On-disk B+ Tree MileStone 1

데이터베이스시스템및응용 Prof. 정형수

2017029561 컴퓨터소프트웨어학부 남지훈

1. Possible call path of the insert/delete operation

(1) Call path of insert operation

1) Insert 하고자하는 key 가 이미 존재하는지 판단

이미 존재하는 경우 Insert 하지않고 원래의 root 를 그대로 return 함.

존재하지 않는 경우 key 값을 담은 새로운 record를 만들어 그 record 의 포인터를 Tree 에 삽입하기 위해 변수에 저장함.

2) Insert 하려는 Tree 가 존재하는지 판단 (root 가 NULL 인 경우)

Tree 가 존재하지 않는 경우 저장한 key 와 pointer(record *) 을 가지는 노드를 root 로 하는 새로운 tree를 만들어 return 함.

Tree 가 존재하는 경우 새로 만들 node를 tree에 삽입하기 위한 다음과정을 진행함.

3) key값을 담을 공간이 이미 존재하는지 판단

존재하는 경우 그 공간에 record pointer을 담고 tree 를 return

존재 하지 않는 경우 insert_into_node_after_splitting

(2) Call path of delete operation

Key_record 에 key에 해당하는 record 의 pointer 을 담고, key_leaf 에는 key에 해당하는 node 의 pointer을 담는다.

1) Key를 가진 record 나 node가 있는지 판단

없는 경우 다른 변화 없이 tree를 return 한다.

있는 경우 delete_entry 함수를 호출한다.

Delete_entry 함수에서 Key_leaf 에 담긴 key_record를 제거한다.

2) Key_leaf 가 root 인지 판단.

Root 인 경우 adjust_root 함수 호출 key_leaf 가 empty root 라면 delete 할게 없으므로, 그냥 return.

Empty root가 아니라면 first child 를 새로운 root로 만듦. 자식이 없다면 NULL return

Root 가 아닌 경우 계속 진행

3) Key_leaf 를 삭제한 이후 node를 보존하기 위해 node의 최소크기를 결정

Leaf node 인 경우 leaf page 의 크기에 맞춰 cut(order – 1)

Internal node 인 경우 interal page 의 크기에 맞춰 cut(order)-1

4) Delete 한 이후 tree를 유지할 조건을 만족하는지 판단

만족할 경우 Tree를 return 하고 마무리.

만족하지 않을경우 Tree 구조의 변화를 시도

5) Get_neighbor_index(n)을 통해 neighbor index를 얻고 neighbor 과 합쳐질 수 있는지 판단. (두 node를 합쳤을 때 key의 개수가 capacity를 넘어가지 않는지 판단)

합칠 수 있는 경우 coalescen_nodes 함수 호출

합칠 수 없는 경우 redistribute_nodes 함수 호출

2. Detail flow of the structure modification (split, merge)

(1) Split

Split 은 Tree 에 새로운 node 를 삽입할 때 insert_into_node_after_splitting 함수가 실행되면서 이루어 진다.

1) Split 을 하기 위해 우선 새로운 key와 pointer 을 받은 후, key 값과 pointer 값을(새로운 값들을 포함하여) 담는 이루어진 임시 배열을 만든다.

   - Temp_pointer[j] = old_node->pointers[i]; | temp_keys[j] = old_node->keys[i];

2) 이후, 새로운 노드를 만들고 배열의 키와 포인터를 절반으로 나누어 절반은 기존의 노드에 복사하고, 나머지 절반은 새로운 노드에 복사한다.

   - Cut(order) 을 통해 어디서 나눌지 결정 하여 split 에 담고, 0 ~ split – 2 까지는 기존의 old_node 에, split -1 ~ order -1 까지는 new_node 에 나누어 담는다. 새 노드의 parent 도 기존 노드의 parent 와 동일하게 설정하고, Split-1 에 담긴 key 를 대표키로 정한다.

3) Insert_into_parent 함수를 통해 기존 노드는 왼쪽에, 새로운 노드는 오른쪽에 두어 split 한다.

   - 기존 노드(left) 의 parent 가 존재하지 않다면 insert_into_new_root 를 통해 key 를 root로 하는 새로운 tree에 insert 한다.
   - 만약에 parent 에게 split 한 것을 저장 할 수 있는 left_index가 남아있다면 그곳에 split된 노드를 저장한다.
   - 새로운 node를 넣을 자리가 없다면 parent 에 대해서도 split 을 진행한다.

