(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2017-0090128
(43) 공개일자 2017년08월07일

(51) 국제특허분류(Int. Cl.)
G06F 17/30 (2006.01)
(52) CPC특허분류
G06F 17/30336 (2013.01)
G06F 17/30091 (2013.01)
(21) 출원번호 10-2016-0010619
(22) 출원일자 2016년01월28일
심사청구일자 없음

(71) 출원인
한국전자통신연구원
대전광역시 유성구 가정로 218 (가정동)
(72) 발명자
이강우
대전광역시 유성구 어은로 57 한빛아파트 110동 1504호
(74) 대리인
특허법인지명

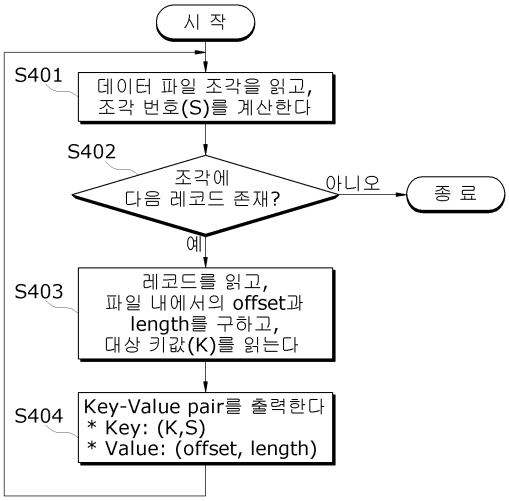
전체 청구항 수 : 총 1 항

(54) 발명의 명칭 **하둡 환경에서 맵리듀스 기반 데이터 처리 성능 향상을 위한 인덱스 구축 및 활용방법**

(57) 요 약

본 발명은 맵리듀스 기반 데이터 처리 방법에 관한 것으로, 특히, 하둡(Hadoop) 환경에서 맵리듀스(MapReduce) 방식으로 빅데이터를 효과적으로 처리하기 위한 보조 인덱스(Secondary Index)를 구축하고, 이를 맵리듀스 기반 데이터 처리에 활용하는 하둡 환경에서 맵리듀스 기반 데이터 처리 성능 향상을 위한 인덱스 구축 및 활용방법에 관한 것이다.

대 표 도 - 도4



(52) CPC특허분류
G06F 17/30318 (2013.01)
G06F 17/30946 (2013.01)

이 발명을 지원한 국가연구개발사업
과제고유번호 1615007697
부처명 국토교통부
연구관리전문기관 국토교통과학기술진흥원
연구사업명 국토공간정보연구사업
연구과제명 공간 빅데이터 저장 관리 인프라 기술 개발
기 여 율 1/1
주관기관 한국전자통신연구원
연구기간 2015.02.14 ~ 2016.02.13

명 세 서

청구범위

청구항 1

조각을 처리하는 각 맵퍼는 자신에게 할당된 파일 조각 정보를 읽고, 각 조각 번호(S)를 계산하고, 해당 레코드의 offset과 length를 구한 후, 지정된 키 컬럼의 값을 얻어 키 값(K)을 획득하여 offset, length 및 키 값(K)을 이용하여 reducer로 보낼 중간 결과 값을 출력하는 단계; 및

상기 리듀서는, 상기 맵퍼에서 입력되는 Key와 Value 리스트(K, S: list(offset, length))를 읽어 key와 value를 조각내 하둡 인덱스 파일에 저장하고, 각 레코드 구간들의 리스트에서 가장 작은 sffset 구간과 가장 큰 옵센 구간을 구해 전체 offset과 length를 계산한 후, 조각 수준 하둡 인덱스 파일에 저장하는 단계를 포함하는 하둡 환경에서 맵리듀스 기반 데이터 처리 성능 향상을 위한 인덱스 구축방법.

발명의 설명

기 술 분 야

[0001] 본 발명은 맵리듀스 기반 데이터 처리 방법에 관한 것으로, 특히, 본 발명은 하둡(Hadoop) 환경에서 맵리듀스(MapReduce) 방식으로 빅데이터를 효과적으로 처리하기 위한 보조 인덱스(Secondary Index)를 구축하고, 이를 맵리듀스 기반 데이터 처리에 활용하는 하둡 환경에서 맵리듀스 기반 데이터 처리 성능 향상을 위한 인덱스 구축 및 활용방법에 관한 것이다.

배 경 기 술

[0002] 일반적으로, 맵리듀스(MapReduce) 기반 작업 수행은 대규모 데이터를 처리하는 작업을 여러 개의 작은 단위로 세부 작업을 분할하여 이를 여러 컴퓨터에 적재하여 병렬적으로 처리하고(Map 작업), 그 수행 결과를 합하여 최종 결과물을 생성하는 방식으로 이루어진다.

