# Enhancing Throughput of Hadoop Distributed File System for Interaction-Intensive Tasks

Xiayu Hua, Hao Wu and Shangping Ren
Department of Computer Science
Illinois Institute of Technology
Chicago, Illinois 60616
Email: {xhua, hwu28, ren}@hawk.iit.edu

***Abstract*—The performance of the Hadoop Distributed File System (HDFS)decreases dramatically when handling *interaction-intensive* files, i.e., files that have relatively small size but are accessed frequently. The paper analyzes the cause of throughput degradation issue when accessing interaction-intensive files and presents an enhanced HDFS architecture along with an associated storage allocation algorithm that overcomes the performance degradation problem. Experiments have shown that with the proposed architecture together with the associated storage allocation algorithm, the HDFS throughput for interaction-intensive files increase 300% in average with only a negligible performance decrease for large data set tasks.**

***Index Terms*—HDFS, Cache, Hierarchical structure, PSO, Storage Allocation Algorithm**

## I. Introduction

Hadoop Distributed File System (HDFS) is designed as a massive data storage framework and serves as the storage component for the Apache Hadoop platform. The HDFS provides a highly reliable and globally accessible storage with high throughput for large data sets [1].

However, the advantage of high throughput that the HDFS provides diminishes quickly when handling *interaction-intensive* files, the files that are of small sizes but are accessed frequently. The reason is that before an I/O transmission starts, there are some necessary initialization steps that need to be completed, such as data location retrieving and storage space allocation. When transferring large data, this initialization overhead becomes relatively small and can be negligible comparing with the data transmission itself. However, when transferring small size data, it becomes significant. In addition to the initialization overhead, files with high I/O data access frequencies can also quickly overburden the regulating component in the HDFS, i.e., the single namenode that supervises and manages every access to datanodes [2]. If the number of datanodes is large, the single namenode can quickly become a bottleneck when there is high frequency of I/O requests.

To overcome the issue for interaction-intensive tasks, efforts are often made from three directions: **(a)** improve metadata structure or use cache to provide faster I/O with less overhead [3], [4], **(b)** extend the namenode with a hierarchical structure [5], [6] to avoid single namenode overload, and **(c)** design a better storage allocation algorithm to improve data accessibility [7], [8].

In this paper, we present an integrated approach to addressing the HDFS performance degradation issue for interaction-intensive tasks. In particular, we extend the HDFS architecture by adding cache support and transforming the single namenode to an extended hierarchical namenode architecture. Based on the extended architecture, we develop a Particle Swarm Optimization (PSO) based storage allocation algorithm to improve the HDFS throughput for interaction-intensive tasks.

## II. Extended HDFS Namenode Structure

To overcome the bottleneck issue existed in the original HDFS architecture for interaction-intensive tasks, we extend the single namenode structure to a two-layer namenode structure. Fig. 1 depicts the new architecture.

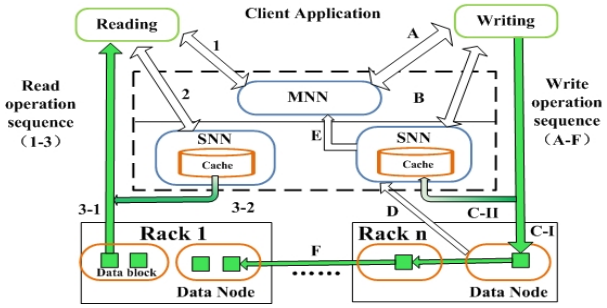



Fig. 1. Hierarchical Namenode Structure

As shown in Fig. 1, the first layer contains the Master Name Node (MNN) which is equivalent to the namenode in the original HDFS structure. The second layer consists of a set of Secondary Name Nodes (SNNs), which are applied to every rack of datanodes. Caches are deployed to every SNN. The functionality of the SNN is to relay communications between namenode and datanode and store interaction-intensive file blocks in its cache to allow fast access. To the MNN, each SNN is its datanode; to each datanode, the SNN in its rack is the namenode. Hence, the new structure fully preserves the original HDFS relations between MNN and SNN, and SNN and datanode structures. Therefore all the original functions and mechanisms of the HDFS are intact and the modifications needed for the structural extension is minimal.

## III. Storage Allocation Algorithm

With this new structure, the throughput degradation caused by the interaction-intensive tasks can be further reduced by

applying an optimized storage allocation strategie. The development of this strategies is done in two steps: (1) determine file block's interaction-intensity value (*I*) and (2) based on its *I* value, decide the file block's storage location.

