브 기억 장치에서 안정되게 기록된다. 전술한 바와같이, 각각의 ALOG는 하나의 RLOG의 절두부로부터 형성된다. ALOG 그 자체는 어떤 변형의 블록들이 아치브 기억 장치내에 존재하는가에 기초하여 주기적으로 절두될 수 있다.

LSN이 아닌 블록에 기억된 DSI만으로는 어떤 로그가 아치브 기억장치의 블록을 갱신할 책임을 가지는지 및 RLOG의 어디에 이 기록들이 존재하는지를 알수 없다. 따라서, 블록내의 정보는 ALOG 또는 RLOG를 절두하도록 적당한 지점을 결정하는 데에는 부족하다. 그러나 더티 블록 테이블은 RLOG의 절두시에 가이드로서 사용될 수 있다. 또한 RLOG 안전 지점은 ALOG 안전 지점을 설정하는데 사용될 수 있다.

F. ULOG 연산

1. ULOG 관리

RLOG의 ULOG로부터의 분리가 RLOG 연산시에 이루어진다는 장점 이외에도 본 발명은 그러한 분리가 ULOG 연산시에 이루어진다는 장점을 또한 갖는다. 예를들어, 트랜잭션-특정 ULOG는 트랜잭션이 위임되면 버려진다. 따라서 ULOG에 대한 스페이스 관리는 간단하며 언두 정보는 영구 기억 장치에 오랫동안 잔류하지 않는다.

또한, 후술되는 바와같이 로그에 이중 기입된 언두 기록은 주기적으로 피할 수 있다. 언두 기록은 비위임된 데이터를 포함한 블록이 영구 기억 장치에 기입될때에만 기입될 필요가 있다.

ULOG와 RLOG를 리두 로그상에서 분리하는 한가지 단점은 비위임된 데이터를 가진 블록이 WAL 프로토

콜을 만족시키기 위하여 영구 기억 장치에 기입될 때 두 로그가 강요되어야 한다는 것이다. 그러나 일반적으로 비위임된 데이터를 가진 블록을 영구 기억 장치에 기입하는 것을 로그의 분리가 비록 실행시키더라도 실제 이득을 제공하도록 충분히 드물어야 한다.

N-로그 언두에 있어서 복수 노드가 블록에서 비위임된 데이터를 동시에 가질 수 있다. 시스템 크래쉬는 이들 트랜잭션이 모두 비실행될 것을 요구한다. 즉, 시스템 크래쉬는 예를들면 언두 회복중에 블록 액세스를 조정하도록 로킹될 것을 요구한다.

모든 블록들이 언두 회복에 대하여 1-로그로 되게하기 위하여 1-노드로부터 비위임된 데이터를 포함하는 블록은 제2노드에 의해 갱신되도록 허용되지도 않는다. 이것은 블록보다 더 적지 않은 로크 입상을 통하여 달성될 수 있다. 요구 노드는 다른 하나의 노드에 의해 언두 처리가 요구되지 않는 블록을 수신 한다. 따라서, 예를들어, 만일 다른 노드로부터의 트랜잭션이 블록을 갱신하고 그다음 중단되면 그 트랜잭션의 효과는 이미 비실행된다.

비록 1-로그 언두가 복잡성을 감소시킨다 하더라도, 회복 시간에 N-로그 언두의 시스템 수행시의 충격은 N-로그 리두에 대해서보다 훨씬 더 작다. 이것은 시스템 크래쉬때에 위임되지 않은 작은 트랜잭션 세트만이 비실행될 필요를 갖기 때문이다. 또한 블록보다 더 적지 않은 로크 입상을 가짐으로써 실질적으로 동시성을 감소시킬 수 있다.

본 발명의 기술은 일반적으로 짧은 트랜잭션에 대하여 ULOG에 기입할 필요성을 제거한다. 이것은 어떤 특정의 짧은 트랜잭션으로부터의 비위임된 데이터를 갖는 블록을 포함한 캐쉬 슬로트가 거의 필요없기 때문이다. 상기 거의 필요없는 이유는 대부분의 짧은 트랜잭션이 위임되어야 하고 캐쉬 슬로트가 필요하기 전에 중단되어야 하기 때문이다. 훔쳐진 캐쉬 슬로트가 비위임된 데이터를 갖는 블록을 포함한다면, WAL 프로토콜은 모드 적당한 ULOG에 대한 언두 레코드의 기입을 요구한다. WAL 프로토콜은 블록에 대한 더티 블록 테이블 엔트리에서 ULast LSN에 의해 식별된 레코드들을 통하여 각각의 ULOG를 강제 기입함으로써 ULOG에 대해 시행된다. 전술한 바와같이, ULastLSN은 각각의 ULOG의 블록에 대하여 최종갱신의 언두 레코드를 식별한다.

WAL 프로토콜에 의해, 트랜잭션을 갱신함이 없이 블록의 상태들을 기억하는데 필요한 정보는 블록의 영구 기억 장치 변형을 새로운 상태로 중복 기입하기 전에 항상 트랜잭션의 ULOG에 이중 기억된다. 따라서 트랜잭션의 갱신 없는 블록의 상태는 트랜잭션 위임전에 항상 이중으로 될 수 있다. 이 정보는 1) 영구 기억 장치내의 블록 변형이거나, 2) 선행 트랜잭션으로부터의 RLOG 정보를 사용하는 영구 기억 장치의 변형으로부터의 "회복 가능 리두"이거나, 3) 상기 트랜잭션에 대한 WAL 프로토콜에 의해 ULOG상에 로그되거나 리두 회복중에 형성된 언두 정보를 사용하여 상기 항목 1) 또는 2)에서 생성된 변형으로부터의 회복가능한 언두이다.

아직 종래 상태에 영구 기억 장치상에 블록에 있어서는 대응하는 ULOG 레코드없이 RLOG 레코드를 가질 수 있다. 이것은 선택적인 언두 로킹이 있을때에는 공통이다. 또한 상기 블록들에 대하여 대응하는 RLOG 레코드 없이 ULOG 레코드를 갖는 것도 가능하다. 이 경우에 ULOG 레코드들은 무시될 수 있다.

따라서, 리두 회복후에 비실행될 필요가 있는 모든 동작들은 ULOG에서 찾을 필요가 없다. 시스템 크래쉬가 발생하면, 손실된 언두 레코드들은 리두 레코드 및 블록의 이전 상태로부터 발생되어야 한다. 동작이 블록의 상태 및 로그된 동작의 값에만 의존하는한, 언두 레코드의 발생은 동작이 최초 실행되었을 때 모든 이용 가능한 정보가 이 지점에서 이용할 수 있기 때문에 가능하다.

동작들은 ULOG상에서 두가지 이유로 귀결된다. 그중 하나는 블록이 영구 기억 장치내에서 기입되기 때문에 WAL 프로토콜이 ULOG에 대한 버퍼기록을 강요하는 것이고, 다른 하나는 WAL 실행을 위한 ULOG의 기입이 선행 ULOG 레코드의 기입을 야기하고 어떤 경우에는 언두 버퍼에 있는 ULOG 레코드를 뒤따른다는 것이다.