[0003] 이러한 맵리듀스 작업은 하나 작업을 여러 대의 일반 컴퓨터를 이용하여 병렬적으로 수행하기 때문에, 대규모 데이터 처리가 가능하게 되어 최근 빅데이터 분석, 기계 학습 분야 등에서 활발히 사용되고 있다. 뿐만 아니라 보다 복잡한 작업은 하나의 작업을 다시 여러 단위 맵리듀스 작업으로 분할하고 이를 순차 또는 병렬적으로 실행시키는 복합 맵리듀스 작업으로 처리하고 있다. 현재 이러한 방식의 작업 처리를 위해 Pig, Hive와 같이 다양한 복합 맵리듀스 작업 처리에 관련 기술들이 등장하고 있다.

[0004] 일반적인 맵리듀스 빅데이터 처리 방법은, 소셜 미디어 로그 데이터 또는 센서의 측정 값과 같이 지속적으로 누적되는 대용량의 데이터 전체를 읽어 분석하는 분야에 주로 활용되었으나, 대규모의 컴퓨터를 활용한 효과적인 빅데이터 처리 방법으로 인해 그 활용 분야가 점차 기존의 DBMS 응용 분야로 확대되고 있다. 특히, 데이터의 변경이 아주 빈번하지 않은 데이터 활용 분야의 경우, 맵리듀스 방식을 활용하면 대규모 데이터 처리시 기존 DBMS를 활용한 방법에 대해 우수한 성능을 보일 것으로 기대한다.

[0005] 기존 DBMS에서는 대규모의 데이터를 신속하게 처리하는 방법으로 인덱스를 활용하는 방법을 사용한다. 인덱스를 활용하여 데이터 전체를 검색하지 않고 분석에 필요한 데이터만을 적은 I/O 횟수로 접근할 수 있어 데이터 처리 성능이 향상된다.

[0006] 맵리듀스 방식에 기반한 데이터 처리 방식에서는 파일에 저장된 데이터를 파일조각(FileSplit)(이하, 조각(split)으로 칭한다) 단위로 각각 읽어 들어 순차적으로 처리한다. 도 1은 3개의 조각으로 구성된 파일의 예를 나타낸 도면이다.

[0007] 도 1에 도시된 바와 같이, 3개의 조각으로 구성된 파일 전체에 15개의 레코드가 저장되어 있는 것을 확인할 수 있다.

[0008] 일반적으로 조각의 크기는 파일 별로 미리 지정된 크기로 고정되나, 레코드의 크기는 실제 저장되는 데이터의

크기에 따라 그 길이가 다를 수 있다.

[0009] 그러나 기존의 인덱스는 조각 단위의 데이터를 처리하는 하둡 파일 시스템과는 달리, 레코드 단위의 데이터를 직접 임의적으로 접근하는 것으로 가정하기 때문에, 기존의 인덱스 구조 및 활용 방식을 그대로 적용하는데 많은 문제가 있다.

발명의 내용

해결하려는 과제

[0010] 따라서, 본 발명은 상기한 문제점들을 해결하기 위한 것으로, 본 발명의 목적은, 하둡 환경의 HDFS 파일 시스템에 저장된 데이터를 MapReduce 처리 방식을 통해 미리 저장된 키 값을 갖는 데이터를 접근할 때, 신속하게 해당 키 값의 레코드를 접근하도록 하는 인덱스 구조, 인덱스 생성 방법 및 MapReduce 처리 방식의 데이터 접근시 활용하는 방법을 제공함에 있다.

과제의 해결 수단

[0011] 상기한 목적을 달성하기 위한 본 발명에 따른 하둡 환경에서 맵리듀스 기반 데이터 처리 성능 향상을 위한 인덱스 구축방법에 따르면, 조각을 처리하는 각 맵퍼는 자신에게 할당된 파일 조각 정보를 읽고, 각 조각 번호(S)를 계산하고, 해당 레코드의 offset과 length를 구한 후, 지정된 키 컬럼의 값을 얻어 키 값(K)을 획득하여 offset, length 및 키 값(K)을 이용하여 reducer로 보낼 중간 결과 값을 출력하는 단계; 및 상기 리듀서는, 상기 맵퍼에서 입력되는 Key와 Value 리스트(K, S: list(offset, length))를 읽어 key와 value를 조각내 하둡 인덱스 파일에 저장하고, 각 레코드 구간들의 리스트에서 가장 작은 sffset 구간과 가장 큰 옵센 구간을 구해 전체 offset과 length를 계산한 후, 조각 수준 하둡 인덱스 파일에 저장하는 단계를 포함할 수 있다.

발명의 효과

[0012] 본 발명에 따르면, 맵리듀스 방식의 데이터 처리과정의 성능을 높이기 위한 맵리듀스 용 인덱스 구조, 맵리듀스 기반의 인덱스 생성 방법, 그리고 인덱스를 활용한 맵리듀스 방식의 데이터 처리 방법을 제안한 것으로서, 다음과 같은 효과를 가질 수 있다.

