

Allocation of Partitioned Data by Using A Neural Network Based Approach

Manghui Tu
Computer Science and Information Systems
Department
Southern Utah University

Peng Li
Computer Science Department
University of Texas at Dallas

Dongfeng Wang
Wind River Systems
Research Center

Nasser Tadayon
Computer Science and Information Systems
Department
Southern Utah University

Abstract

In this paper, we investigate allocation of data partitions in the network for data objects that are partitioned by using secret sharing schemes [12] or erasure coding schemes [11]. We defined the problem as finding M nodes in the network to host the data partitions so that the total cost of reading K partitions and updating M partitions by all nodes in the network is minimized. This problem is NP-hard. We proposed a SOFM-like algorithm and a heuristic algorithm for this problem. The numerical study shows both algorithms are effective for this problem.

Key Words: Security, data partitioning, secret sharing, erasure coding, SOFM like Algorithm.

1. Introduction

The rapid development and acceptance of the Internet has led us to an electronic information world. Today, as our lives rely so much on the information, the security has been a critical issue for information data storage, transmission and processing [2, 13, 16].

To protect a critical information system, intrusion tolerance should be designed to provide multiple layer protection, besides intrusion detection. Partitioning and scattering techniques are used by many proposed intrusion tolerance strategies [1, 9]. Basically, a critical data in the system is partitioned and then encrypted. To ensure that no data will be revealed if a partition is compromised, no partition should contain a significant amount of information. Encrypted copies of the partitions are replicated and scattered across distributed servers. Redundancies in the data partitions can be used to provide integrity protection.

Many of the above intrusion tolerance algorithms are built upon one basic scheme, the Shamir's secret sharing scheme [12], which is also called (K, M) threshold scheme. Essentially, a secret data item d is partitioned into M partitions and distributed to M partition-holders, one partition for a partition-holder. A polynomial f of degree K is chosen to compute the M partitions, each partition d_i is $f(x_i)$ for some chosen x_i and the original data d is the constant field of f . Thus, from K or more partitions, the data item d can be reconstructed. Furthermore, no information about d can be inferred from fewer than K partitions.

Another approach is erasure coding scheme [11]. An erasure coding scheme is similar to a secret sharing scheme, except that it is highly space efficient but provides weaker security protection to data [11, 13, 16]. To provide confidentiality protection, data partitions are generally encrypted. The encryption keys are generally secret shared and dispersed with the data partitions to improve access performance [6, 7]. The two partition schemes have similar impact on access performance [13, 16].

In distributed secured information systems adopting the (K, M) threshold scheme, the numerical relationship between M and K varies depending on applications. Usually M is much bigger than K , and the redundancy is to provide convenience. In these distributed information systems, a data partition is read and may be updated from multiple locations by clients in the network. To retrieve the complete and accurate information, a read needs to access at least any K data partitions among the M partitions. An update needs to update all of the M data partitions. To improve the read response time and reduce network communication overhead, partitions are better to be located close to the clients. Since accessing K data partitions is sufficient for a read, then M partitions can be distributed and each K partitions can serve their own cluster of client read accesses. However, to improve the response time and reduce communication overhead for update, M partitions are better to stay close to each other. Thus, the resident location of the M data objects has great impact on the information system reliability, availability and performance.

Many research efforts have been devoted to the optimal placement of distributed files, data objects and other network services [4, 15, 17]. However, all these works only consider the allocation of one single data object or multiple independent data objects, while data partitions have dependency when considering allocation [13]. There are very limited work on the allocation of partitioned data [8, 10], however, access performance is not their major concern.

In this paper, we consider a distributed secured information system deployed over a network with general topology, with N nodes. The distributed information system adopts the (K, M) threshold scheme. We address the placement issues of such M data partitions in the network.

Our goal is to improve the access performance, especially to reduce the communication overhead in the whole network due to the frequent accesses on the partitioned data. We define this problem as a (N, M, K) placement problem. We propose a SOFM-like algorithm and a heuristic algorithm for the secure data placement problem. In our numerical study, we compare the solutions solved by these algorithms with the optimal solutions. The results show that both algorithms are effective.

