《大数据处理实践》

实验三 默克尔树(Merkle Tree)

实现及应用

（参考指导）

华中科技大学科学与技术学院

大数据课程组

莫益军、石宣化、姚德中、郑龙

2020年10月

目录

[1. 实验目的 1](#_Toc54089858)

[2. 实验评分构成 1](#_Toc54089859)

[3. 默克尔树概述 1](#_Toc54089860)

[3.1 默克尔树概述 1](#_Toc54089861)

[3.2 实验关键代码参考 2](#_Toc54089862)

[4. 实验内容及参考代码 4](#_Toc54089863)

[4.1 实验准备及步骤 4](#_Toc54089864)

[4.2 Merkle树实验参考框架代码 5](#_Toc54089865)

# 实验目的

本实验目的是通过实现Merkle树及其关键特性，了解以下内容：

1. 默克尔树构建与存储。
2. 基于默克尔树检测副本数据不一致性。
3. 默克尔树随着数据增加和删除的动态更新。

# 实验评分构成

本实验主要目的是学习默克尔树编程，并了解其在分布式存储优化和分布式数据一致性的作用，实验评分以满分100分计算。完成默克尔树构建与存储的占40分，基于默克尔树检测副本数据不一致性占20分，默克尔树随节点增加退出的动态更新占40分。

# 默克尔树概述

### 3.1 默克尔树概述

默克尔树是一种哈希树，广泛应用于分布式文件系统和区块链中，以保证分布式数据存储的完整性和一致性。默克尔树对文件和数据进行分块，对数据进行逐块哈希生成叶子节点，根据数据分块顺序进行逐层哈希合并，形成哈希树，默克尔树通常表现为二叉树形式，也可以构建为多叉树。现以1GB的文件，以64MB切片的二叉默克尔树为例说明默克尔树的构建过程。对于1GB文件来说，以64MB大小为数据切片，可以分割为16个数据块，形成如图1所示的数据块。

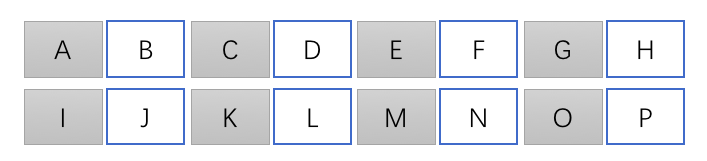


图1 1GB文件文件切片哈希

鉴于MD5散列和SHA-1不具备强抗碰撞性，能被破解，默克尔树采用SHA-2哈希算法。首先对数据块进行哈希运算生成叶子节点，然后逐层进行两两哈希，构建生成如图2所示的哈希树。

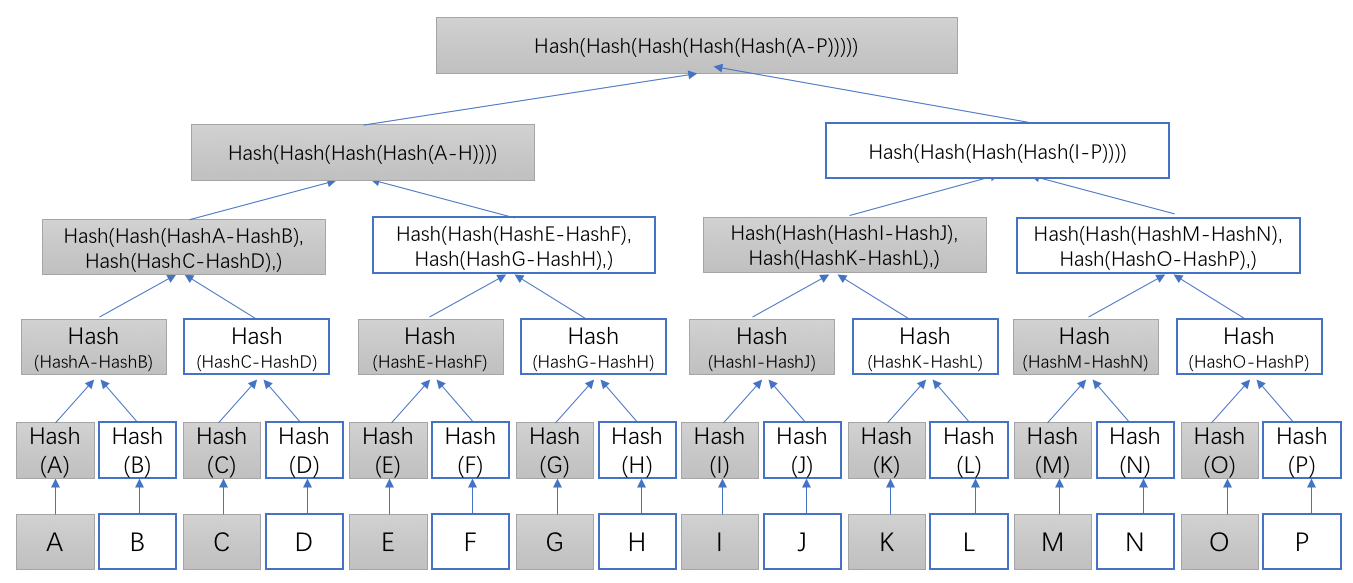


图2 Merkle Tree结构

### 3.2 实验关键代码参考

* 默克尔树参考数据结构

typedef **struct** node {

**int** key; //节点的键，用来加速检索，可选数据块或其名称的代表性特征

**int** hashval; //本节点的哈希值，叶子节点为数据块的哈希，中间节点和根节点为左右子节点拼接后值的哈希。

**struct** node \* parent;

**struct** node \* left;

**struct** node \* right;

} Node;

typedef **struct** mttree {

**int** treesize; //哈希树规模，节点数

Node \*\*arr;

