Fuzzy Pool Balance: An algorithm to achieve a two dimensional balance in distribute storage systems

Wenjing Wu[1] Gang Chen[1]

[1] *IHEP, 19B Yuquan Road, Beijing, China 100049*

E-mail: `wuwj@ihep.ac.cn`

Abstract. The limitation of scheduling modules and the gradual addition of disk pools in distributed storage systems often result in imbalances among their disk pools in terms of both disk usage and file count. This can cause various problems to the storage system such as single point of failure, low system throughput and imbalanced resource utilization and system loads. An algorithm named Fuzzy Pool Balance (FPB) is proposed here to solve this problem. The input of FPB is the current file distribution among disk pools and the output is a file migration plan indicating what files are to be migrated to which pools. FPB uses an array to classify the files by their sizes. The file classification array is dynamically calculated with a defined threshold named T_{max} that defines the allowed pool disk usage deviations. File classification is the basis of file migration. FPB also defines the Immigration Pool (***IP***) and Emigration Pool (***EP***) according to the pool disk usage and File Quantity Ratio (***FQR***) that indicates the percentage of each category of files in each disk pool, so files with higher ***FQR*** in an ***EP*** will be migrated to ***IP***(s) with a lower ***FQR*** of this file category. To verify this algorithm, we implemented FPB on an ATLAS Tier2 dCache production system. The results show that FPB can achieve a very good balance in both free space and file counts, and adjusting the threshold value T_{max} and the correction factor to the average ***FQR*** can achieve a tradeoff between free space and file count.

1.Introduction

The Nowadays mass storage systems are mostly designed in a distributed way, comprising none or several head nodes storing metadata and a set of disk pool nodes storing the real data. While the distributed disk pool nodes bring a lot of benefit such as load balance and aggregated IO throughput, at the same time, various reasons such as file counts and size deviation, data migration to tape systems, garbage collection of removed data can lead to an imbalanced data distribution among disk pools. In this paper we mainly address the imbalance caused by file counts and size deviation. Typical reasons causing so include: 1) the limitation of the storage system scheduling module. The scheduling module decides what files are stored in which disk pools. In theory it can take several factors into account in making such a decision, i.e. disk usage, current disk activities, file counts in each disk pool, but in practical systems, these scheduling modules tend to be designed in a simpler way and takes less factors into account, therefore, over time in a production system, the imbalanced distribution will happen. 2)

Disk pool nodes can be added gradually as the production storage system scales up in its capacity, so old disk pool nodes have much less free space than the new ones.

This imbalanced free space distribution usually triggers the scheduling module to direct new IO operations to the new pool nodes which can cause various harms to the storage system such as 1) single point of failure on the newly added pool node. 2) Imbalanced system and IO load among disk pool nodes, which will reduce the throughput of the whole system.

Furthermore, the limitation of scheduling module can also result in an imbalanced file counts distribution among disk pools which can also be harmful such as 1) Pool nodes with a larger number of files take a much longer time to do initialization. 2) It causes file system even operating system level crisis, i.e. it over uses inode [1] quota. 3) In some implementation of the storage system, it consumes a lot of memory to initiate a disk pool with a large number of files. 4) Disk pools with much larger number of file counts have the tendency of serving more IO operations than the other ones, thus it can cause low system resource utilization and uneven distribution of system load [2].

One solution to this problem is to migrate the files among disk pools to achieve a more even file distribution. Some storage system may already have a module that can move files around based on the free space of all disk pools. This will address one aspect of the imbalance, but not to the imbalanced file counts. Therefore we propose an algorithm named Fuzzy Pool Balance (FPB) in this paper, which aims at achieving a two-dimensional balance, namely both free space and file counts among disk pools by migrating as less files as possible.

The rest of this paper is organized in the following way: Section 2 addresses the related work; Section 3 describes the general idea of FPB; section 4 defines the key conceptions in FPB, including file classification and its array calculation; section 5 describes an implementation of FPB and shows its effect on a production system; section 6 concludes.