이들 동작에 대하여는 이 레코드들이 ULOG상에 존재하도록 보장되기 때문에 회복중에 언두 레코드를 발생할 필요가 없다. 이것은 영구 기억 장치내의 블록의 변형이 그 동작 다음에 오는 상태를 갖기 때문에 리두 로그 트랜잭션에 대한 ULOG 레코드를 구성할 수 없으므로 중요하다. 다행하게도 상기 블록들에는 ULOG 레코드들이 이미 존재하고 있다.

리두중에, 손실된 언두 레코드들이 발생된다. 리두의 종단까지 발생된 언두 레코드와 ULOG상의 언두 레코드들의 결합은 모든 비위임된 트랜잭션을 롤링백할 수 있다.

2. ULOG의 최적화

본 발명에 있어서, ULOG의 사용은 언두 로그 버퍼의 내용을 필요할때에만 ULOG에 기입하는 것을 확실히 함으로써 최적화될 수 있다. 일반적으로, 언두 버퍼는 현재의 트랜잭션으로부터의 비위임된 데이터를 포함하는 블록이 영구 기억 장치에 기입될 때 ULOG에 기억될 필요만을 갖는다. 만일 트랜잭션이 위임되어 있으면, 트랜잭션에서의 갱신을 비실행(undo)할 필요가 없으며 따라서 언두 버퍼는 버려질 수 있다.

제10도는 WAL 프로토콜을 사용하여 상기 ULOG 최적화를 실행하는 절차를 설명하기 위한 흐름도(1000)이다. 블록의 변형은 영구 기억 장치에 기입되는 것으로 가정한다.

기입될 블록이 비위임된 데이터를 포함하면(단계 1010), 리두 버퍼는 영구 기억장치내의 RLOG에 기입되어야 하며 어떤 언두 버퍼가 영구 기억 장치내의 ULOG에 기입된다(단계 1020).

리두 버퍼를 RLOG에 기입하고 언두 버퍼를 ULOG에 기입한후(단계 1020), 또는 블록들이 비위임된 데이터를 포함하지 않은 경우에(단계 1010), 블록은 영구 기억 장치에 기입된다(단계 1030). 이것은 WAL 프로토콜에 따른다.

따라서, 언두 버퍼는 기억되어질 비위임 데이터가 있는 경우에만 기입된다. 트랜잭션이 위임될때마

다 대응하는 언두 로그 버퍼는 영구 기억 장치에 기입될 필요가 없기 때문에 버려질 수 있다. 또한, 트랜잭션을 위한 ULOG 그 자체는 언두가 이제 더 이상 필요없기 때문에 버려질 수 있다.

위임된 트랜잭션은 영구 기억 장치내 RLOG의 트랜잭션을 위하여 모든 리두 레코드를 기록함으로써 중복된다. 갱신 블록은 그후 어떤 시간에 영구 기억 장치에 기입될 수 있다. 갱신 블록이 기입되기 전에 크래쉬가 발생한 경우에도 RLOG는 블록의 상태를 복원하도록 검색될 수 있으며, 시스템은 트랜잭션이 RLOG에 위임 레코드를 저장함으로써 위임된다는 것을 인식한다.

3. 트랜잭션 중단

따라서 ULOG는 트랜잭션 위임시 버려질 수 있으며, 마찬가지로 트랜잭션 효과의 비실행은 더 이상 필요하지 않다. 트랜잭션 중단에 있어서, 그 상황은 다소 달라진다. 트랜잭션에 대한 ULOG 레코드가 버려지기 전에, 중단된 트랜잭션에 의해 변경된 모든 블록들은 비샐행된 자신들의 변화를 가질 뿐만 아니라 결과적인 비실행 블록 상태가 ULOG 이외의 다른 곳에 중복 기억되는 것을 확실히 할 필요가 있다. 비실행된 상태에서 그 블록 자신들은 영구 기억 장치에 기입되어야 하며("FORCE" 중단이라 함), 또는 언두 트랜잭션이 RLOG에 기입 및 강요되어야 한다("NO-FORCE" 중단이라함). 트랜잭션의 위임과 유사하게, RLOG상의 로깅 동작은 이 경우에 영구 기억 장치내 블록들을 강요할 필요성을 제거한다.

a. NO-FORCE 중단

NO-FORCE 중단은 이전의 갱신 효과를 반전시키는 중단 트랜잭션의 추가 동작으로서 언두 연산을 처리함으로써 실현될 수 있다. 이와같은 "보상"동작은 RLOG상에 "보상 로그 레코드(LLR)"로서 로그된다.

보상 로그 레코드는 언두 레코드를 효과적으로 RLOG에 이동시킨다. 그러나 이 레코드들은 다른 RLOG 레코드로부터 구분하기 위하여 여분의 정보가 필요하다. 또한, SI는 재실행될 다른 로그 트랜잭션에 대하여 CLR을 정확히 배열할 필요가 있다.

제11도는 수개의 부속물을 갖는 CLR(1100)을 도시한 것이다. TYPE 부속물(1110)은 상기 로그 레코드를 보상 로그 레코드로서 식별한다.

TID 부속물(1120)은 트랜잭션에 대한 유일한 식별자이다. 이 부속물은 상기 RLOG CLR에 대응하는 ULOG 레코드의 탐색을 돕는다.

BSI 부속물91130)은 전술한 바와같은 이전 상태 식별자이다. 이와 관련해서 BSI 부속물(1130)은 CLR이 인가되는 때에 블록 상태를 식별한다.

BID 부속물(1140)은 이 레코드로서 로그된 동작에 의해 수행된 블록을 식별한다.

UNDO-DATA 부속물(1150)은 비실행될 동작의 특성을 나타내며 관련된 본래 동작이 블록 상태에 포함된 후에 비실행된 동작에 대한 충분한 정보를 제공한다. UNDO-DATA 부속물(1150)의 값은 ULOG 또는 언두 버퍼에 기억된 대응하는 언두 레코드로부터 얻어진다.

RLSN 부속물(1160)은 이 동작이 언두에 대하여 행하여진 것과 동일한 동작을 나타내는 RLOG 레코드이다. 이 부속물은 ULOG 레코드의 RLSN 부속물(440)로부터 얻어진다.

RLOG의 위치에 의해 식별될 수 있기 때문에 명백하게 기억될 필요가 없는 LSN(1170)은 FRLOG상에서 CLR을 유일하게 식별한다. LSN은 리두 스캔을 제어하고 RLOG를 체크 포인팅하기 위하여 사용된다.

트랜잭션 위임시와 마찬가지로, 트랜잭션이 중단될 때 트랜잭션의 동작을 나타내는 모든 리두 레코드들은 RLOG에 기입되어야 한다. 중단된 트랜잭션에 대하여 이것은 CLR내의 언두 동작을 포함한다. 위임을 위하여, RLOG는 트랜잭션을 위한 모든 리두 레코드가 안정되게 기억되게 하도록 강요된다. 중단을 위하여, 이것은 반드시 필요한 것은 아니다. 필요한 정보는 아직 ULOG상에 존재한다.
그러나, ULOG는 중단된 트랜잭션의 CLR이 RLOG에 중복기입될때까지 버려질 수 없다. 이때 RLOG상의 CLR은 ULOG레코드로 대체된다.

