Enhance Hadoop's Local Disk Selection Algorithm under Virtualization Environment

Xiaoding Bian(xiaodingbian@vmware.com)

Author2(author2@vmware.com)

Author3(author3@vmware.com)

July 7, 2013

1 ABSTRACT

Class LocalDirAllocator in Hadoop implements a round-robin algorithm for disk allocation for creating files, which is used by mapred and dfs-client, etc. The purpose of this design is to balance the I/O load of local disks on each node. But in virtualization environment, the I/O occurs on VM disk is actually memory address mapping to physical host. If a physical host contains a few independent VMs, each VM's local file selection is average, the actual I/O load on physical host might be not in balance. A proposal to solve this problem it to let Hadoop know the mapping topology from virutal disks to physical disks, then for each VM, calculate the prefence to access each virtual disk to make sure the actual I/O access occurs on physical is balanced.

2 DESCRIPTION

Hadoop has become very widespread across industries, especially in web companies. One important characteristic of Hadoop is the MapReduce framework, which splits the computation accross many(thousands) TaskTrackers to execute in parallel. For TaskTracker, Hadoop implements a round-robin scheme for disk allocation for creating files, this algorithm allocates the candidate disks one by one for every incoming request. The purpose of this design is to balance the I/O load of local disks on each TaskTracker node. But if Hadoop cluster is deployed on virtualization environment, every TaskTracker node is a virtual machine(VM), the disk I/O occurs on VM's disks is actually mapped to physical host's disks rather than interact with hardware directly.

Consider figure 1, VM_1 and VM_2 are hosted on a same physical host, VM_1's disks are created from physical host's physical disk 1 and physical disk 2, VM_2's are from physical disk 2 and physical disk 3. The I/O throughput on each virtual disk of a VM is ultimately redirected to the corresponding physical disk. If the I/O load across virtual disks of each VM is in balance, and all VMs are fairly scheduled. Then the physical I/O load on physical disk 2 will be heavier than that on physical disk 1 and physical disk 3.

To overcome this limitation, in this article, we propose a solution: let JobTracker be aware of and maintain each TaskTracker's disk topology, this algorithm can be summized as 3 steps:

2.1 Step 1: Let Hadoop cluster be aware of disk mapping topology

- Each slave VMs collect its disk topology which maps its virtual disks to physical disks. Take table 1 as an example, VM_1 and VM_2 are hosted on physical Host 1, VM_3 and VM_4 are hosted on physical host 2. The virutal disks to physical disks mapping relationship is illustrated.

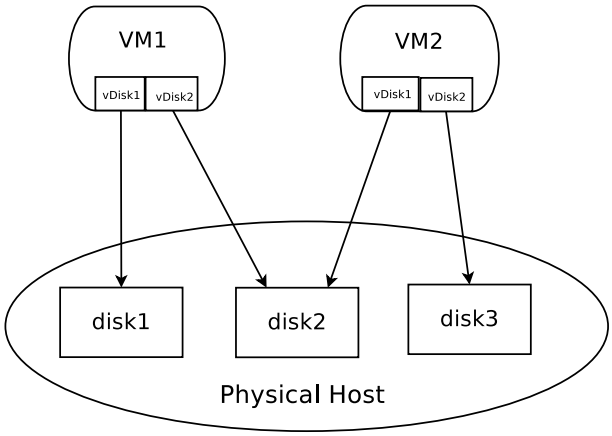

Figure 1: A map example

Table 1: Virtual Disks to Physical Disks mapping topology

pHost1	VM_1	VM_2
	$vDisk_{11} \mapsto pHost1 : pDisk1$	$vDisk_{21} \mapsto pHost1 : pDisk2$
	$vDisk_{12} \mapsto pHost1 : pDisk2$	$vDisk_{22} \mapsto pHost1 : pDisk3$
pHost2	VM_3	VM_4
	$vDisk_{31} \mapsto pHost2 : pDisk1$	$vDisk_{41} \mapsto pHost2 : pDisk1$
	$vDisk_{32} \mapsto pHost2 : pDisk2$	
	$vDisk_{33} \mapsto pHost2 : pDisk3$	

- When TaskTracker service launched, report this information to JobTracker. After JobTracker merged all topology info, classify overall topology based on each physical host, and send back related sub-topology to each VM when next heatbeat.