### A. Interaction-Intensity

The interaction-intensity value (*I*) of a file is a measurement of how frequent this file is accessed (read or write) by one or multiple applications. All the blocks of a file shares the same *I* value of this file. Since the *I* value of a file varies from time to time, it is represented as a function of time.

More precisely, for a given file (f), its interaction-intensity value $I$ at time instance $t$ is defined as: $I(f, Q, t) = |R|$ where $Q$ is a given length of a time quantum during which the *I* value is calculated. It is a constant. The set $R$ denotes the number of requests to file $f$ with given time interval $Q$. Formally, $R = \{(f_i, t_i) | f_i = f \wedge max\{0, t - Q\} \leq t_i \leq t\}$ where $(f_i, t_i)$ denotes a access request with file name $f_i$ and request arriving time $t_i$. The intuitive meaning of $I(f, Q, t)$ is that for the file $f$ at the time instant t, the total number of requests of this file submitted to HDFS in the last time quantum.

### B. Brief Introduction To Particle Swarm Optimization

The concept of the Particle Swarm Optimization (PSO) was first introduced in [9]. As an evolutionary algorithm, the PSO algorithm depends on the explorations of a group of independent agents (the particles) during their search for the optimum solution within a given solution domain (the search space). Each particle makes its own decision of the next movement in the search space using both its own experience and the knowledge across the whole swarm. Eventually the swarm as a whole is likely to converge to an optimum solution.

To apply the PSO to the storage allocation problem we need to: *(1)* define a particle structure and a search space; *(2)* define the optimize objective functions for both MNN and SNN layers and; *(3)* use PSO to explore the solution domain and eventually derive a near optimal solution vector. This solution vector is the allocation plan that maximizes the overall throughput of an incoming batch of file blocks.

### C. Storage Allocation at the MNN Layer

In the MNN layer, the MNN only allocates blocks to either racks or caches. The solution vector ($\overrightarrow{V_M}$) at the MNN layer is used as an allocation plan to indicate the storage place for each incoming file block.

To specify the structure of $\overrightarrow{V_M}$, we use $F_M$ to denote the set of file block IDs that are available and with *I* value larger than zero. A queue $Q_F$ is defined to contain the IDs of all the files whose blocks are in $F_M$. As the HDFS uses the FIFO policy to schedule their tasks, the queue $Q_F$ is ordered by the block's arrival time. The subset $F_{MI} \subseteq F_M$ contains blocks in $F_M$ with their $I$ values larger than the interaction intensity threshold $I_{low}$. In other words, $F_{MI}$ contains all the incoming interaction-intensive blocks.

If there are $n$ racks at the SNN layer, to the MNN, there are $2n$ datanodes: the first $n$ datanodes represent the datanode pool in each rack and the other $n$ represent the caches in each rack. For example, assume that MNN decides a block be allocated to the $k$th datanote, if $k \leq n$, then this block is assigned to the datanode pool of rack $k$; otherwise, if $k > n$, that means this block is allocated to the cache on the $(k - n)$th rack.

The solution vector at the MNN layer is denoted as $\overrightarrow{V_M}$. Its size is $|F_M|$. $\overrightarrow{V_M}[i]$ represents the location where block $Q_F[i]$ is allocated to. The value domain of entry $i$ in vector $\overrightarrow{V_M}$ is defined as follows:

$$\overrightarrow{V_M}[i] \in \begin{cases} [1, 2n], \text{ if } Q_F[i] \in F_{MI} \\ [1, n], \text{ if } Q_F[i] \in F_M - F_{MI} \end{cases} \tag{1}$$

As given in Eq. (1), for blocks in $F_{MI}$, the value domain of their corresponding entries in $\overrightarrow{V_M}$ is $[1, 2n]$ which indicates that the blocks in $F_{MI}$ may be stored in cache. For the rest of the blocks, as their domain is in $[1, n]$, they can only be allocated to datanodes in a rack.