   새 노드의 key 값에 해당하는 자리가 없다면 split 이 일어난다. Split 이 일어나다가 parent 에게 split 된 node를 담을 수 있는 공간이 있다면, 그곳에 split 된 것을 넣고, 그렇지 않다면 parent 에 대해서도 공간이 생길 때까지 split을 한다. 만약 root 까지 split 해야 한다면, 대표키로 지정한 노드를 root로 하는 새로운 Tree 를 만든다.

(2) Merge – Coalescence

노드의 key의 개수가 매우 적어지면 효율을 위해 merge를 진행한다.

(n->num_keys < min_keys) 를 만족하는 경우.

옆 노드의 키의 개수를 확인하여 옆 노드와 합쳤을 때 capacity를 넘어 서지 않는다면 coalescence, 넘어 선다면 redistribution 해야한다.

1) 합쳐질 노드가 왼쪽 끝에 있다면 그 왼쪽 끝을 없애고 오른쪽으로 옮긴다.

2) Neighbor 에 key 들을 옮기기 위해 neighbor 에 다음 키가 들어갈 수 있는 index 를 negibor_insertion_index에 저장한다.

3) n 이 leaf node 가 아니라면 neighbor 의 마지막 key (keys[neighbor_insertion_index]) 및 pointer 을 대표로 지정한 후 그 뒤에 n 의 key 들을 집어넣는 과정을 진행 그리고 neighbor 에 삽입된 node 들이 neighbor을 부모로 가지도록 설정.

4) Leaf node 라면 neighbor에 key 와 pointer 들을 neighbor에 넣고, 바뀐 neighbor 의 마지막 pointer이 없어진 노드의 오른쪽 노드를 가리키도록 한다.

5) Delete_entry를 실행하고 root를 return 한다.

(3) Merge – distribution

Neighbor 노드로부터 임으로 key, pointer 쌍을 가져와 더 이상 key가 부족하지 않도록 만드는 과정이다.

1) 노드의 왼쪽에 neightbor 이 있는경우 neighbor 의 마지막 key, pointer 쌍을 노드의 왼쪽 끝으로 가져온다.

2) 노드가 가장 왼쪽에 있다면 neighbor의 key,point 쌍을 노드의 오른쪽으로 붙인다.

3) 노드의 num_key를 1 키우고, neighbor은 1 줄인 후, Tree 를 return 한다.

3. Designs or required changes for building on-disk b+ tree

(1) Pages and records

주어진 b+ tree 를 on-disk b+tree로 바꾸기 위해서는 b+ tree 의 data를 담는 노드의 구성을 변화시켜야 한다. 현재 노드는 키와 포인터의 배열을 지니고 있는 특수한 성질이 없는 노드로 이루어진다. 하지만 구현해야 할 on-disk b+ tree 에는 header page, free page, leaf page, internal page 총 4가지가 필요하므로, 이 4가지 page 에 대한 구조체를 각각 구현해야 한다고 생각된다. 4096 Bytes 로 정해진 page size 도

구조체 정의 시 설정하면 된다.

Ex) typedef struct Leaf_page { … } Leaf_page ;

Leaf_page 로 구현되어있을 말단 노드에는 record를 담을 수 있는 형태여야 한다. 이 때 record 는 8 bytes 의 integer key 와 120 bytes 의 string value를 담으므로,int value 만 하나 가지고 있던 record 구조체의 구성 역시 총 128bytes 의 key 와 value 로바꿔 줘야 한다.

(2) Insert

Insert 함수를 정의하는 매개변수의 변화가 필요하다. 기존에는 key 와 value 만을 넣었지만 이제는 page를 구성하는 변수들을 매개변수로 주어줘야 할것이다.

기존에는 key와 value 만 존재하기 때문에 필요 없었지만, on-disk 에서는 leaf_page 들의 split이 일어날때는 copy를, 그 외에 internal node들이 split 될때는 lift_up 이 일어나도록 split 과정을 변경할 필요가 있다.

Leaf page를 internal page에 copy 할 때는 공간의 절약을 위해 leaf page 의 모든 변수가 아닌 internal page 에 존재하는 변수만 copy 하도록 한다.

(3) Delete

Dsik-on b+ tree 에서는 delete 시에 record를 삭제해야 하므로 Delete 함수는 record 나 page 가 사라질 때 그 안에 있는 데이터를 확실하게 제거 하는 함수를 정의해야 할 것이다.

Merge가 너무 자주 일어나면 split 역시 자주 일어날 가능성이 높기 때문에 DB가 느려질 것이다. Lazy merge를 구현하기 위한 노력이 필요할 것 같다.

On-disk 에서의 merge 과정은 page를 삭제하고 page에 새로운 내용을 기입하는 과정이므로 이러한 과정을 수행하는 새 함수를 만들어야 할것같다.