[0013] 첫 째, 파일 조각 단위의 작업을 수행하는 맵리듀스 방식의 데이터 처리에서 활용할 수 있는 인덱스 구성 및 활용 방법을 제공하여 데이터 처리 성능 즉, 데이터 처리 성능은 불필요한 디스크 블록 I/O 횟수를 줄일 수 있으며, 파일 조각 내 불필요한 레코드의 디코딩 시간을 줄일 수 있다.

[0014] 둘 째, 일반적으로 두 개의 파일 블록(block) 사이에 한 개의 레코드가 겹치는 경우가 높기 때문에 파일 조각을 읽는 경우가 많은 경우 두 번의 디스크 블록 I/O가 발생되지만, 본 발명은 파일의 모든 조각에 키 값에 해당하는 레코드가 모두 존재하는 경우에도 조각의 첫 번째 레코드와 마지막 레코드 모두 주어진 키 값의 레코드인 경우를 제외하면 한번의 디스크 블록 I/O를 줄일 수 있다.

[0015] 세 째, 인덱스를 활용하면 조각 내 관련 레코드가 포함된 영역만을 메모리에 적재하기 때문에 메모리 사용량을 줄일 수 있다.

[0016] 네 째, 맵리듀스 방식의 인덱스 생성 방법을 제공하기 때문에 대규모 데이터 파일에 대한 인덱스 생성을 신속하게 완료할 수 있다.

도면의 간단한 설명

[0017] 도 1은 3개의 조각으로 구성된 파일의 예를 나타낸 도면이다

도 2는 본 발명의 실시예에 따라, 검색 대상이 되는 카 값에 따라 파일내의 대상 레코드의 분포 위치를 나타낸 도면.

도 3은 본 발명의 실시예에 따른 활용 인덱스 구조를 나타낸 도면.

도 4는 본 발명에 따른 인덱스 파일 생성 방법에서 조각 파일을 매핑하는 과정에 대한 동작 플로우챠트를 나타낸 도면.

도 5는 도 4의 과정에 의해 매핑된 데이터 조각 파일을 리듀싱(Reducing)하는 과정을 나타낸 동작 플로우챠트.

도 6은 본 발명에 따른 인덱스 파일을 활용한 데이터 접근에 있어 인덱스를 통한 맵리듀스 방식을 위한 파일 조각 과정에 대한 동작 플로우챠트.

도 7은 본 발명에 따른 인덱스 기반 파일 접근을 위한 파일 조각내 레코드 적재과정을 나타낸 동작 플로우챠트.

발명을 실시하기 위한 구체적인 내용

[0018] 본 발명의 이점 및 특징, 그리고 그것들을 달성하는 방법은 첨부되는 도면과 함께 상세하게 후술되어 있는 실시예들을 참조하면 명확해질 것이다. 그러나 본 발명은 이하에서 개시되는 실시 예들에 한정되는 것이 아니라 서로 다른 다양한 형태로 구현될 것이며, 단지 본 실시 예들은 본 발명의 개시가 완전하도록 하며, 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자에게 발명의 범주를 용이하게 이해할 수 있도록 제공되는 것이며, 본 발명은 청구항의 기재에 의해 정의된다. 한편, 본 명세서에서 사용된 용어는 실시 예들을 설명하기 위한 것이며 본 발명을 제한하고자 하는 것은 아니다. 본 명세서에서, 단수형은 문구에서 특별히 언급하지 않는 한 복수형도 포함한다. 명세서에서 사용되는 "포함한다(comprises)" 또는 "포함하는(comprising)"은 언급된 구성요소, 단계, 동작 및/또는 소자 이외의 하나 이상의 다른 구성요소, 단계, 동작 및/또는 소자의 존재 또는 추가를 배제하지 않는다.

[0019] 이하, 첨부한 도면을 참조하여 발명에 따른 하둡 환경에서 맵리듀스 기반 데이터 처리 성능 향상을 위한 인덱스 구축 및 활용방법에 대하여 상세하게 살펴보기로 하자.

[0020] 도 2는 검색 대상이 되는 키 값에 따라 파일 내의 대상 레코드의 분포 위치를 나타낸 도면이다. 도 2에서는 설명의 편의상 모든 레코드의 크기는 모두 10byte로 가정한다.

[0021] 도 2에 도시된 바와 같이, 키 값이 k7인 레코드를 접근하는 경우, 기존의 맵리듀스 방식에서는 전체 조각들을 읽고, 각 조각에 저장된 모든 15개의 레코드를 디코딩(Decoding)해야 하지만, 인덱스를 활용하는 경우에는 하나의 조각(몦plit3)만 읽고 4번의 레코드만을 디코딩하면 되기 때문에 데이터 접근 시간을 줄일 수 있다.