To compare the interaction-intensity of incoming files with those which are already in the cache, indicator $\Gamma$ is introduced for each incoming file to represent the ratio of the file's current $I$ value versus the average $I$ value of all cached files: $\Gamma_f = \frac{I_f}{(\sum_{i=1}^{N} I_i)/N}$, where $N$ is the total number of file blocks cached in the SNN layer. Given the indicator $\Gamma$, assume the incoming files are allocated to rack $r$, and there are $A[r]$ blocks on rack $r$, the cost function *Cost* for the solution vector $\overrightarrow{V_M}$ in the MNN layer is defined as:

$$\begin{aligned} \textit{Cost} = k_1 \cdot \sum_{r=n}^{2n} \Psi(r - n) \cdot \frac{\sum_{\overrightarrow{V_M}[i]=r} \Gamma_i^{-1}}{A[r]} + \\ k_2 \cdot \sum_{r=1}^{n} \Omega(r) \cdot \frac{\sum_{\overrightarrow{V_M}[i]=r} \Gamma_i}{A[r]} \end{aligned} \tag{2}$$

where the value *Cost* is determined by two factors: the estimated cost of placing blocks into caches ($\Psi$), and the estimated cost of placing blocks into datanodes ($\Omega$). $k_1$ and $k_2$ are the weight factors of these two components.

Allocating blocks with high (low) $\Gamma$ value into a storage place that has low (high) I/O workload, such as an idle cache, reduces the *Cost* value; while putting blocks with low (high) $\Gamma$ value to a low (high) workload storage place, increases the *Cost* value. As smaller cost of I/O tasks can bring larger throughput to the system, Eq. (2) becomes the objective function for the PSO-based algorithm. In Eq. (2), $\Psi(r)$ is defined as follows and we leave the definition of $\Omega(r)$ to the next subsection:

$$\begin{aligned} \Psi(r) = P^{\frac{B * A[r]}{R[r]}} \cdot (k_3 \cdot \frac{W^{act}[r] \cdot \boldsymbol{B} \cdot A[r]}{W^{avg}[r]} \\ + k_4 \cdot \frac{(R^{act}[r] + 1) \cdot \boldsymbol{B} \cdot A[r]}{R^{avg}[r]}) \end{aligned} \tag{3}$$

where array $R$ records the size of remaining cache space available in each rack, arrays $R^{act}$ ($W^{act}$) and $R^{avg}$ ($W^{avg}$) record the number of active reading (writing) channels connected to the cache and average reading (writing) throughput of the cache in each rack, respectively; they are obtained from the HDFS. *B* denotes the block size defined by the system.

Constants $k_3$ and $k_4$ are the weight factors used to measure the ratio of read/write frequencies of the corresponding task.

$P$ is a coefficient used to introduce penalty into the cost function for using caches. As the total space of caches is scarce compared to the storage volume provided by datanodes, the penalty $P^{\frac{B*A[r]}{R[r]}}$ increases exponentially to the ratio of required cache space versus remaining cache space. As a result, when the cache is nearly full, the PSO is more likely to allocate the interaction-intensive blocks to the datanodes with lighter work load rather than to the cache. Furthermore, if the size of blocks assigned to the cache of rack $r$ is larger than its available space, i.e., $B * A[r] > R[r]$, the value of $\Psi(r)$ will be greatly scaled up and this allocation plan is unlikely to be chosen by the PSO.

### D. Storage Allocation at the SNN Layer

For each allocation solution generated by the MNN during the PSO searching procedure, the SNNs calculate their own feedback factor $\Omega$. In fact, the factor $\Omega$ itself is a quality evaluation criterion for allocation plans generated at the SNN layer. In other words, this is the objective function for the PSO applied in this layer.

For the SNN in rack *r*, its solution vector is denoted as $\overrightarrow{V_S^r}$. Use $S_r$ to define the set of blocks assigned from the MNN to rack $r$, then the cardinality of $\overrightarrow{V_S^r}$ is $|S_r|$. Similar as the $\overrightarrow{V_M}$ to the MNN, each entry in $\overrightarrow{V_S^r}$ indicates the storage destination allocated to each block assigned from the MNN to this rack. Use $D_r$ to denote the number of datanodes in rack $r$, the value domain for each entry in $\overrightarrow{V_S^r}$ is $[1, D_r]$, i.e., each block in the set $S_r$ can be placed into any datanode in this rack.

For the SNN in rack $r$, its cost function $\Omega_r$ is defined as: $\Omega(r) = \sum_{i=1}^{D_r} (k_3 * \frac{|Y_i|}{E_R(i)} + k_4 * \frac{|Y_i|}{E_W(i)})$ where $E_W(i)$ and $E_R(i)$ are the predicted writing and reading speeds on node $i$ which are given by the auto-regressive integrated moving average prediction model presented in [10]. $k_3$ and $k_4$ are the weight coefficients for reading and writing time costs. $Y_i$ is used to record the number of blocks allocated to datanode $i$.