The remainder of this paper is organized as follows. Section 2 describes our system model and problem definition. Section 3 introduces the SOFM-like algorithm for the (N, M, K) placement algorithm. Section 4 introduces the heuristic algorithm. Section 5 presents the numerical results of our algorithms. Section 6 states the conclusion of the paper.

2. System Model and Problem Specification

The secure data object in the distributed computation system is deployed over a network modeled by a connected and undirected graph $G(V, E)$. G is a general graph. V denotes the whole set of nodes in G and $|V| = N$. E denotes the whole set of edges in G . The graph nodes represent communication endpoints and edges represent bi-directional network links. Let $l(u, v)$ denote a link between node u and node v , where u and v are the end nodes of this link and $u \in V, v \in V$. Let $L(u, v)$ denote the set of links that connect node u and node v , where u and v are the end nodes of these links and $u \in V, v \in V$. Let $d(l(u, v))$ denote the communication distance of the link $l(u, v)$. Let $d(u, v)$ denote the minimum communication distance between two nodes u and v , $d(u, v) = \min \{d(l(u, v)): l(u, v) \in L(u, v)\}$. The length $d(u, v)$ is the minimum number of hops between two nodes u and v , if all edges in G have uniformed distance weight. The M secure data partitions are deployed on M nodes in G . At each node, only one of the M data partitions is located. Any K partitions are required for secure data reconstruction. The numerical relationship between M and K is expressed as $M = \alpha K + \beta$, M, K, α, β are all non-negative integers. $\alpha > 0, \beta \geq 0$. These M nodes in G are defined as the resident set of the M data partitions in G and denoted as $R(N, M, K)$.

The secured distributed computation system support both client read access and update access to the data partitions. We assume that a client access will always choose the closest path. A read access needs to read the closest K data partitions. An update access needs to update all the M data partitions. The update policy is the minimum-spanning-tree write described in [13] and is similar to that in [15]. Essentially, the write issuing node v propagates the data-item along the edges of a minimum spanning tree of the distance graph. The distance graph is a connected weighted graph in which the participants include node v , nodes in $R(N, M, K)$, and the nodes connecting v and $R(N, M, K)$. Each edge represents the distance in the communication network between a pair of participants.

Each node v is associated with a non-negative number $r(v)$ as the number of read access to any K data partitions, issued by node v . A non-negative number $w(v)$ is associated with node v as the number of update accesses to all M data partitions, issued by node v . We say that a partition located node u covers a node v if v read the data partition on node u .

We define that the communication distance of a node $v \in V$ to any nonempty subset of $S \subseteq V$ is $d(v, S) = \min_{u \in S} d(v, u)$. Let $MST(v, i, R(N, M, K))$ denote a minimum spanning tree introduced by a node v to access a set of i nodes in $R(N, M, K)$, $v \in V$. If $v \in R(N, M, K)$, then there are only i nodes in the spanning tree. For a node $v \in V$, the total distance of $MST(v, i, R(N, M, K))$ is defined as $Dist(MST(v, i, R(N, M, K)))$. Let $MSTSet(v, i, R(N, M, K))$ denote the set of all minimum spanning trees induced by a node v to access any i nodes in $R(N, M, K)$. We define $MinDist(v, i, R(N, M, K)) = \min_{MST(v, i, R(N, M, K)) \in MSTSet(v, i, R(N, M, K))} Dist(MST(v, i, R(N, M, K)))$.