  Take table 1 as example again, when this step finished, VM_1 and VM_2 will be aware of the sub-topology as table 2a, VM_3 and VM_4 as table 2b

Mapping Info sent to VM_1 & VM_2
$vDisk_{11} \mapsto pHost1 : pDisk1$
$vDisk_{12} \mapsto pHost1 : pDisk2$
$vDisk_{21} \mapsto pHost1 : pDisk2$
$vDisk_{22} \mapsto pHost1 : pDisk3$

(a) sub-topology related to pHost1

Mapping Info sent to VM_3 & VM_4
$vDisk_{31} \mapsto pHost2 : pDisk1$
$vDisk_{32} \mapsto pHost2 : pDisk2$
$vDisk_{33} \mapsto pHost2 : pDisk3$
$vDisk_{41} \mapsto pHost2 : pDisk1$

(b) sub-topology related to pHost2

2.2 Calculate the Disk Allocation Probability Distribution for each VM

After step 1 finished, each VM is aware of the disk mapping topology related to the physical host it hosted on.

Take table 2 as example, in this case, $[VM_1, VM_2, ..., VM_n]$ are hosted on a physical host which contains m physical disks($pDisk_1, pDisk_2, ..., pDisk_m$), a_{ij} is the VM_i' allocation probability for its virtual disk which mapped $pDisk_j$.

Table 2: disk mapping topology

	$pDisk_1$	$pDisk_2$	$pDisk_3$	...	$pDisk_m$
VM_1	a_{11}	a_{12}			
VM_2		a_{22}	a_{23}		a_{2m}
VM_3	a_{31}	a_{32}			
...					
VM_n	a_{n1}		a_{n3}		a_{nm}

Our target is to balance I/O load on all physical disks, this can be descriped as an optimization issue called convex cost with linear constraints:

$$\begin{aligned} min \ & \sum_{j=1}^{m}(\sum_{i=1}^{n} a_{ij} - \frac{n}{m})^2 \\ subject\ to \ & \sum_{j=1}^{m} a_{ij} = 1, \ \forall i \\ & a_{ij} = 0, \ \forall (i,j) \notin \Omega \\ & 0 \le a_{ij} \le 1, \ \forall (i,j) \in \Omega \end{aligned} \tag{1}$$

Here, Ω is defined as the union of VMs and physical disks pairs.

Let M_i be the number of physical disks VM_i occupies. In this article we provide a iterative algorithm 1 to calculate the approximations of the root of this equation 1.

Algorithm 1: calculate disk allocation probability distribution

Data: disk mapping topology of a given VM
Result: the allocation probability distribution of each virtual disk
initialization: let $K = 100, \varepsilon = 10^{-3}$;
foreach $i \in (1,n)$ **do** $a_{ij_1} = a_{ij_2} = ... = a_{ij_{M_i}} = \frac{1}{M_i}$;
for $t \leftarrow 0$ **to** K **do**
  $done = 1$;
  foreach $j \in [1,m]$ **do** $S_j = \sum_{i=1}^{n} a_{ij}$;
  foreach $j \in [1,m]$ **do** **if** $|S_j - \frac{n}{m}| > \varepsilon$ **then**
    $done = 0$;
    break;
  else
    continue;
  end
  if $done == 1$ **then**
    return matrix **a**;
  else
    adjust probability values to approximate optimal values;
    foreach $a_{ij}, i \in (1,n), j \in (1,m)$ **do** $a_{ij} = a_{ij} * \frac{n}{mS_j}$;
    foreach $i, i \in (1,n)$ **do** $T_i = \sum_{j=1}^{m} a_{ij}$;
    foreach $a_{ij}, i \in (1,n), j \in (1,m)$ **do** $a_{ij} = \frac{a_{ij}}{T_i}$;
  end
end