### E. Apply PSO to the Storage Allocation Problem

In the case of the storage allocation problem, a particle in the PSO is a candidate allocation plan. The structure of particles in the MNN and SNN layers are $\overrightarrow{V_M}$ and $\overrightarrow{V_S}$, respectively.

The particle moves within the search space from one point to another. The edge of the search space is defined by the value domain of the solution vector. Each point in the search space represents an allocation combination. Since the incoming file blocks have different interaction-intensities and the storage places in the HDFS have different I/O performances, different combinations can provide different I/O throughput for incoming blocks. When one combination is selected (one particle moves to the point in the search space corresponding to this combination), the estimated cost of this combination (the quality of the corresponding point) can be evaluated by the object function. In our scenario, minimum cost indicates maximum throughput. Furthermore, the terminating condition of the PSO applied in this scenario is reaching a given value of the iteration time.

Let array $p[x]$ represents the coordinates of a particle's current location, array $b[x]$ represent the coordinates of the best known position within the history of this particle, and array $g[x]$ denote the coordinates of the best known position within the history of the entire swarm. According to the PSO procedure, the movement of a particle between two iterations is determined by the velocity vector denoted as array $V[x]$ which has the same cardinality as the particle. In this array, $V[i]$ represents the $i$th component velocity, which is determined as: $V[i] = \omega \cdot V[i] + C_1 \cdot r_p \cdot (b[i] - p[i]) + C_2 \cdot r_g * (g[i] - p[i])$ where coefficient $\omega$ is the inertia weight. $C_1$ and $C_2$ are two acceleration parameters which control the weight of learning from the particle's own best position and the weight of learning from the global best position, respectively. $r_p$ and $r_g$ are two random numbers between zero and one. Since the PSO applied to the storage allocation algorithm has limited iteration time, the speed of particle convergence is critical. Hence, the Evolutionary State Estimation (ESE) technique [11] is introduced to improve the convergence speed and quality.

The PSO-based algorithm with ESE is illustrate in Algorithm 1.

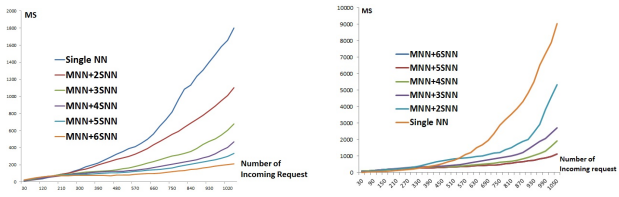
**Algorithm 1:** Algorithm of PSO with ESE

```
Generate an initial population of particles within search spaces;
Each particle evaluate its current location by the Optimal Object Function;
while iteration time limit not met do
    update current global best location g[x];
    configure ω, C1 and C2;
    for each particle do
        for each dimension do
            Determine the velocity component;
        Move to new location by updating array p[x];
        Update particle's best location array b[x];
        Evaluate its current location by the Optimal Object Function;
return g[x];
```

## IV. Experiment Specifications and Result Analysis

In this section, we are to empirically show that modifications made to the original HDFS is able to (1) delay the time when the namenode becomes overloaded; and (2) the system throughput is increased for interaction-intensive tasks. The testbed consists of 130 workstations and servers. The test applications of the experiments is a pure I/O programs combined with the Montage program that is dedicated to processing space photo image blocks in parallel [12].

Fig. 2(a) and Fig. 2(b) show the comparison of the I/O delays caused by processing read and write requests, respectively. The results clearly indicates that the I/O delays increase much sooner and faster for a system with only a single MNN than a

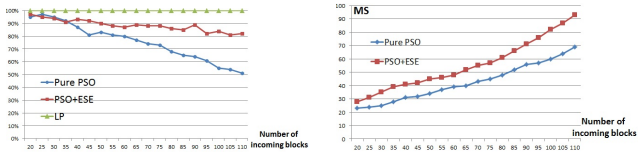



(a) Read Delay (b) Write Delay

Fig. 2. I/O Delays under Different Number of SNNs

system with multiple SNNs; the more SNNs the system has, the later and slower the I/O delays.

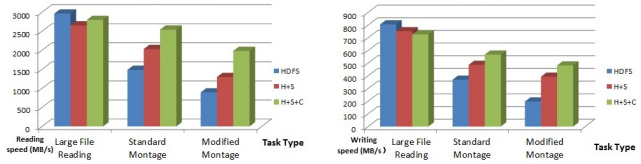
The second experiment compares the solution qualities and calculation times of PSOs with and without ESE, respectively. We calculate the *Cost* values using Linear Programming (LP) approach which gives the optimal solution, and using PSOs with and without ESE approaches. The results are illustrated in Fig. 3(a) and Fig. 3(b), respectively.