[0022] 한편, 키 값이 k3인 경우는 인덱스가 없는 방법과 같이 총 3번의 조각 I/O가 유발되지만, 총 5번의 레코드 디코딩 시간만 소요되기 때문에 약간의 성능 향상을 보이게 된다.

[0023] 본 발명에서는 인덱스 대상 파일과 키에 대상되는 컬럼에 대해 총 2개의 인덱스 파일을 생성하여 사용하는 방법을 설명한다. 첫 번째로는 사용되는 2개의 인덱스 파일의 구조 및 내용에 대해 설명한다.

[0024] 도 3은 본 발명에서 사용하는 두 개의 인덱스 파일의 구조 및 내용을 보여주는 도면으로서, 각 키 값에 대해 해당 키 값을 포함하는 레코드가 위치한 조각의 위치를 저장하는 "조각 수준 인덱스(split-level index: slx)"를 보여주는 것이다.

[0025] 도 3 (a)에 도시된 바와 같이, 조각 수준 인덱스 파일은 일반 HDFS 파일과 같이 (key, value) pair 형식의 레코드를 갖는 파일로 구성되어 있고, key 부분에는 인덱스 대상 칼럼의 키 값이 기록되고, value 부분에는 해당 키 값을 포함한 조각의 번호, 그리고 조각 내 해당 인덱스 키 값을 갖는 첫 번째 레코드 (R_{first})까지의 offset, 그리고 R_{first}와 조각 내 마지막 레코드 사이의 길이가 기록된다. 예를 들어 "k1" 키 값에 해당하는 레코드들은 모두 1번, 2번 조각에 있고, 1번 조각에는 r1, r4, r5 레코드가 해당하기 때문에 offset은 r1의 offset에 해당하는 0, 그리고 length는 r1과 r5까지의 길이에 해당하는 50이 기록된다. 그리고, 2번 조각에는 r8이 해당하므로 offset은 70, 그리고 길이는 10이 된다.

[0026] 도 3 (b)와 같은 두번째 인덱스 파일은 "조각 내 인덱스(intra-split index: isx)"를 보여준 것이다.

[0027] 조각 내 인덱스도 (key, value) pair 형식의 레코드로 갖는 파일을 구성되어 있다. Key 부분은 각 키 값과 해당 키 값을 갖는 조각 번호로 구성되고, value 부분은 해당 조각 내의 키 값을 갖는 레코드들의 위치(offset과 length)를 기록한다. 예를 들어, "k1" 키 값에 해당하는 레코드들은 모두 1번, 2번 조각에 위치하기 때문에, (k1, 1)과 (k1, 2)가 인덱스 파일의 레코드의 키로 존재하고, 조각 1번과 조각 2번 내에서의 해당 레코드들(r1, r4, r5, r8)의 위치 정보가 기록되는 것을 확인할 수 있다.

[0028] 이와 같은 구조를 갖는 인덱스 파일을 생성하는 방법에 대하여 도 4와 도 5를 참조하여 설명하기로 한다.

[0029] 먼저, 본 발명에서의 인덱스 파일 생성은 인덱싱의 대상이 되는 파일에 속한 모든 레코드를 리드하고, 리드된 각 레코드의 해당 컬럼 값의 키 값을 읽어 생성되는 것으로 이러한 과정은 맵리듀스 방식으로 수행된다. 즉, 데이터 파일을 조각 단위로 분할하고, 분할된 각각의 조각은 대응되는 각각의 맵퍼(미도시, Mapper)를 통해 1차

처리되고, 1차 처리된 결과를 다시 키 값에 따라 정렬하여 리듀서(미도시, Reducer)에서 2차로 처리함으로서 인덱스 파일이 생성되는 것이다.

[0030] 이와 같은 인덱스 생성과정에서, 맵퍼와 리듀서에서 수행되는 처리 과정에 대하여 도 4 및 도 5를 참조하여 각각 단계적으로 살펴보기로 한다.

[0031] 도 4는 인덱스 생성 과정 중 Mapper에서 수행하는 과정을 나타낸 도면이다.

[0032] 도 4에 도시된 바와 같이, 조각을 처리하는 각 맵퍼는 자신에게 할당된 파일 조각 정보를 읽고, 각 조각 번호(S)를 계산한다(S401). 여기서, 상기 조각 번호를 계산하는 방법은 제공되는 파일 조각의 offset을 맵리듀스 과정에서 사용하는 조각 크기로 나눈 몫으로 계산할 수 있다.

[0033] 이와 같이, 조각 번호가 계산되면 해당 조각에 포함되는 다음 레코드가 존재하는지 판단하고(S402), 판단 결과, 다음 레코드가 존재하는 경우, 할당된 조각에 포함된 모든 레코드에 대해 다음과 같은 작업을 수행한다.