2. Related Work

Data migration is a popular concept in distribute storage systems and it improves the data service quality. Traditionally, data migration is triggered by the change of the data access model. The goal of traditional data migration is to use the minimum steps to move data from the current distribution to a desired new distribution according to the users' data access model change. In this case, both current and future data distribution are known, and the algorithms focuses on calculating the minimum steps that are required to finish this migration. Paper [3] [4][5] addressed these cases.

However, the case discussed in this paper is different from the traditional cases. Compared to the traditional cases, it has the following features:

1) The data migration is triggered by the imbalanced data distribution concerning both free space and file counts among disk pools instead of users' data access model change.

2) It knows only the current data distribution and the goal is to calculate a migration plan that can migrate the current data distribution to a future distribution to achieve a balance of both free space and

file counts. The future distribution is unknown and fuzzy, and it requires migrating as less files as possible instead of imposing to use absolute minimum steps, so there is no NP hard problem.

Therefore, we propose the new algorithm FPB to solve the case discussed in this paper.

3. FPB Introduction

FPB is a new algorithm that aims at achieving a balanced distribution of both free space and file counts among disk pools. It takes the current file distribution as input, with several restraints imposed to get a migration plan which describes what files moving from which source disk pools to which destination pools.

The file distribution can be described in the following way: each disk pool is associated with a list of files stored in this disk pool. In this list, it includes the file counts of this disk pool and the size of each file.

As the volume and counts of files addressed here are both large, a key restraint to the migration plan is to have as less file migration among disk pools as possible. To achieve the goal of having a two-dimensional balanced disk pools, two other restraints are also imposed in FPB. 1) The free space deviation among all disk pools should not exceed a threshold value $\boldsymbol{T_{max}}$. 2) File counts are adjusted towards balance among all disk pools during the process of free space balancing. And among these two goals, the free space balance has higher priority.

Based on the above goal and restraints, FPB uses a classification array **STAT** to categorize all the files by their sizes, and categories are also the objects of migration. Disk pools are also labeled as either Immigration Pools (***IP***) or Emigration Pools (***EP***), and in FPB, files will be migrated from ***EP*** into ***IP*** by categories. The restrict definition of ***IP*** and ***EP*** will not guarantee the file counts balance of each file category, but this can avoid excessive files moving around among the disk pools, and eventually within all categories, a balance of both file counts and disk usage can be achieved. Also FPB is designed to work on disk pools with a large amount of files (over thousands) with diverse sizes, so it may not work appropriately in the small scale such as a few files in each disk pools.

4. Notions and Definitions in FPB

4.1 Basic Definitions

To better describe the algorithms of FPB, the following notions and variables are defined:

$\boldsymbol{S_{af}}$: the average free space of all disk pools

IP **(Immigration Pool)**: Disk Pool whose free space is higher than $\mathbf{S_{af}}$

EP **(Emigration Pool)**: Disk Pool whose free space is lower than $\mathbf{S_{af}}$

$\boldsymbol{T_{max}}$: a threshold value for free space deviation among disk pools, in other words, of all disk pools, the free space differences between any given two disk pools should not exceed $\boldsymbol{T_{max}}$

FQR **(File Quantity Ratio)**: for a certain file category, the percentage of file counts in one disk pool compared to the total file counts in all disk pools. ***FQR*** can be calculated as $fqr_{ij} = \frac{Nf_{ij}}{\sum_{k=1}^{N} Nf_{kj}}$, Where fqr_{ij} is the ***FQR*** of file category j in disk pool i, Nf_{ij} is the file counts of category j in disk pool i, N is the total number of disk pools.

$\boldsymbol{F_{qa}}$: the average ***FQR*** of all disk pools

$\boldsymbol{F_{qi}}$ **(Immigration *FQR*):** a threshold value, File Categories in ***IP*** with ***FQR*** below $\boldsymbol{F_{qi}}$ will immigrate

$\boldsymbol{F_{qe}}$ **(Emigration *FQR*):** a threshold value, File Categories in ***EP*** with ***FRQ*** above $\boldsymbol{F_{qe}}$ will emigrate

Usually,

$\boldsymbol{F_{qi} = F_{qa} + R_{in}}$

$\boldsymbol{F_{qe} = F_{qa} + R_{out}}$

Where $\boldsymbol{R_{in}}$ and $\boldsymbol{R_{out}}$ are correction values, and $\boldsymbol{- F_{qa}} <=(\boldsymbol{R_{in}, R_{out}}) <= \boldsymbol{F_{qa}}$.