(a) Solution Quality (b) Allocation Time

Fig. 3. PSO with and without ESE Comparison

As shown in Fig. 3(a), the solution quality (*Cost*) for the pure PSO algorithm without ESE decreases at a faster rate than it is with ESE. Even with 110 incoming blocks, the PSO with ESE can still provide solutions with quality that is no less than 80% of the optimal solution. Fig. 3(b) illustrates their time costs. From Fig. IV, it is clear that the PSO with ESE significantly improves the quality of the solution with only a slight increase on the calculation time.




(a) Read Performance (b) Write Performance

Fig. 4. Performance Comparison for Different Types of Applications

The third experiment evaluates the I/O throughput of handling interaction-intensive tasks using the original HDFS structure, HDFS with SNN (H+S) and HDFS with SNN and cache support(H+S+C). Storage allocation algorithm are deployed on both of extended structures with cache size zero for the H+S structure. We consider three test cases: (1) pure I/O tasks with large data volumes; (2) standard Montage program with a large number of small files and has frequent I/O requests on these files; and (3) a modified Montage program that increases 40% or more its frequency for I/O requests. The results are depicted in Fig. 4(a) and Fig. 4(b).

As shown in the figures, when dealing with pure I/O tasks, the original design of the HDFS has the best performance. This is because the hierarchical structure and the algorithm do introduce some overhead. However, the influence of the additional overhead on the performance is small. When the test case is changed from the pure I/O task to the standard Montage program and then the modified one, the performance of the original HDFS decreases significantly. As a contrast, the performance of the H+S+C structure becomes significantly better than both the original HDFS and H+S structures.

## V. Conclusion

This paper has presented an enhanced HDFS in which the performance of handling interaction-intensive tasks is significantly improved. The modifications to the HDFS are: **(1)** changing the single namenode structure into an extended namenode structure; **(2)** deploying caches on each rack to improve I/O performance of accessing interactive-intensive files; and **(3)** using PSO-based algorithms to find a near optimal storage allocation plan for incoming files.

Structurally, only small changes were made to the HDFS, i.e. extending single namenode to a hierarchical structure of namenodes. However, the experimental results show that such a small modification can significantly improved the HDFS throughput (up to 300% in average)when dealing with interaction-intensive tasks and only cause slight performance degradation for handling large size data accesses.

## References

[1] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. Ieee, 2010, pp. 1–10.

[2] D. Borthakur, "The hadoop distributed file system: Architecture and design," *Hadoop Project Website*, 2007.

[3] S. Chandrasekar, R. Dakshinamurthy, P. Seshakumar, B. Prabavathy, and C. Babu, "A novel indexing scheme for efficient handling of small files in hadoop distributed file system," in *Computer Communication and Informatics (ICCCI), 2013 International Conference on*. IEEE, 2013, pp. 1–8.

[4] X. Liu, J. Han, Y. Zhong, C. Han, and X. He, "Implementing webgis on hadoop: A case study of improving small file i/o performance on hdfs," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*. IEEE, 2009, pp. 1–8.

[5] H. Liao, J. Han, and J. Fang, "Multi-dimensional index on hadoop distributed file system," in *Networking, Architecture and Storage (NAS), 2010 IEEE Fifth International Conference on*. IEEE, 2010, pp. 240–249.

[6] D. Borthakur, "Hdfs architecture guide," *HADOOP APACHE PROJECT http://hadoop. apache. org/common/docs/current/hdfs design. pdf*, 2008.

[7] H. Hsiao, H. Chung, H. Shen, and Y. Chao, "Load rebalancing for distributed file systems in clouds," 2013.

[8] A. Indrayanto and H. Y. Chan, "Application of game theory and fictitious play in data placement," in *Distributed Framework and Applications, 2008. DFmA 2008. First International Conference on*. IEEE, 2008, pp. 79–83.

[9] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol. 4. IEEE, 1995, pp. 1942–1948.

[10] S. Vazhkudai, J. M. Schopf, and I. Foster, "Predicting the performance of wide area data transfers," in *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*. IEEE, 2002, pp. 34–43.

[11] Z.-H. Zhan, J. Zhang, Y. Li, and H.-H. Chung, "Adaptive particle swarm optimization," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 39, no. 6, pp. 1362–1381, 2009.

[12] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: the montage example," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press, 2008, p. 50.