NO-FORCE 방법의 바람직한 특성은 매체 회복에 대하여 리두 위상만이 필요하다는 것이다. 갱신은 이들이 ALOG 합병중에 처리되는 순서로 인가된다. 어떤 필요한 언두가 CLR의 인가에 의해 달성되지 않기 때문에 ALOG를 처리하는 동안 어떠한 별도의 언두 위상도 필요로 하지 않는다.

액티브 트랜잭션 테이블이라 하는 제2테이블은 언두 연산을 실시하는데 필요한 정보를 기록한다. 더티 블록 테이블(900)과 마찬가지로 액티브 트랜잭션 테이블은 그 정보가 시스템 크래쉬의 경우에 보존되도록 RLOG상의 체크 포인트 정보의 일부로 된다.

액티브 트랜잭션 테이블은 비실행의 필요가 없는 트랜잭션을 나타내며, 또한 언두/리두 로깅의 상태 및 언두 진행 상태를 나타낸다. 회복중에 발생하는 것들을 포함하는 모든 시스템 크래쉬로부터의 회복을 보장하기 위하여 액티브 트랜잭션 테이블에는 충분한 정보가 부호화되어야 한다. 회복 성능을 개선하는 어던 정보가 또한 포함될 수 있다.

제12도는 액티브 트랜잭션 테이블(1200)의 일예를 도시한 것이다. 테이블(1200)은 레코드(1201, 1202, 1207)을 포함한다. 각각의 레코드는 수개의 부속물을 포함한다.

TID 부속물(1210)은 트랜잭션에 대한 유일한 식별자이다. 이것은 RLOG 레코드에 대하여 사용된 트랜잭션 식별자와 동일한다.

STATE 부속물(1220)은 액티브 트랜잭션이 2단계 위임의 일부로서 준비되었는지의 여부를 나타낸다. 2단계 위임은 복수 노드 트랜잭션내의 일부를 취할때에 사용된다. 이러한 트랜잭션을 위임하기 위하여 모든 노드는 이들이 트랜잭션을 위임(2단계)하기 전에 트랜잭션을 준비(1단계)하여야 한다. 상기 준비는 하나의 노드가 위임되고 다른 것이 중단될 경우 발생하는 부분 위임을 피하기 위해

실행된다. 준비된 트랜잭션은 롤백될 필요가 있기 때문에 액티브 트랜잭션 테이블(12000)에 유지되어야 한다. 비준비된 트랜잭션과는 달리, 준비된 트랜잭션은 자동적으로 중단되어서는 안된다.

ULOGloc 부속물(1230)은 트랜잭션-특정 ULOG의 위치를 나타낸다. 이 부속물은 ULOG를 찾기 위한 다른 방법이 없을 경우에만 존재할 필요가 있다. 예를들어, TID(1210)는 트랜잭션에 대한 ULOG를 찾는 대체적인 방법을 제공한다.

HIGH 부속물(1240)은 상기 트랜잭션의 ULOG에 기입된 언두 레코드를 갖는 최종 동작인 동작의 RLOG LSN을 나타낸다. 상기 ULOG 레코드는 트랜잭션이 위임되지 않은 경우 트랜잭션을 백하도록 준비하기 위하여 시스템 크래쉬후의 리두 연산중에 기누을 다른 RLOG 레코드가 발생되게 하는 방식으로 RLSN 기누에 RLOG LSN을 포함한다.

NEXT 부소굴(1250)은 비실행의 필요가 없는 트랜잭션에서 다음 동작의 RLOG LSN을 나타낸다. 롤백되지 않은 트랜잭션에 있어서, NEXT 부속물은 그 트랜잭션에 의해 수행된 최종 동작의 레코드 번호이다.

어떤 시스템이 회복중에 비실행된다 하더라도 이 시스템들은 제기된 실시예에서 비실행되지 않는다. 그대신 CLR이(TYPE 부속물에 의해) 태그되며 따라서 CLR은 회복중에 식별된다.

ULOG의 순차적인 특성 때문에 언두 레코드가 ULOG에 강요될 때 모든 선행 언두 레코드들이 또한 지속되도록 보장된다. RLOG 레코드들은 ULOG 레코드와 동일한 순서로 기입된다. 따라서 RLOG 레코드가, 예를들면 그 효과가 영구 기억 장치상의 블록의 변형에 이미 존재하기 때문에 재실행 될 필요가 없다고 판정되면, 모든 선행 RLOG 레코드들은 ULOG에 언두 레코드를 구비한다. 이것은 블록이 기입될 때 ULOG가 강요되기 때문에 발생하며, 따라서 모든 이전의 ULOG 레코드들은 동시에 기입된다. 만일 언두 레코드가 상기 트랜잭션을 위한 재실행중에 발생되면 상기 모든 이전의 레코드들이 이미 ULOG에 존재하기 때문에 언두 레코드들이 버려질 수 있다.

ULOG 레코드가 기입되는 최종 RLOG 레코드의 RLOG LSN은 그 트랜잭션에 대한 액티브 트랜잭션 테이블 엔트리의 HIGH 부속물(1240)(제12도)에 기입된다. 상기 표시된 리두로그 레코드 이전의 RLOG 레코드들은 이들의 모드 ULOG 레코드를 이미 갖고 있기 때문에 리두 연산중에 언두 레코드를 발생하지 않는다. ULOG에서 지정된 레코드를 따르는 RLOG 레코드는 언두 레코드를 발생할 필요가 없다.

언두 레코드 발생은 각각의 트랜잭션에 이미 인가되어 있는 언두 레코드의 번호가 주의깊게 감시되는 경우에는 또한 피할 수 있다. 따라서, 비실행 "하이 워터 마크"는 액티브 트랜잭션 테이블(1200)의 NEXT 부속물(1250)에 부호화 된다. NEXT 부속물(1250)은 트랜잭션에 인가될 다음 언두 레코드의 레코드 번호를 포함한다.

정상 처리중에 NEXT 부속물(1250)은 항상 트랜잭션의 가장 가까운 트랜잭션에 대한 레코드 번호이다. NEXT 부속물(1250)의 값은 이 동작들이 로그될 때 증분된다. 언두 회복중에, NEXT 부속물(1250)의 값은 각각의 언두 동작이 인가되고 그 CLR이 로그된 후에 감소되며 그 선행 언두 레코드를 다음 언두 동작으로서 명명한다. 롤백중에 시스템 크래쉬가 발생하면 NEXT 부속물(1250)에 의해 표시된 것보다 더 높은 레코드 번호를 갖는 언두 레코드는 재인가될 필요가 없으며 따라서 재실행중에 다시 발생될 필요가 없다.

결과적으로 재실행중에 언두 레코드는 레코드 번호가 HIGH 부속물(1240)의 값과 NEXT 부속물(1250)의 값 사이에 있는 RLOG 레코드에 대하여 발생된다. HIGH 부속물(1240)의 값이 NEXT 부속물(1250)의 값보다 더 크거나 같으면 언두 레코드는 전혀 발생될 필요가 없다.

b. 포스 중단

"FORCE" 중단에 의해 CLR은 기입되지 않는다. 그 대신, 블록들이 비실행될 때 블록들은 영구 기억 장치에 강요된다. 이러한 형태의 중단에 있어서는 CLR을 블록에 기입함이 없이 블록이 언두 레코드를 인가한 결과, 및 언두 연산이 수행되는 순서를 포함한다는 사실을 알아야 한다.