4.2 File Classification

In order to achieve the file accounts balance among disk pools, FPB puts files into different categories by their sizes. FPB uses the array **STAT** to classify files and the elements of **STAT** are different file sizes in ascending order. The initial values of **STAT** are defined such as [10K 2M 20M 100M 800M 1G 3G], and a file category is marked by the tow neighboring elements except for the edge elements, i.e., 10K marks the category of files whose sizes are less than 10KB;[2M 20M] marks the category of files whose sizes are between 2M and 20M;3G marks the category of files who sizes are greater than 3GB.

In FPB, the interval that marks the category is used to estimate the total file size of this category in a disk pool, so the granularity of **STAT** will affect the accuracy of the estimation, therefore affect the free space deviation between disk pools. In FPB we define a threshold value $\boldsymbol{T_{max}}$ for the free space deviation, so **STAT** can be calculated mathematically according to the value of $\boldsymbol{T_{max.}}$ to ensure the free space deviation not exceeds $\boldsymbol{T_{max..}}$

The following formulas show how **STAT** is calculated and expanded into finer grain.

In **STAT**, for a given interval *i* [St_{i-1}, St_i], we estimate the size of a file in this category as $S_{est} = \frac{St_i + St_{i-1}}{2}$, so the deviation between S_{est} and the real file size Sf_j is $St_{i-1} - \frac{St_i + St_{i-1}}{2} \le Sf_j - S_{est} \le St_i - \frac{St_i + St_{i-1}}{2}$ this is equal to $|Sf_j - S_{est}| \le \frac{St_i - St_{i-1}}{2}$. The total estimated size of all files is $S_{et} = S_{est} \times Nf$, where *Nf is the total file counts in* [St_{i-1}, St_i]. And the real total size of all files is $S_{rt} = \sum_{j=1}^{Nf} Sf_j$.

As we define the deviation between S_{et} and S_{rt} as S_{var} , we get

$$S_{var} = |S_{et} - S_{rt}| = \sum_{j=1}^{Nf} |Sf_j - S_{est}| \le \frac{(St_i - St_{i-1}) \times Nf}{2} \le \beta \tag{1}$$

In the interval *i* [St_{i-1}, St_i], we also define a threshold value β which limits the deviation between the real and estimated total file size in this interval. Given the current number of interval in **STAT** is M, we get

(2)

Combine (1) and (2), whence

(3)

Furthermore, we assume we need to insert *n* intervals into array **STAT**, so *n-1* elements need to be inserted into **STAT**. And we assume in sub intervals 1, 2…n, the respective file counts are n_1, n_2… n_n and the respective deviations are S_{var1} , S_{var2} … S_{varn} , so based on (3), we get

$$S_{var1} = \frac{(St_i - St_{i-1})´\ n_1}{2n} \pounds\ b \qquad (4.1)$$

$$S_{var2} = \frac{(St_i - St_{i-1})´\ n_2}{2n} \pounds\ b \qquad (4.2)$$

…

$$S_{varn} = \frac{(St_i - St_{i-1})´\ n_n}{2n} \pounds\ b \qquad (4.n)$$

Sum up (4.1) ~ (4.n), we get

$$\sum_{j=1}^{n} S_{var\,j} = \frac{(St_i - St_{i-1}) \times \sum_{j=1}^{n} nj}{2n} \leq \beta \times n \text{ ,where } b = \frac{T_{max}}{M+n} \qquad (5)$$