} MtTree;

* 默克尔树构建参考

Node \* build\_tree(MtTree \* mt, Node \* node1, Node \* node2) {

Node \* new = malloc(sizeof(Node));

new->key = concatenate(node1->key, node2->key); //将子节点的键拼接

new-> hashval = hash(concatenate(node1-> hashval,node2-> hashval)); //将子节点的哈希拼接后再哈希。

new->parent = NULL;

new->left = node1;

new->right = node2;

node1->parent = new;

node2->parent = new;

push(mt, new);

return new;

}

* 按键进行默克尔树遍历定位参考

**int** traverse(Node \* root, **int** key){

if(root == NULL){

return -1;

}

else{

if(root->key == key)

return root-> hashval;

else {

if((root)->key < key)

return traverse((root)->right, key);

else

return traverse((root)->left, key);

}

}

}

* 默克尔树遍历打印

**void** showtree(Node \* root) {

**static** **int** depth = 0;

if (root == NULL){

depth = 0;

}

else{

if(root-> left != NULL){

depth++;

showtree(root-> left);

depth--;

for(**int** i=0;i<depth;i++){

printf("==");

}

puts("|");

}

for(**int** k = 0; k < depth-1; k++)

printf("==");

printf("--key: %d hash: %d\n", root->key, root->hashval);

if(root->right!=NULL){

for(**int** i=0;i<depth;i++){

printf("==");

}

puts("|");

depth++;

showtree(root->right);

depth--;

}

}

}

# 实验内容及参考代码

## 4.1实验准备及步骤

**实验3-1:单Merkle树构建**

步骤1：新建一个目录，目录下存放16个文件。

步骤2：对16个文件按文件名字符串大小进行排序。

步骤3：读取16个文件内容，采用SHA256对其进行哈希。

步骤4：按二叉哈希树构建默克尔树，遍历打印默克尔树（包括根节点、中间节点和叶子节点的SHA256哈希值）。

**实验3-2:删除数据块的Merkle树更新**

步骤1：从目录中随机删除一个文件。

步骤2：遍历受删除文件影响的默克尔树分支，并对其进行更新以重建默克尔树。

**实验3-3:增加数据块的Merkle树更新**

步骤1：从目录中增加一个文件。

步骤2：遍历受增加文件影响的默克尔树分支，插入并更新默克尔树。

**实验3-4:删除数据块的Merkle树更新**

步骤1：在前面实验基础上，再增加一个目录，并存放文件（其中有一个文件与前述实验不同）。

步骤2：遍历默克尔树，定位并输出不一致的文件。

注：为简化编程，文件目录下文件可选择小的txt文件。

## 4.2 Merkle树实验参考框架代码

**int** main(**int** argc, **char** \*argv**[]**) {

if(argc == 1 || argc > 3) {

puts("error parameters");

return 0;

}

*//含程序自身参数，总参数数为2时，第2个参数为用于构建默克尔树的目录*

*//含程序自身参数，总参数数为3时，第2个参数为用于构建默克尔树的目录，第3个参数为用于构建默克尔树和比较一致性的目录，*

*//相关逻辑自行补充*

Node \*\* oldnode = NULL;

While(1) *//整体实验大循环，完成实验3-1至3-4*

{

*//初始化创建叶子节点，注意根据目录下文件数分配空间*

*//*initalize\_leaf*函数，输出Node类型的数组，每个叶子节点的key可使用文件名，哈希值为文件所有数据的哈希，叶子节点初始化时，父节点、左右子节点为空，数据结构参考前述数据结构。未初始化时，key值可以设成-1.*

*//hash函数可以是MD5，也可以是SHA-256，或自行设计哈希函数*

Node \*\* node = initalize\_leaf(argv[1]);

if compare(node, oldnode)==0 *//比较路径下是否有文件更新或删除，如果没有更新则不重建默克尔树*

print(‘No change in the directory.’)

continue;

if oldnode == NULL

oldnode = node;

*//构建哈希树*

Node \* root = malloc(sizeof(Node));

MtTree \* mt = malloc(sizeof(MtTree));

//*哈希树叶子节点构建，自行设计迭代实现整体默克尔树构建*

for (i=0; \*node[i]->key!=-1; i+=2)

{

root = build\_tree(mt, \*node[i], \*node[i+1]);

}

Showtree(root);

oldnode = node;

} *//结束大循环*

}