그 목표는 시스템 크래쉬의 결과로서 수개의 노드가 단일 블록상에서 트랜잭션을 비실행할 수 있는 N-로그 언두를 지지하는 것이다. 그러므로 각 노드에 의해 수행된 언두 연산의 진행은 안정되게 기록되어야 한다. 이것은 CLR이 NO-FORCE 경우에 달성하는 것이다. CLR 없이 어떤 다른 기술이 요구된다.

다른 방법은 필요한 정보를 영구 기억 장치의 블록에 기입하는 것이다. 비록 CLR이 언두 동작의 완전한 설명을 포함해도, 상기 설명의 모두가 필요한 것은 아니다. FORCE 중단의 경우에 필요한 것은 언두 트랜잭션이 결과를 기록하는 것이고 그들중 일부는 비실행된다.

G. 정상 연산

정상 연산중에, 트랜잭션 개시, 블록 갱신, 블록 기입, 트랜잭션 중단, 트랜잭션 준비 및 트랜잭션 위임 연산들은 회복이 필요할 때 충격을 갖는다. 따라서, 정상 연산중에 회복이 가능하게 되도록 로깅에 대하여 많은 단계들이 취해져야 한다.

제13도는 트랜잭션 개시 연산 절차(1300)를 나타낸다. 먼저, START-TRANSACTION 레코드가 RLOG에 기입되어야 한다(단계 1310). 다음에 트랜잭션이 액티브상태로서 액티브 트랜잭션 테이블(1200)에 들어간다(단계 1320). 그 다음 트랜잭션의 ULOG와 그 등가물이 ULOGloc(1230)에 기록된다(단계 1330). 마지막으로, HIGH(1240) 및 NEXT(1250) 값이 0으로 설정된다(단계1340).

제14도는 블록 갱신 연산의 절차(1400)를 도시한 것이다. 먼저, 요구되는 소망의 동시 제어가 갱신 블록을 로크하도록 수행된다(단계 1410). 이 갱신 블록은 그것이 캐쉬내에 이미 존재하지 않을 경우 영구 기억 장치로부터 액세스된다(단계 1420). 캐쉬내의 블록의 변환시에는 지시된 트랜잭션이 수행된다(단계 1430). 다음, 블록 DSI가 ASI로 갱신된다(단계 1440). 이어서, 갱신용 RLOG 및 ULOG 레코드가 구성되어 적절한 버퍼에 포스트된다(단계 1450). 최종 LSN(950)(제9도)이 적절히 갱신된다(단

계 1460). 그리고 NEXT(1250) 값이 언두 레코드의 ULOG LSN으로 설정된다(단계 1470).

블록이 클리어 되면(단계 1475), 더티 상태가 다시 형성된다(단계 1480). 그러면 그 블록은 회복 LSN(920)이 RLOG 레코드의 LSN으로 설정된 상태에서 더티 블록 테이블(900)(제9도)로 주입된다.

제15도는 블록이 비위임 데이터를 포함한 경우에 대한 블록 기입 동작의 흐름도(1500)를 도시한 것이다. 먼저, WAL 프로토콜이 실시된다(단계 1510). 구체적으로 말하자면 블록을 영구 기억 장치에 기입하기에 앞서서, 모든 언두 버퍼가 그 블록의 대응하는 최종 ULSN(958)(제9도)까지 기입되고, 또 RLOG 버퍼가 최종 기누(955)(제9도)까지 기입된다.

각각의 트랜잭션이 블록의 최종 ULSN에서 식별되면, 이들 트랜잭션에 대한 HIGH가 더티 블록 테이블에서의 최종 ULSN 부속물들에 의해 식별된 언두 레코드의 RLSN 부속물들중의 RLOG LSN 값으로 설정된다. 각각의 최종 ULSN은 TID 및 ULOG LSN을 통해 트랜잭션을 식별해야만 한다. 이들 로그의 경우, 상기 레코드들이 이미 기입되어 있기 때문에 전혀 기입의 필요성이 없는 경우가 있다.

더티 브록 테이블(900)로부터 블록이 제거된 후(단계 1520), 그 블록은 영구 기억 장치에 기입된다(단계 1530). 이어서, 그 블록이 영구 기억 장치에 기입되었음을 나타내기 위하여 블록 기입 레코드가 RLOG에 기입될 수 있으나 이는 선택적으로 행해진다.

제16도는 트랜잭션 중단 연산에 대한 흐름도(1600)를 도시한 것이다. 먼저, NEXT 필드(1250)내의 값에 의해 표시된 언두 레코드가 위치 설정된다(단계 1610). 그리고 요구되는 동시 제어가 비록 그들이 정상 갱신동작에 의해 처리되던중이라 하더라도 실제 수반된 블록상에 수행된다(단계 1620).

다음, 현재 언두 로그 레코드가 지정된 블록에 인가되고(단계 1630), 언두 동작에 대한 CLR이 RLOG에 기입된다(단계 1640). 또, NEXT 필드(1250)의 값이 "현재" 언두 레코드로서 인가될 다음 언두 로그 레코드(단계 1640)를 인덱스하도록 증분된다.

어떤 언두 로그 레코드가 트랜잭션을 위해 더 많이 남아 있으면(단계 1660), 제어동작이 단계 1610으로 복귀된다. 그렇지 않으면, ABORT 레코드가 RLOG상에 위치된다(단계 1670). 이어서, RLOG가 ABORT 레코드를 통해 영구 기억 장치에 기억된다(단계 1680). 그 다음 ULOG가 디스카드된다(단계 1690). 마지막으로, 액티브 트랜잭션 테이블(1200)(제12도)로부터 상기 트랜잭션이 제거된다(단계 1695).

제17도는 트랜잭션 준비 연산에 대한 흐름도(1700)를 도시한 것이다. 먼저, 트랜잭션의 준비 로그 레코드가 RLOG에 기입된다(단계 1710). 다음, RLOG가 이러한 준비 로그 레코드를 통해 실시된다(단계 1720). 마지막으로, "준비중"인 트랜잭션의 상태가 액티브 트랜잭션 테이블(1200)에서 변경된다(단계 1730).

제18도는 트랜잭션 위임 연산에 대한 흐름도(1800)를 도시한 것이다. 먼저, 트랜잭션 위임 로그 레코드가 RLOG에 기입된다(단계 1810). 다음, RLOG가 이러한 레코드를 통해 실시된다(단계 1820). 이어서, ULOG가 디스카드된다(단계 1830). 마지막으로, 액티브 트랜잭션 테이블(1200)로부터 트랜잭션이 제거된다.

H. 시스템 크래쉬 회복 처리

전술한 설명에서는 로그, 상태 식별자 및 회복에 관한 여러 가지 특징을 설명하였었다. 이들은 여러 가지 방법으로 효과적인 회복 개념과 결합될 수 있다. 하기에서 그 바람직한 방법이 설명된다.

1. 분석 단계

분석 단계는 엄밀히 말해서 반드시 필요한 것은 아니다. 그러나, 분석 단계를 거치지 않으면 다른 회복 단계 동안 몇가지 불필요한 작업이 수행될 수 있다.