Combine (3) and (5) , we get

and

Whence

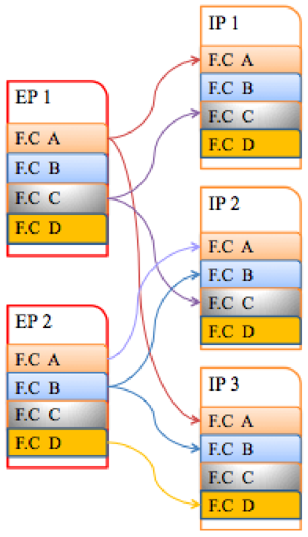
$$n \geq \frac{S_{var}}{2 \times T_{max}} + \sqrt[2]{\frac{S_{var}{}^2}{4T_{max}{}^2} + \frac{S_{var} \times M}{T_{max}}} \qquad (6)$$

And the step for each sub interval is

$$S_{step} = \frac{St_i - St_{i-1}}{n} \qquad (7)$$

4.3 File Migration

.



```
For p1 in EPs
  For c in Categories_ranked_in_descending_order
    If p1.c.FQR > $F_{qe}$ and p1.em_space > 0
    em_FQR = p1.c.FQR- $F_{qe}$
    For p2 in IPs
      If p2.c.FQR < $F_{qi}$ and p2.im_space > 0
        im_FQR = p2.c.FQR- $F_{qi}$
        If im_FQR >= em_FRQ
          move_no = c.total_fileno* em_FQR
          update_FQR(p2.c)
          mresults.update(p1,p2,c,move_no)
          break
        else
          move_no = c.total_fileno*im_FRQ
          em_FQR -= im_FQR
          update_FQR(p2.c)
          mresults.update(p1,p2,c,move_no)
    else
      continue
```

Figure 1: File Migration Flow Figure 2: FPB Pseudo Code

As show in figure 1, disk pools are categorized into ***EP*** and ***IP*** according to their free space, and files will only be migrated from ***EP***s into ***IP***s. In an ***EP***, categories whose ***FQR*** are higher than $\boldsymbol{F_{qe}}$ will emigrate a certain amount of its files into ***IP***s. In the same sense, in an ***IP***, categories whose ***FQR*** are lower than F_{qi} will immigrate files of the same categories from the ***EP***s within its immigration capacity quota. For example, in figure 1, in EP1, the ***FQR*** of category A and C exceeds the threshold value, so respectively a certain amount of files of category A will be emigrated into IP1 and IP3, and a certain amount of files of category C will be emigrated into IP1 and IP2.

To store the information of the migration plan, we use a four-element data structure mresult {ep, ip, file_category, move_no}.

Figure 2 shows the pseudo code describing the algorithm of file migration

5. An Implementation and Test Results

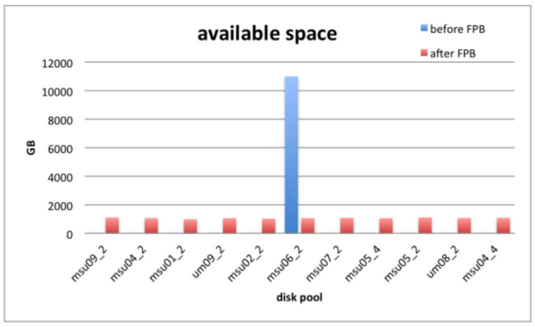
5.1 FPBM

Based on the FPB algorithm, we implemented the Fuzzy Pool Balance Manager (FPBM) on the Mass storage system dCache [6]. FPBM includes 4 basic components: 1) CM (Classification Manager) which generates the **STAT** array based on the file distribution among all disk pools and categorize files based on their sizes; 2) BM (Balance Manager) which generates a migration plan based on the file classification, pool categories and all threshold values. The migration plan describes what files to be moved from its source pools to the destination pools; 3) FM (File Mover) which executes the migration of the files among disk pools, and this component relies on the specific commands of the storage system; 4) MM (Migration Monitoring) which monitors the rates of file migration and provides basic statistics of the migration. Among all these 4 components, only FM is specific to storage systems, because it relies on the system specific commands to move the files. In dCache, the admin interface [7] can be used to migrate the files among disk pools.

5.1 Test Resutls

FPBM is used on a production dCache system of an ATLAS Tier2 site AGLT2 [8]. The system includes 11 disk pools of 120TB storage space. Both the free space and file counts among the disk pools are imbalanced as a new disk pool is added to the system. Before the migration, 10 of the 11 disk pools has less than 10GB free space respectively and the new disk pool has 11TB free space. The threshold values are set as: T_{max}=1000GB, $F_{qe} = F_{qa}$ -0.15, $F_{qi} = F_{qa}$ +0.15.