[0034] 먼저, 다음 레코드를 리드하고, 리드한 레코드의 offset과 length를 구한다. 여기서, 상기 offset과 length의 계산 방법은 레코드의 인코딩 방식에 따라 그 방법이 다를 수 있다.

[0035] 상기와 같이 해당 레코드의 offset과 length가 구해지면, 지정된 키 컬럼의 값을 얻어 키 값(K)을 획득한다(S403).

[0036] 이어, 상기 S401단계에서 구한 조각번호(S)와 S403 단계에서 구해진 offset, length 및 키 값(K)을 이용하여 reducer로 보낼 중간 결과 값을 출력한다(S404). 여기서, 중간 결과 값은 key-value로 구성되며, key는 (K, S)로 구성되고, value는 (offset, length)로 구성될 수 있다.

[0037] 이와 같이, 과정을 모든 레코드에 대하여 수행하고, 모든 레코드에 대한 작업이 끝나면 맵퍼 작업은 종료된다.

[0038] 한편, 상기와 같이 맵퍼 과정이 완료되어 출력된 중간 결과값을 이용하여 리듀서에서 리듀싱하는 과정에 대하여 도 5를 참조하여 살펴보자.

[0039] 도 5는 인덱스 생성 과정 중 리듀서에서 수행하는 작업 과정을 나타낸 도면이다.

[0040] 먼저, 리듀서의 입력은 상기한 도 4에서와 같이 맵퍼에서 생성한 (K, S: offset, length) 값으로부터 생성된다. 여기서, 맵퍼에 의해 생성된 모든 데이터 값들을 key에 해당하는 (K, S) 별로 그룹핑되어 key-value 형태로 제공되고, 이때 key 값은 (K, S)이고, value는 해당 (K, S)를 갖는 (offset, length)들의 리스트로 구성된다(S501).

[0041] 이어, 다음으로 생성되는 Key와 Value 리스트가 존재하는 경우(S502), 아래와 같은 과정을 수행한다. 즉, 리듀서 과정에서는 상기 설명한 2가지 인덱스(조각 수준 인덱스, 조각내 인덱스) 모두 생성한다. 리듀서는 입력으로 앞서 언급한 바와 같이 각각의 입력 데이터(K, S: list(offset, length))에 대해 다음과 같은 작업을 수행한다.

[0042] 먼저, 입력된 (K, S: list(offset, length))를 조각내 하둡 인덱스 파일에 저장한다(S503).

[0043] 이어, 각 레코드 위치 (offset, length) 리스트에서 offset 값이 가장 작은 구간(record 1; R_{first})과 offset값이 가장 큰 구간(record 2 ; R_{last})를 구한 후, 이 중 record1의 offset R_{first}값과, record2의 offset R_{last} - record1의 offset R_{first} + record2의 length로 계산된 전체 length 값을 계산하여(S504) 조각 수준 하둡 인덱스 파일에 저장한다(S505). 이때 저장될 데이터의 key는 K가 되고, value는 S, offsetR, lengthR로 구성한다

[0044] 모든 입력 데이터에 대한 작업이 끝나면 reducer 작업은 종료된다.

[0045] 한편, 인덱스 파일을 활용한 데이터 파일 접근은 인덱스가 생성된 컬럼 값을 활용한 데이터 접근시에 사용된다. 이하, 맵리듀스 방식을 통한 데이터 접근시 활용하는 방법에 대하여 살펴보자.

[0046] 먼저, 맵리듀스 방식에서 데이터를 접근하는 방식은 입력 파일을 일정크기의 조각으로 분할하여 복수의 맵퍼에게 할당하는 작업과, 각 맵퍼에서 할당된 파일 조각에서 레코드를 디코딩하여 읽어 처리하는 과정, 그리고 맵퍼에 의해 처리된 결과를 재구성하여 리듀서에게 전달하면 이를 리듀서가 처리하는 과정을 수행하게 된다. 이하, 도 6을 참조하여 인덱스를 활용하여 신속하게 입력 파일을 분할한 후, 파일 조각들을 구성하는 방법과 분할된 조각에서 처리할 레코드를 읽는 과정을 설명한다.

[0047] 도 6는 맵리듀스 방식에서 입력 파일을 일정크기의 조각으로 분할하는 과정을 나타내 도면이다.

[0048] 도 6에 도시된 바와 같이, 먼저 질의 대상이 되는 인덱스 키 값(K)를 받으면(S601), 조각수준 인덱스 파일(slx)를 읽어 각 key-value 데이터를 접근한다(S602).

[0049] 그리고, 데이터의 key 값과 입력되는 질의 대상이 되는 인덱스 키 값(K)를 비교하여 상호 다를 경우, 다음 key-value 데이터를 접근하고, 동일한 경우에는 해당 value (S, offset, length)를 이용하여 파일 조각을 생성한다(S603~S605).