분석 단계의 목적은 최종 체크 포인트에서 기억된 시스템 상태를 그 시스템이 크래쉬된 순간에 데이터 베이스의 상태까지 끌어올리는 것이다. 이를 위하여, RLOG에 대한 최종의 완전한 체크 포인트의 정보가 판독되어 더티 블록 테이블(900)(제9도) 및 액티브 트랜잭션 테이블(1200)(제12도)의 값을 초기화하는데 사용된다. 이어서, 이러한 최종 체크 포인트에 따른 RLOG 레코드가 판독된다. 분석 단계는 로그된 동작을 상기 두 테이블상의 효과에 따라 시뮬레이트 한다.

특정 레코드, 즉 개시 트랜잭션 레코드는 실제로 액티브 트랜잭션 테이블에 대한 개시 트랜잭션 연산과 마찬가지로 취급된다. 또, 갱신 로그 레코드는 실제로 더티 블록 테이블(900) 및 액티브 트랜잭션 테이블(1200)에 대한 블록 갱신 동작과 마찬가지로 취급되지만, 갱신 동작이 수행되지는 않는다. 그리고, 보상 로그 레코드는 NEXT 속성 1250의 값이 감분되는 것을 제외하고는 더티 블록 테이블(900) 및 액티브 트랜잭션 테이블(1200)에 대한 블록 갱신 동작과 사실상 마찬가지로 취급되지만, 갱신 동작이 수행되지는 않는다.

블록 기입 레코드의 경우, 블록이 더티 블록 테이블(900)로부터 제거된다. ABORT 트랜잭션 레코드의 경우에는 액티브 트랜잭션 테이블(1200)로부터 트랜잭션이 삭제된다. 또, 준비 트랜잭션 레코드의 경우 액티브 트랜잭션 테이블(1200)내의 트랜잭션 상태가 "준비 상태"로 설정된다. 또, 위임 트랜잭션 레코드의 경우에는 액티브 트랜잭션 테이블(1200)로부터 트랜잭션이 삭제된다.

액티브 트랜잭션 테이블(1200)내의 트랜잭션에 대한 HIGH 부속물(1240)을 재기억시키고자 하는 경우, ULOG가 그 ULOG에 기입된 최종 레코드의 RLSN 속성을 알 수 있도록 반드시 액세스 되어야 한다. 이러한 LSN은 HIGH 부속물(1240) 값이 된다. 또한, 체크 포인트에서의 HIGH 부속물(1240) 값이 사용될 수 있거나 혹은 갱신될 수 있다. 이러한 RLOG LSN은 ULOG 트랜잭션상에 이미 기록된 언두 레코드 동작의 발생을 방지하는데 사용될 수 있다. 단지 이러한 값을 갖는 RLOG 트랜잭션 레코드만이 언두 정보를 가질 필요가 있다.

NEXT 부속물(1250)은 (1) 로그 레코드가 갱신용인 경우 그 로그 레코드가 RLOG 트랜잭션에 기입된

최종 동작의 RLOG LSN, 혹은 (2) 트랜잭션을 위해 기입된 최종 CLR의 RLSN 부속물중 어느 하나이다. 따라서, NEXT 부속물(1250)이 RLOG의 분석 단계 동안 재기억될 수 있다. NEXT 부속물(1250)은 ULOG 레코드내의 RLSN 값을 거쳐, 수행될 다음 언두 레코드를 식별한다. 이것은 또한 언두 레코드에 기입된 CLR을 가짐으로써 이미 보상되었던 동작에 대한 언두 레코드이 발생을 방지하는데 사용될 수 있다. 따라서, 액티브 트랜잭션 테이블(1200)내의 NEXT 트랜잭션 부속물(1250)보다 더 큰 리두 트랜잭션 레코드 RLOG LSN은 리두 회복 단계 동안 기존 CLR가 인가될 경우에 언두가 행해지게 되는 것과 같이 그들에 대해 발생된 언두 정보를 갖는 것이 필요치는 않다.

2. 리두 단계

리두 단계에서는 재구성된 더티 블록 테이블(900)내의 더티로서 표시된 모든 블록이 캐쉬내에서 판독된다. 이러한 리두는 RLOG의 스캐닝과 겹쳐지는 부피로 행해질 수 있다.

일부 블록은 그들이 국부적인 리두로 수반될 필요가 있는지를 결정하도록 몇몇 노드에 의해 판독될 수 있으나, 그 노드들중 단지 하나만이 실제로 한 블록에 대한 리두를 수행하게 된다. 그러나, 이것은 블록 기입 레코드를 RLOG에 기입함으로써 거의 완전학 방지될 수 있다. 블록 기입 레코드가 실시될 필요는 없기 때문에, 이것이 필요하지 않을 때 하나의 블록이 때때로 영구 기억 장치로부터 판독될 것이다. 그렇지만, 이러한 판독에 대한 결점은 매우 적다. 모든 블록의 한가지 로그 변환이 영구 기억 장치내에 존재하므로, 단지 하나의 노드가 그 블록의 DSI와 동일한 BSI를 갖는 로그 레코드를 가질 수 있다. 이러한 노드는 블록에 대한 리두 처리를 독립적으로 수행하는 노드이다. 그러므로, 리두는 각각 자체 RLOG를 갖는 시스템의 분리 노드들에 의해 병렬로 수행될 수 있다. 여기서는 동시 제어가 전혀 요구되지 않는다.

리두 단계는 리두를 필요로 하는 더티 블록을 액세싱하고 또한 RLOG 레코드에서 표시된 변경을 포스팅함으로써 노드 캐쉬의 상태를 재구성한다. 결과적인 캐쉬은 크래쉬 발생시의 상태대로 더티 블록을 포함한다. 리두를 받을 블록들은 로크된다. 결과적으로 더티 블록 테이블(900) 및 액티브 트랜잭션 테이블(1200)은 동시에 재구성된다. 리두를 받을 블록들은 로크된다.

분석 단계후에 더티 블록 테이블(900)에 표시된 바와 같이 더티 블록의 리두 레코드만이 재실행될 필요가 있다. RLOG의 리두 스캔은 더티 블록 테이블(900)에 기록된 가장 빠른 회복 LSN(920)에서 시작한다. 이것은 리두의 안전 지점이다. 그러므로, 각 블록에 대한 모든 갱신은 각 블록이 영구 기억 장치에 기입되어 있기 때문에 리두 스켄에 확실히 포함된다.

전술한 바와 같이, RLOG 레코드를 그 대응하는 블록에 인가하고자할 때 두가지의 경우만이 발생할 수 있다. RLOG 레코드의 BSI가 블록의 DSI와 같지 않으면, 로그된 동작은 무시될 수 있다. RLOG 레코드의 BSI가 블록의 DSI와 동일하면 적당한 리두 동작이 수행된다.

리두 단계 방법은 반복되는 역사를 포함한다. 블록의 회복 LSN에 의해 표시된 RLOG 레코드로 시작하는 모든 갱신 RLOG 레코드는 나중에 비실행의 필요가 있는 트랜잭션에 속하는 것에 적용된다. 여기에서 사용되는 원리는 재실행될 동작에 있어서 원래 동작이 적용되는 상태와 정확히 동일한 상태로 블록에 적용되어야 한다는 것이다.