As shown in Figure 5, after FPBM, the free space is evenly distributed among the disk pools with each disk pool having 1.0~1.1TB free space, and the deviation of this is way below the threshold value T_{max} (1000GB). As shown in Figure 6, the ***FQR*** is also adjusted in the disk pool. Take a particular pool for example, for the category [38M 44M], the ***FQR*** is drawn closer to F_{qa} by the migration process which means the file counts are balanced among the disk pools at the same time. However in some disk pools, the ***FQR*** does not change, that is because of the definition of ***EP*** and ***IP***. For example, in an ***EP***, even if the ***FQR*** of a certain category is lower than F_{qi}, it still won't immigrate any files, so the ***FQR*** of this category remains the same. The same applies to ***IP***s with categories whose ***FQR*** are higher than F_{qe}. However, in the result of this empirical example, we only measured the changes of FQR of each file category which could indirectly indicate the change of file counts distribution, but not measure directly the change of file counts of each disk pool. This needs to be improved in the future.



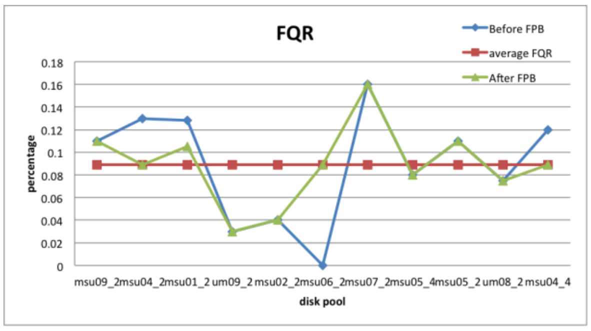


Figure 5: Balance of Free space

Figure 6: Balance of File Counts

6. Conclusions

FPB introduces the idea of size based file classification and calculates the classification array mathematically according to the threshold value. FPB also uses the notion of ***FQR*** to adjust the file counts while migrating the capacity to achieve a two dimensional balance in terms of free space and file counts among disk pools. The results of applying it on a production storage system prove its effect and also indicate that adjusting the threshold value $\boldsymbol{T_{max}}$ and the correction factor to the $\boldsymbol{F_{qa}}$ can achieve a tradeoff between free space and file counts among disk pools: smaller $\boldsymbol{T_{max}}$ results in a finer grain of classification array **STAT** which leads to better free space but worse file counts distribution among disk pools; $\boldsymbol{F_{qi}}$ and $\boldsymbol{F_{qe}}$ being closer to $\boldsymbol{F_{qa}}$ leads to better file counts but worse free space distribution among disk pools. However, in FPB, the priority is to balance the free space, and at the same time, FPB helps to adjust the file counts in the condition of moving as less files as possible.

References

[1] Understand UNIX / Linux Inodes Basics with Examples, http://www.thegeekstuff.com/2012/01/linux-inodes/, 2012

[2] Wenjing Wu, Research on Performance of Grid Storage System over 10Gb Networks, doctoral dissertation, 2010. p. 64-p83

[3] Golubchik, L., et al., Data migration on parallel disks: Algorithms and evaluation. Algorithmica, 2006. 45(1): p. 137-158.

[4] Hall, J., et al. On algorithms for efficient data migration. 2001: Society for Industrial and Applied Mathematics Philadelphia, PA, USA.

[5] Anderson, E., et al., An experimental study of data migration algorithms. LECTURE NOTES IN COMPUTER SCIENCE, 2001. 2141: p. 145-158.

[6] Ernst M, Fuhrmann P, Gasthuber M, et al. dCache, a distributed storage data caching system[C]//Proceedings of Computing in High Energy Physics. 2001, 1: 2.

[7] dCache admin interface, http://www.dcache.org/manuals/Book-1.9.5/start/intouch-admin.shtml, 2013

[8] AGLT2 dCache Monitoring, http://head01.aglt2.org:2288/