Application Note: Disk Workload Analyzer

6/22/2001

Problem

How does disk cache size affect performance for accessing large (terabyte or greater) disk units? Or at a more detailed level, how can we rapidly count the number of unique disk I/O requests that take place between any two requests for the same disk address?

Solution

Let the "time" of a disk reference be the sequence number of that reference within a disk address trace. Use a JudyL array to look up a given disk address and retrieve the "time" (i.e. sequence number) of the last reference. Then look up this "time" in a Judy1 array and count the number of unique references between then and the last reference.

Example

Let's process a disk trace that contains the following data. The sequence number wouldn't really appear in the data but is implied by the order in which the data is read:

Sequence number	Disk address
1	98321
2	12121
3	27
4	5466
5	12121
6	27
7	100

After all this data is loaded into our two Judy arrays we end up with:

JudyL Index (key)	Value
Disk address	**Sequence #**
27	6
100	7
5466	4
12121	5
98321	1

Judy1 Index (bit set)
1
4
5
6
7

Let's say the next address to process is 5466 again. We look up 5466 in the JudyL array and retrieve its last "time" of 4. Next, using `J1C()` (the `Judy1Count()` macro), we count from 4 to the end of the Judy1 array. Since `J1C()` calls are inclusive, we get the answer 4, which is the number of unique disk references between now and the last time 5466 was referenced.

In a real application where this technique was used, the Judy arrays were hundreds of millions of elements long and this method sped up the overall application by 100X. This speedup was due to the speed of `J1C()` compared to the original application, which counted each item in the stack of disk references to determine the “depth” of the previous reference.

Example Code

```
        while (Get the next block number accessed)
        {
                int     Return;

//              Insert the Block into Block array

                JLI(PPBlock, BlockArray, Block);

//              Is this the first time this Block accessed?
                if (*PPBlock == 0)
                {
//                      Allocate a struct for it
                        PBlock  = AllocBlockStruct();

//                      Use JudyL Value as pointer to the block struct
                        *PPBlock = PBlock;

//                      And stuff with goodies
                        PBlock->RefCnt          = 0;
                        PBlock->DistanceSum = 0;
                }
                else //      Second plus access to this block
                {
                        Word_t Distance;

//                      Count the number of references to this block
                        PBlock->RefCnt += 1;

//                      Count number of stack entries from old to end of stack
                        J1C(Distance, StackArray, PBlock->SeqNum, ~0);

//                      Sum the distances for an overall average
                        PBlock->DistanceSum += Distance;

//                      Delete the old reference from the stack
                        J1U(Return, StackArray, PBlock->SeqNum);

                        assert(Return == 1);
                }
//              Save new reference number for next appearance
                PBlock->SeqNum      = TotalBlocks;

//              And put new sequence number on the Stack
                J1S(Return, StackArray, TotalBlocks);

//              Count unique blocks
                TotalBlocks++;
        }
```