RLOG 레코드를 블록에 적용할때에 블록의 DSI는 재실행 동작을 위하여 ASI로 갱신된다. 노드는 RLOG 동작이 적용될 때 블록에서 적당한 로크를 요구한다. 리두는 다른 노드가 로크를 요구하지 않기 때문에 로크가 승인되기를 기다리지 않는다. 그러나, 요구된 로크는 언두의 개시전에 승인되어야 한다. 이것은 동시 제어가 언두 단계를 위하여 초기화되는 방법이다.

RLOG상에 로그된 정상 갱신이 리두를 필요로 하면 ULOG 레코드는 그것을 위해 승인되어야 한다. HIGH값과 NEXT 값 사이의 LSN을 갖는 트랜잭션의 모든 RLOG 리두 레코드는 그 레코드에 대해 발생된 언두 정보를 갖는다. 이 정보는 상기 레코드를 식별하는 RLSN 부속물을 갖는 ULOG 레코드를 포함한다.

동작이 재실행될 필요가 없으면 부적당하게 발생될 수 있는 선행 언두 레코드는 ULOG가 WAL 프로토콜을 토하여 이 동작을 위해 영구 기억 장치 및 ULOG 레코드에 기입되어 있기 때문에 디스카드된다. HIGH 부속물(1240)은 이때에 상기 레코드의 RLOG LSN에 의해 갱신될 수 있으며, 상기 레코드는 체크포인트가 취해지면 현재의 회복 처리가 실패할 경우 후속되는 회복기간동안 여분의 언두 레코드 발생을 감소시킨다.

각각의 트랜잭션에 있어서, 발생된 얻두 레코드는 트랜잭션의 ULOG 버퍼에 기억된다. 상기 레코드 및 그 ULOG 및 CLR상의 레코드들은 액티브 트랜잭션이 확실하게 롤백될 수 있게 한다. 따라서 리두 단계의 종단에서 모든 필요한 언두 로그 레코드가 존재한다.

3. 언두 단계

언두 회복은 N-로그이다. 따라서, 언두 회복 단계는 트랜잭션 롤백동안 필요로 하는 것과 동일한 방법으로 동시 제어를 필요로 한다. 복수의 노드는 동일 블록에 대하여 언두 변화를 행할 필요가 있다. 정상 데이터 베이스 동작은 정상 동작이 트랜잭션 중단과 동시에 처리될 수 있는 것과 마찬가지로 언두 단계가 시작되는 것으로 가정할 수 있다. 모든 적당한 로킹은 이 동작을 허용하기 위하여 적소에 배치된다. 이것은 모든 노드가 리두 단계를 완료할때까지 언두 단계를 개시 않음으로써 보장된다. 그러므로 리두 단계중에 어떤 노드에 의해 요구된 모든 로크는 언두 개시전에 적당한 노드에 의해 유진된다.

액티브 트랜잭션 테이블(1200)의 첫 번째의 모든 액티브 트랜잭션(준비된 트랜잭션이 아님)은 롤백된다. 언두 처리는 한 번의 예외를 가지고 명백히 중단된 트랜잭션을 롤링백할때와 정확히 동일하게 진행한다. 어떤 언두 레코드는 그 레코드가 리두 단계중에 재발생된 언두 버퍼에, 및 영구 기억 장치의 ULOG에 존재한다. 이들 중복된 언두 레코드는 검출 및 무시될 수 있다. 이것은 다음 언두 레코드를 얻기 위해 루틴내에 포함될 수 있으며, 따라서 크래쉬가 발생할대 트랜잭션 동작을 비실행하는

코드의 나머지는 시스템이 정상적으로 동작할때에 트랜잭션을 비실행할 필요가 있는 코드와 실질적으로 동일하다. 이들 자원 사이의 여분의 ULOG 레코드는 모든 언두 레코드가 RLOG 레코드의 LSN에 의해 식별되기 때문에 제거될 수 있다.

V. 결론

별도의 RLOG 및 ULOG의 사용은 언두 정보가 절대적으로 필요할때만 ULOG에 기억되게 함으로써 로깅 연산의 최적화를 가능하게 한다. 언제 이러한 필요성이 발생하는가에 대한 시험은 비위임된 트랜잭션에 관련된 변화를 필요로 하는 모든 정보가 기억되었는지 또는 재형성될 수 있는지에 대한 시험이다.

최적화 또한 회복중에 발생된 변화의 수를 계수함으로서 얻어질 수 있다.

기술에 숙련된 사람이라면 본 발명의 취지 및 범위에서 벗어남이 없이 여러 가지 수정 및 변형이 가능하다는 것을 알 수 있다. 예를들어, 제1도에 도시된 구성은 다르게 구성할 수 있으며 각각의 노드에 할당된 언두 및 리두 로그의 수는 변화시킬 수 있다. 본 발명은 첨부된 청구 범위에 포함되는 상기 모든 변형 및 수정을 포함하는 것으로 한다.

(57) 청구의 범위

청구항 1

복수의 노드 및 소구획으로 분리된 비소멸성 기억 매체를 포함하는데, 상기 복수의 노드가 트랜잭션에 의해 상기 두획들에 대한 변화를 형성하고 각각의 트랜잭션이 적어도 하나의 노드에 의해 적어도 하나의 구획에 형성된 일련의 변화를 포함하며, 각각의 트랜잭션은 그 트랜잭션 및 그 트랜잭션의 완료의 표시에 의해 영향을 받은 변화의 레코드가 기억 매체에 신뢰성 있게 기억된 경우에 위임되고 그렇지 않은 경우에는 비위임되는 데이터 처리 시스템에 있어서, 상기 다수 노드의 첫 번째 노드가, 적어도 한 구획의 복사본을 기억하는 메모리와 ; 메모리에 연결되고 메모리의 적어도 한 구획의 복사본에 대한 변화를 형성하는 처리 수단과 ; 메모리의 적어도 한 구획의 복사본에 대하여 처리 수단에 의해 형성된 변화의 순차 리스트를 포함하고 다른 비위임된 트랜잭션 및 각각의 버퍼에 대하여 첫 번째 노드에 의해 형성된 변화에 각각 대응하는 적어도 하나의 언두 버퍼와 ; 메모리의 적어도 한 구획의 복사본에 대하여 대응하는 노드의 처리 수단에 의해 형성된 변화의 순차 리스트를 포함하는 리두 버퍼와 ; 메모리에 연결되고 적어도 한 구획의 복사본을 기억매체에 다시 기억시키는 기억 수단과 ; 언두 버퍼 및 리두 버퍼에 연결되고, 비위임된 트랜잭션의 모든 변화의 효과가 제거되고 위임된 트랜잭션의 모든 변화의 효과가 재발생될 수 있도록 언두 버퍼 미치 리두 버퍼의 부분들을 기억 매체에 선택적으로 기억시키는 로그 관리 수단을 포함한 것을 특징으로 하는 데이터 처리시스템.