2.3 Determine disk allocation order based on probability distribution

Given a VM and its steady-state allocation probability for each virtual disk $\mathbf{A} = [a_1, a_2, ..., a_n]$, where N is $\mathbf{A}$'s virutual disks number. For each incoming creating file request, to determine which virtual disk it should assign, we tend to customize a $n \times n$ transition probability matrix $\mathbf{P}_A$ to caculate probability distribution.

Assume the probability distribution for last request is

$$\mathbf{A}^k = [a_1^k, a_2^k, ..., a_n^k] \tag{2}$$

then

$$\mathbf{A}^{k+1} = \mathbf{A}^k \mathbf{P}_A \tag{3}$$

For matrix $\mathbf{P}_A$, we need to customize it to get better performance, our target is:

- If $vDisk_k$ is assigned for the last request, current request should **NOT** be assigned $vDisk_k$.
- Beside balance disks in long-term, we should also balance them in short-term. For example, given steady-state disks allocation probability $\mathbf{A} = [0.1, 0.2, 0.3, 0.4]$, even if we only request 10 times, the allocation sequence determined by our algorithm should also approximate to $\mathbf{A}$.

For target 1, only need to set $p_{ii} = 0,\ \forall i \in [1, n]$.

To achieve target 2, let steplength $d_{k,k+1} = |(index_{k+1} - index_k)\ mod\ n|$, where $index_k$ defined as the assgined disk's index inside VM's virutal disks array for the kth request. We tend to minimize the average value of collection $[d_{0,1}, d_{1,2}, ..., d_{k,k+1}]$. By doing this, our algorithm will tend to select virtual disk one by one to guarantee the disk balance even in worst case or for limited times of request, similar to Round-Robin algorithm whose steplength equals 1.

Then, for transition probability matrix

$$\mathbf{P}_A = \begin{pmatrix} 0 & p_{1,2} & p_{1,3} & \cdots & p_{1,n} \\ p_{2,1} & 0 & p_{2,3} & \cdots & p_{2,n} \\ p_{3,1} & p_{3,2} & 0 & \cdots & p_{3,n} \\ \cdots & \cdots & \cdots & \cdots & \\ p_{n,1} & p_{n,2} & p_{n,3} & \cdots & 0 \end{pmatrix} \tag{4}$$

If virtual disk i is selected last time, this time virtual disk j will be selected by probability p_{ij}.

To minimize average steplength,

$$\begin{aligned} & p_{ij} = 0,\ if |(j-i)\ mod\ n| > \alpha_i, \forall i \in [1, n] \\ s.t. & \sum_j p_{i,j} = 1,\ \forall i \\ & \mathbf{AP} = \mathbf{A} \end{aligned} \tag{5}$$

Based on this equation,

$$a_j = \sum_{k=1}^{n} a_k * p_{k,j},\ \forall j \tag{6}$$

To optimize the following equation:

$$\begin{aligned} & min\ \beta_j, \forall j \\ & s.t\ a_j = a_{j-1} * p_{j-1,j} + a_{j-2} * p_{j-2,j} + ... + a_{j-\beta} * p_{j-\beta,j} \end{aligned} \tag{7}$$

We need to realign $\mathbf{A}$ to make sure the adjacent elements' difference value not too much. Asending sort $\mathbf{A}$ as $\bar{\mathbf{A}} = [\bar{a_1}, \bar{a_2}, ..., \bar{a_n}]$. if n is odd, let $\mathbf{B} = [\bar{a_1}, \bar{a_3}, \bar{a_5}, ..., a_{\bar{N}-1}, \bar{a_N}, a_{\bar{N}-2}, a_{\bar{N}-4}, ..., \bar{a_4}, \bar{a_2}]$, else, $\mathbf{B} = [\bar{a_1}, \bar{a_3}, \bar{a_5}, ..., a_{\bar{N}-2}, \bar{a_N}, a_{\bar{N}-1}, a_{\bar{N}-3}, ..., \bar{a_4}, \bar{a_2}]$