[0050] 도 7은 상기 생성된 각 파일 조각에서 해당 키 값을 갖는 레코드를 구하는 과정을 나타낸 도면이다.

[0051] 도 7의 과정은 파일 조각을 할당 받은 맵퍼에 의해 수행되며, 먼저 하둡 설정 정보를 통해 검색 대상의 키 값(K)과 대상 FileSplit을 입력 받는다(S701).

[0052] 맵퍼는 주어진 파일 조각으로부터 해당 조각의 번호(S)를 계산한다(S702). 즉, 입력받은 FileSplit을 포함하는 데이터 파일의 FileSplit 번호(S)를 계산한다. 여기서, 조각 번호를 계산하는 방법은 파일 조각의 offset을 설정된 조각 크기로 나눈 몫으로 정의된다.

[0053] 이어, 조각내 인덱스 파일(isx)를 읽어 주어진 키 값 K와 조각 번호 S를 key로 하여 해당 key-value 데이터를 접근하여 해당 레코드 번호 리스트를 검색한다(S703).

[0054] 그리고, 파일 조각에 해당하는 파일 영역을 읽는다(S704). 상기 S703 단계에서 획득된 key-value 데이터의 value 값은 주어진 키 값을 갖는 레코드들의 위치 정보이므로 이 정보를 이용하여 해당 레코드를 디코딩하여 적재시킨다(S705).

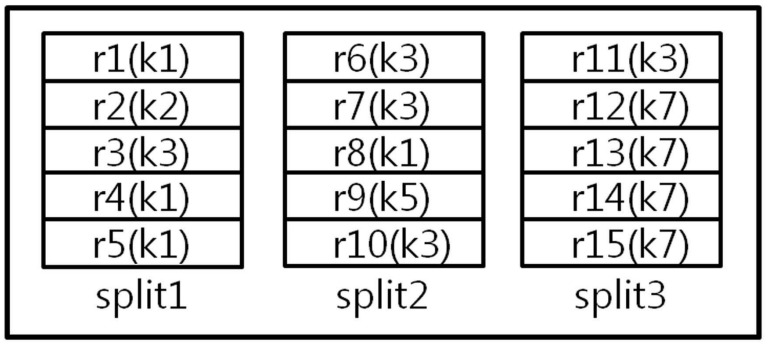
[0055] 앞서 설명한 방식을 통해 맵리듀스 방식을 통해 주어진 키 값을 갖는 레코드를 불필요한 파일 I/O와 불필요한 레코드 디코딩 과정 없이 접근하여 처리할 수 있게 되는 것이다.

[0056] 본 발명에 따른 하둡 환경에서 맵리듀스 기반 데이터 처리 성능 향상을 위한 인덱스 구축 및 활용방법을 실시예에 따라 설명하였지만, 본 발명의 범위는 특정 실시 예에 한정되는 것은 아니며, 본 발명과 관련하여 통상의 지식을 가진 자에게 자명한 범위 내에서 여러 가지의 대안, 수정 및 변경하여 실시할 수 있다.

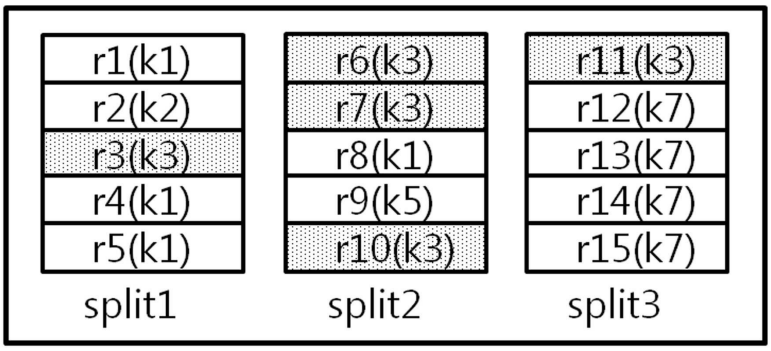
[0057] 따라서, 본 발명에 기재된 실시 예 및 첨부된 도면들은 본 발명의 기술 사상을 한정하기 위한 것이 아니라 설명하기 위한 것이고, 이러한 실시 예 및 첨부된 도면에 의하여 본 발명의 기술 사상의 범위가 한정되는 것은 아니다. 본 발명의 보호 범위는 청구범위에 의하여 해석되어야 하며, 그와 동등한 범위 내에 있는 모든 기술 사상은 본 발명의 권리 범위에 포함되는 것으로 해석되어야 할 것이다.