청구항 2

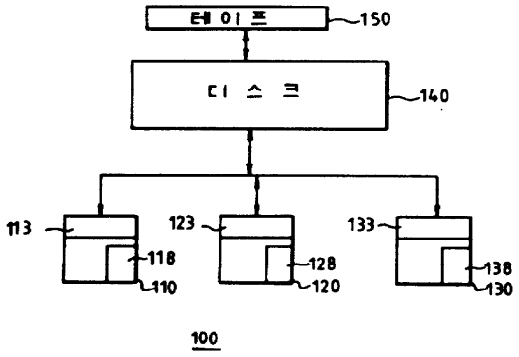
제1항에 있어서, 상기 로그 관리 수단은 기억 수단이 대응하는 비위임된 트랜잭션으로부터의 변화를 기억하기 전에 언두 버퍼의 내용을 기억 매체에 기억시키는 수단을 포함한 것을 특징으로 하는 데이터 처리시스템.

청구항 3

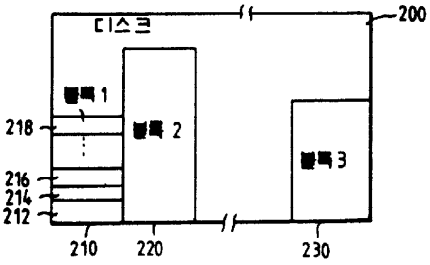
제1항에 있어서, 상기 로그 관리 수단은 제1트랜잭션이 완료되었음을 나타내는 표시뿐만 아니라 리두 버퍼에 기록된 제1트랜잭션의 모든 변화를 기억 매체에 신뢰성 있게 기억시키고, 이로써 제1트랜잭션이 위임되게 하는 수단을 포함한 것을 특징으로 하는 데이터 처리 시스템.

도면

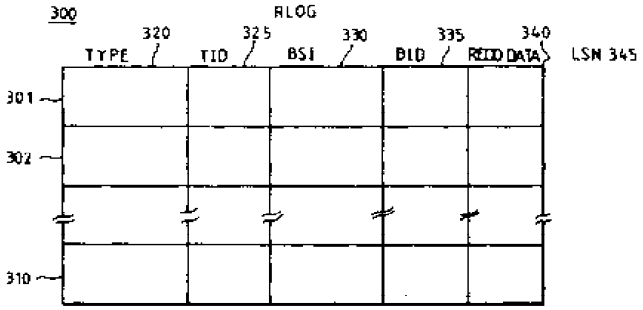
도면1



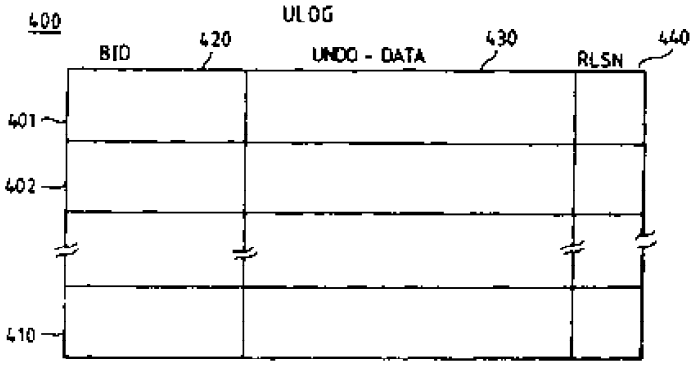
도면2



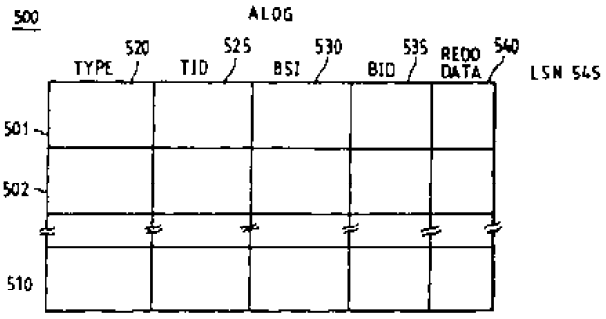
도면3



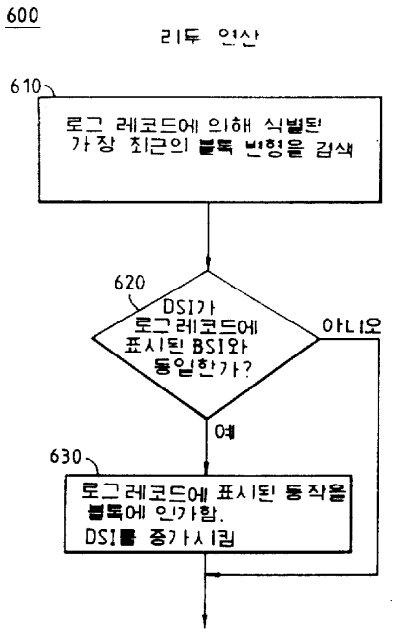
도면4



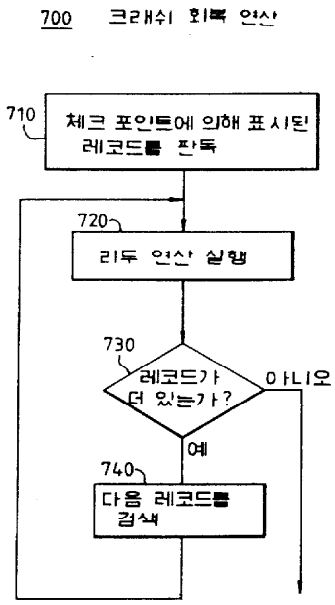
도면5



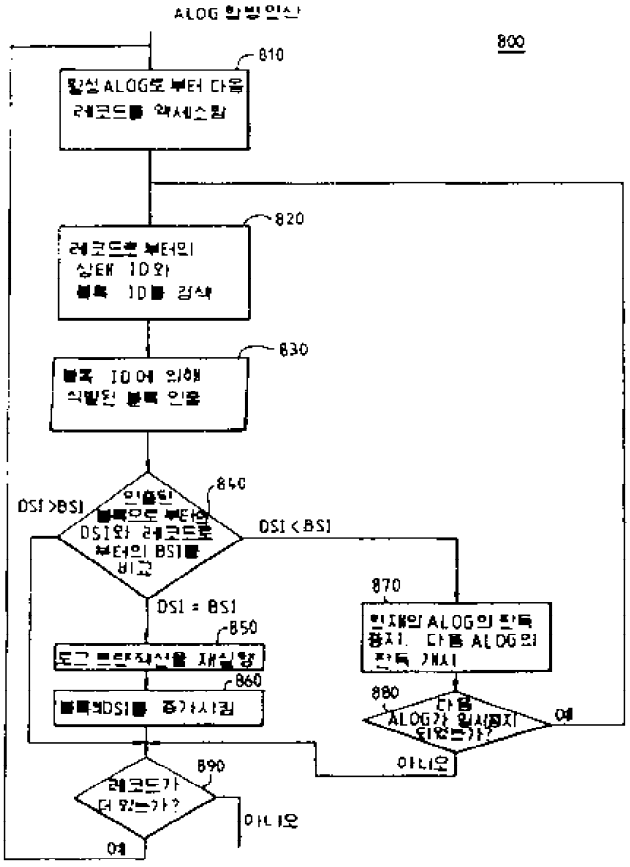
도면6


600
리두 연산
610
로그 레코드에 의해 식별된
가장 최근의 블록 변형을 검색
620
DSI가
로그레코드에
표시된 BSI와
동일한가?
아니오
예
630
로그레코드에 표시된 동작을
블록에 인가함.
DSI를 증가시킴