Let $\mathbf{P}_B$ be the transition probability matrix for $\mathbf{B}$, Algorithm 3 shows how to determine β_j for each column,

Algorithm 2: Determine no-zero elements number for each column of $\mathbf{P}_B$

```
Data: sorted steady-state probability distribution B
Result: no-zero elements number β_j for each column of P_B
foreach j ← 1 to n do sum = 0;
for t ← 1 to n do
    sum+ = b_{i−t};
    if sum ≥ b_j then
        β_j = t;
        break;
    end
end
```

Next, we can calculate optimal $\widetilde{\mathbf{P}}$ under constraint:

$$\begin{aligned} & min||\mathbf{b}\widetilde{P}-\mathbf{b}||_2 \\ & s.t.\sum_{j=1}^{m} p_{ij}=1,\forall i \\ & p_{ij}=0,\forall(i,j)\notin\Omega_{\widetilde{P}} \\ & 0\le p_{ij}\le 1,\forall(i,j)\in\Omega_{\widetilde{P}} \end{aligned} \tag{8}$$

Here we provide a interative algorithm 3:

Algorithm 3: Determine no-zero elements number for each column of $\mathbf{P}_B$

```
Data: B and β_j, ∀j
Result: Transition probability matrix P_B
initialization: let K = 1000, ε = 10^{−3};
for t ← 1 to K do
    for i ← 1 to n do
        count number of no-zero elements number M based on all β_j;
        foreach j do p_ij = 1/M;
    end
    calculate B̂ = BP_B;
    if Σ_i (b̂_i − b_i)^2 < ε then
        return B;
    end
    foreach i do c_i = b̂_i / b_i;
    foreach i, j do p_ij = p_ij / c_i
end
```

In some worst cases, algorithm 3 may not be able to convergent to the ε, to solve it, a greedy stragety is designed as algorithm 4

Finally, let $e_i=(0,0,...,1,0,...,0)^T$, where the i element equals 1, all others equal 0, then the disk

allocation for each creating file request can be described as algorithm 5

Algorithm 4: Greedy algorithm to improve algorithm 3

```
while True do
    run algorithm 3;
    if cannot achieve ∑_i (b̂_i − b_i)^2 < ε then
        save P̃_B;
        find k_min = {k|β_j[k'] > β_j[k_min], ∀k'};
        if β_j[k_min] == length of B then
            break
        else
            β_j[k_min] += 1;
            re-initialize P_B based on new β_j;
            P_B = 0.5 * (P_B + P̃_B)
        end
    end
end
```

Algorithm 5: Algorithm 1

Data: steady-state probability distrubution **B**
initialization: $\alpha = random(n)$, $\pi_1 = e_\alpha$;
calculate transition probability matrix **P** based; **foreach** *kth request from client,* $k > 1$ **do**
$\pi_k = \pi_{k-1}\mathbf{P}$;
select disk based on the prabability π_k;

Based on this algorithm, we can guarantee the actual disk selection sequence approximate to the steady-state probability distribution, even in worst case or for limited times of request. If all the probability values of the steady-state distribution are the same, this algorithm equivalent to Round-Robin algorithm.

3 EXPERIMENT

To demonstrate the feasilibity of this algorithm, a simulation program is implemented as **git@github.com:xiaodingbian/disk-allocation.git**. In this repository, log file "result_20_40.log" is a running result with 20 VMs and 40 physical disks. And cos similarity function is used to evaluate whether generated sequence for a few requests can follow to steady-state probability distribution.