도면

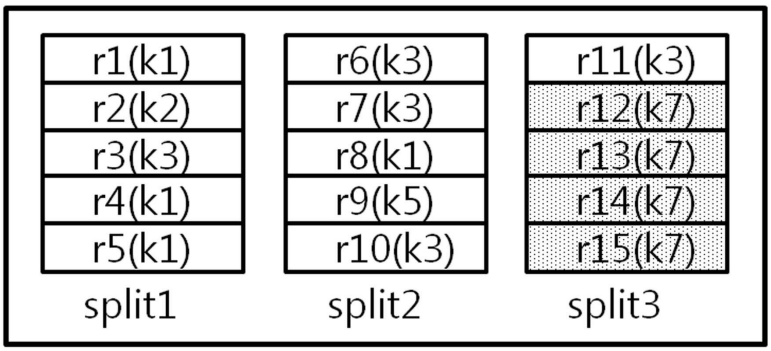
도면1



도면2



하둡 파일 (key=k3인 경우)



하둡 파일 (key=k7인 경우)

도면3

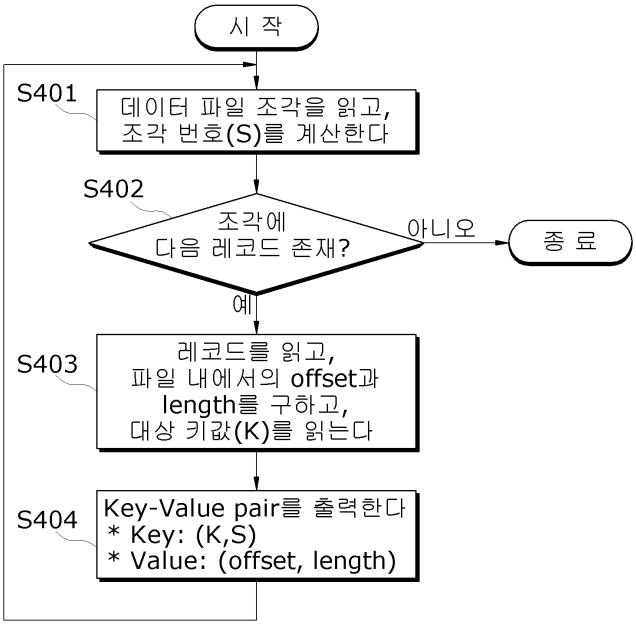
하둡 파일 Key (키 값)	하둡 파일 Value (조각 영역 리스트)
k1	(1,0,50), (2,70,10)
k2	(1,10,10)
k3	(1,30,10), (2,50,50), (3,100,10)
k5	(2,80,10)
k7	(3,110,40)

(조각수준 인덱스: slx)

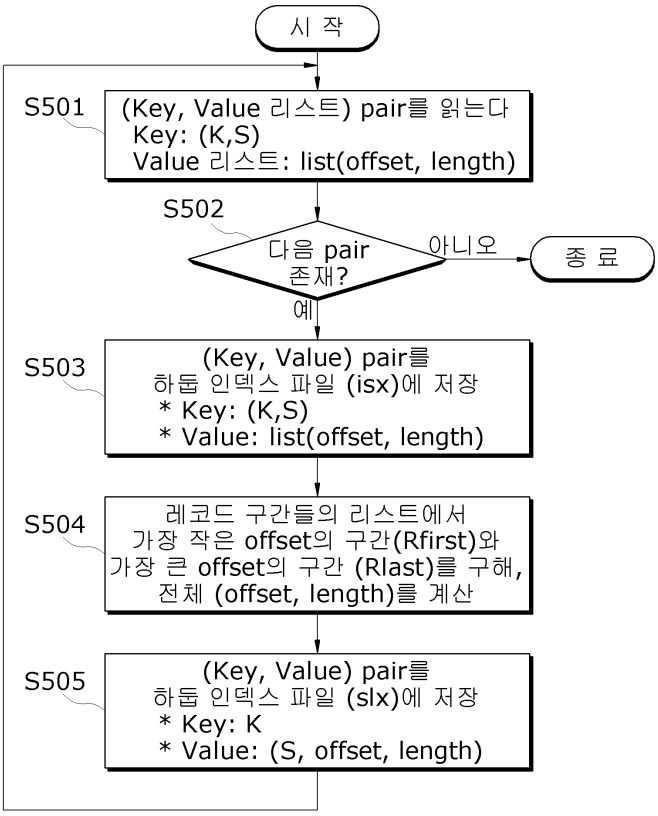
인덱스 파일 Key (키 값, 조각번호)	인덱스 파일 Value (조각내 레코드 위치 리스트)
(k1, 1)	(0,10), (30,10), (40,10)
(k1, 2)	(70,10)
(k2, 1)	(10,10)
(k3, 1)	(20,10)
(k3, 2)	(50,10), (60,10), (90,10)
(k3, 3)	(100,10)
(k5, 2)	(80,10)
(k7, 3)	(110,10), (120,10), (130,10), (140,10)