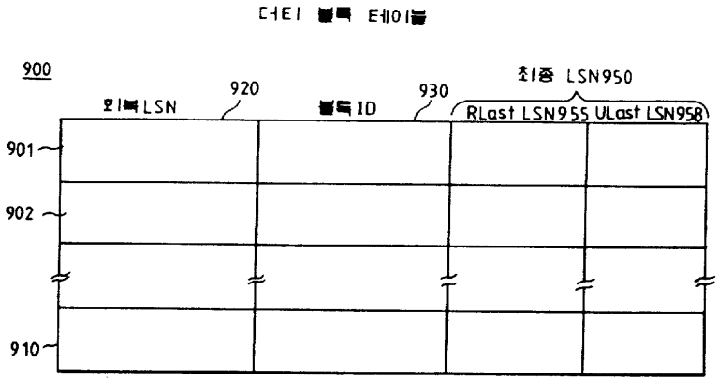
도면7


700 크래쉬 회복 연산
710
체크 포인트에 의해 표시된
레코드를 판독
720
리두 연산 실행
730
레코드가
더 있는가?
아니오
예
740
다음 레코드를
검색

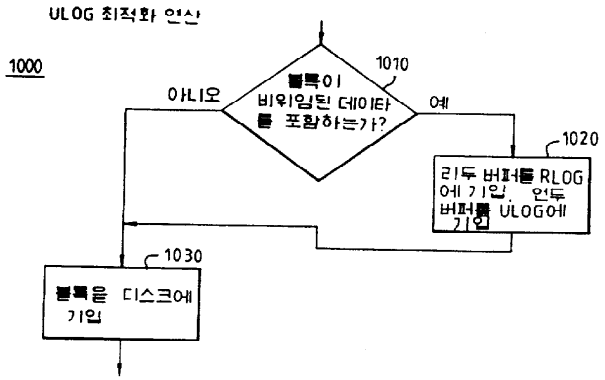
도면8


800
810
820
830
840
DSI>BSI
DSI<BSI
DSI=BSI
850
860
870
880
890
아니오
예

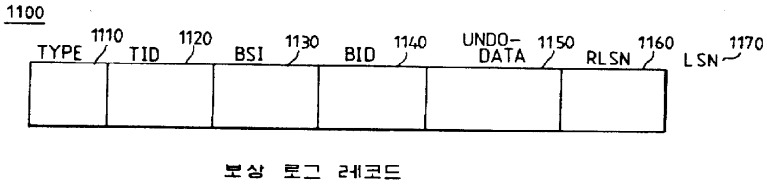
도면9


900
920
930
최종 LSN950
RLast LSN955 ULast LSN958
901
902
910

도면10

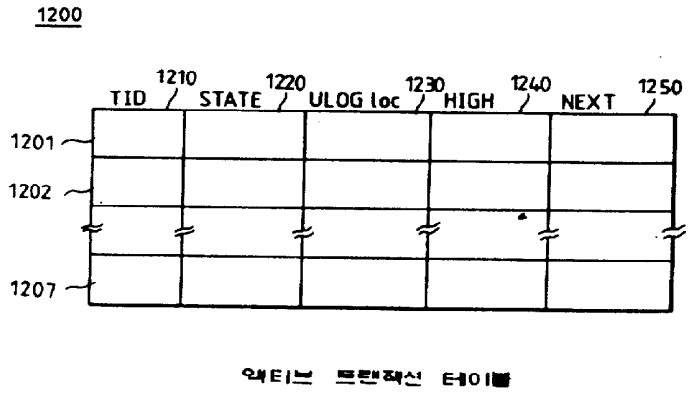


도면11



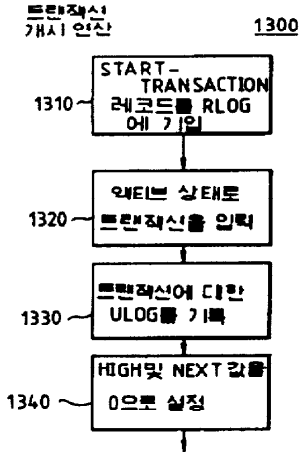
보상 로그 레코드

도면12

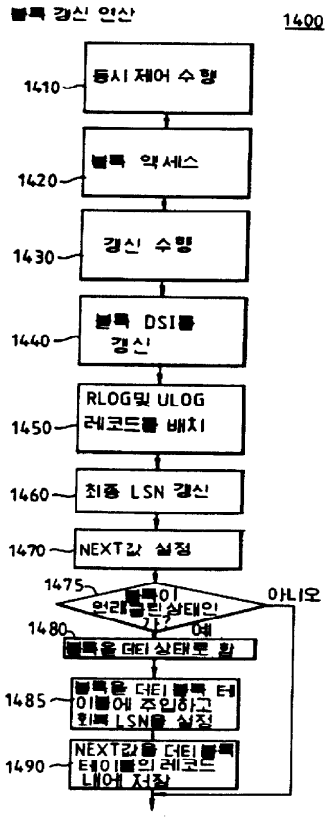


액티브 트랜잭션 테이블

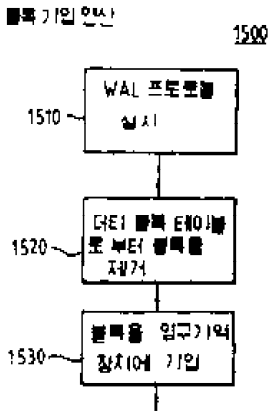
도면13



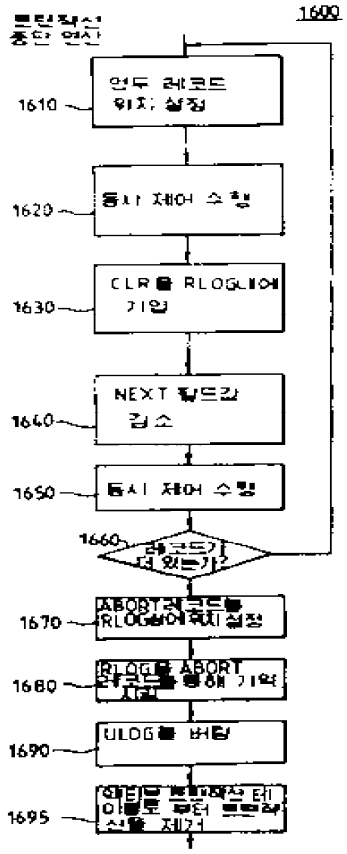
도면14


블록 갱신 연산
1400
1410
동시 제어 수행
1420
블록 액세스
1430
갱신 수행
1440
블록 DSI를 갱신
1450
RLOG및 ULOG 레코드를 배치
1460
최종 LSN 갱신
1470
NEXT값 설정
1475
아니오
예
1480
1485
1490

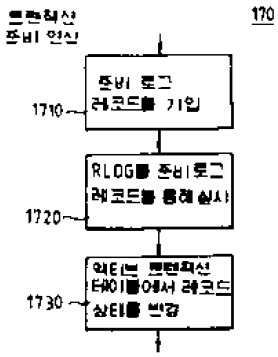
도면15


1500
1510
1520
1530

도면16


1600
1610
1620
1630
1640
1650
1660
1670
1680
1690
1695

도면17


170
1710
1720
1730

도면18