(조각내 인덱스: isx)

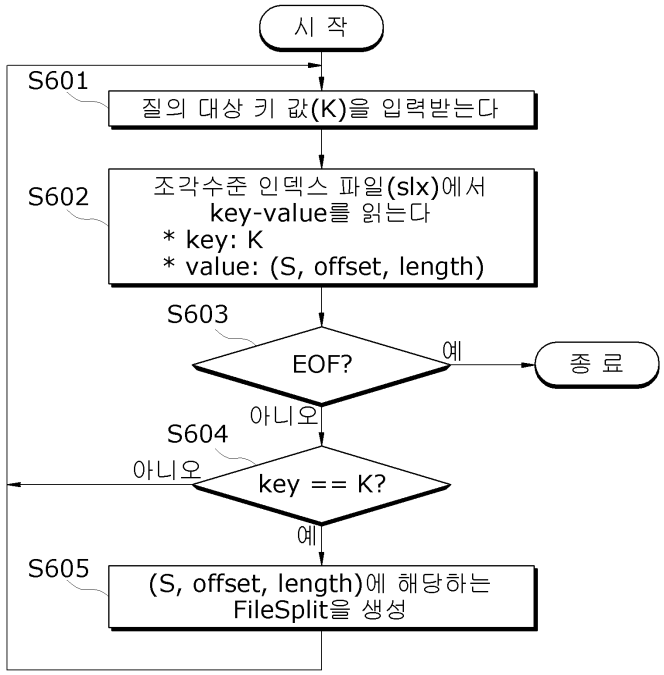
도면4


시 작
S401
데이터 파일 조각을 읽고,
조각 번호(S)를 계산한다
S402
조각에
다음 레코드 존재?
아니오
종 료
예
S403
레코드를 읽고,
파일 내에서의 offset과
length를 구하고,
대상 키값(K)를 읽는다
S404
Key-Value pair를 출력한다
* Key: (K,S)
* Value: (offset, length)

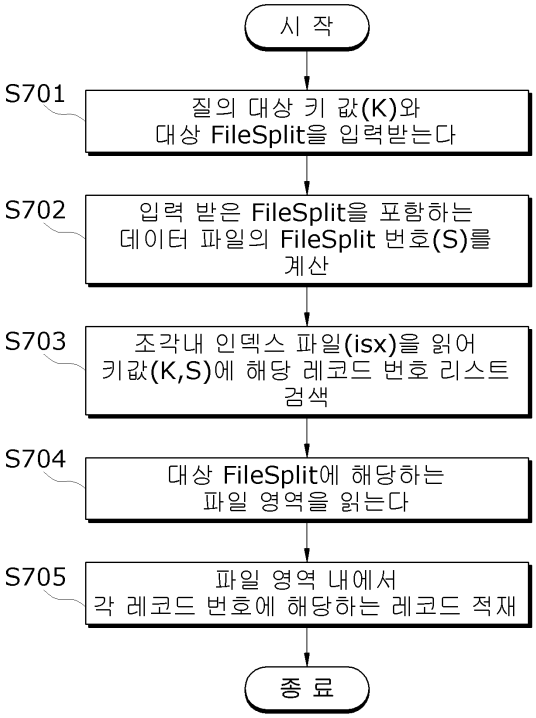
도면5


시 작
S501
(Key, Value 리스트) pair를 읽는다
Key: (K,S)
Value 리스트: list(offset, length)
S502
다음 pair
존재?
아니오
종 료
예
S503
(Key, Value) pair를
하둡 인덱스 파일 (isx)에 저장
* Key: (K,S)
* Value: list(offset, length)
S504
레코드 구간들의 리스트에서
가장 작은 offset의 구간(Rfirst)와
가장 큰 offset의 구간 (Rlast)를 구해,
전체 (offset, length)를 계산
S505
(Key, Value) pair를
하둡 인덱스 파일 (slx)에 저장
* Key: K
* Value: (S, offset, length)

도면6


시 작
S601
질의 대상 키 값(K)을 입력받는다
S602
조각수준 인덱스 파일(slx)에서
key-value를 읽는다
* key: K
* value: (S, offset, length)
S603
EOF?
예
종 료
아니오
S604
아니오
key == K?
예
S605
(S, offset, length)에 해당하는
FileSplit을 생성

도면7


시 작
S701
질의 대상 키 값(K)와
대상 FileSplit을 입력받는다
S702
입력 받은 FileSplit을 포함하는
데이터 파일의 FileSplit 번호(S)를
계산
S703
조각내 인덱스 파일(isx)을 읽어
키값(K,S)에 해당 레코드 번호 리스트
검색
S704
대상 FileSplit에 해당하는
파일 영역을 읽는다
S705
파일 영역 내에서
각 레코드 번호에 해당하는 레코드 적재
종 료