The cost of node v to access K nodes in $R(N, M, K)$ is thus defined as $MinDist(v, K, R(N, M, K))$. The cost of node v to access all nodes in $R(N, M, K)$ is $MinDist(v, |R(N, M, K)|, R(N, M, K))$. Let $ReadCost(R(N, M, K))$ denote the read cost from all nodes in V to read the partitioned data, which requiring to access any K nodes in $R(N, M, K)$. Then

$$ReadCost(R(N, M, K)) = \sum_{v \in V} r(v) * MinDist(v, K, R(N, M, K))$$

Let $WriteCost(R(N, M, K))$ denote the cost of all the nodes in V to update the partitioned data, which requiring to access all nodes in $R(N, M, K)$. Then

$$WriteCost(R(N, M, K)) = \sum_{v \in V} w(v) * MinDist(v, |R(N, M, K)|, R(N, M, K))$$

Note that $MinDist(v, |R(N, M, K)|, R(N, M, K)) = Dist(MST(v, |R(N, M, K)|, R(N, M, K)))$. $Dist(MST(v, |R(N, M, K)|, R(N, M, K)))$ is constant for any node $v \in R(N, M, K)$. Let $Cost(R(N, M, K))$ denote the total cost of all nodes in G to access the secure data. $Cost(R(N, M, K))$ is given by

$$Cost(R(N, M, K)) = ReadCost(R(N, M, K)) + WriteCost(R(N, M, K)).$$

Our goal is to optimally allocate the M partitions in G , such that $Cost(R(N, M, K))$ is minimized. It is easy to find out that the (N, M, K) placement problem is NP-hard, since the optimal file allocation problem is NP-hard for networks with general topologies [15]. Next, we first describe a very important property for the (N, M, K) problem and then propose the SOFM-like and the heuristic algorithms for this problem. Storage cost of the data partitions, message losses, and node failures are not considered in this paper.

3. Replica Placement by a SOFM-like Algorithm

3.1 The SOFM Neural Network

We consider using a modified Self-Organizing Feature Map (SOFM) algorithm [5] for the secure data placement problem since it can deal with large scale problems with relative low computation complexity. Kohonen's SOFM typically has two layers of nodes, the input layer and the Kohonen output layer. The input layer is fully connected to the output layer. An input vector is denoted as $\mathbf{x}_v = [x_{v1}, \dots, x_{vl}]^T$, $v = 1, \dots, N$, where l is the dimension of input vector. Each neuron in the output layer is represented by a prototype vector $\mathbf{w}_i = [w_{i1}, \dots, w_{il}]^T$, $i = 1, \dots, M$, where M is the number of output neurons. The SOFM is trained iteratively. At each iteration, an input vector \mathbf{x}_v is randomly chosen from the input data set. The output neurons compete to be selected as the winning neuron, or the best-matching neuron, according to their similarities to the input vector. Usually, the similarity is evaluated by some distance metric (e.g. Euclidean distance). The index of the winning neuron with respect to input vector \mathbf{x}_v , denoted by $Winner(\mathbf{x}_v)$, is determined if

$$Winner(\mathbf{x}_v) = \arg \min_{i \in \{1, \dots, M\}} \|\mathbf{x}_v - \mathbf{w}_i\|,$$

where $\|\cdot\|$ denotes the Euclidean norm.

After the winning neuron is identified, $Winner(\mathbf{x}_v)$ as well as its neighboring neurons is activated and ready to update their synaptic weights. Let $N_{Winner}(t)$ denote the topological neighborhood of the winning neuron $Winner(\mathbf{x}_v)$ at time t . At the beginning of the training process, the radius of neighborhood is chosen to be fairly large for learning speed and it is reduced for convergence as $t \rightarrow \infty$. In the adaptive process, the weights of all neurons are adjusted by the following rule:

$$\mathbf{w}_i(t+1) = \begin{cases} \mathbf{w}_i(t) + \alpha(t)\eta_{i, Winner}(t)[\mathbf{x}_v - \mathbf{w}_i(t)], & i \in N_{Winner}(t) \\ \mathbf{w}_i(t), & \text{otherwise} \end{cases} \quad (1)$$

where $\alpha(t)$ is the adaptation coefficient and $\eta_{i, Winner}(t)$ is the neighborhood coefficient kernel centered on the winning neuron. For the same reason as for $N_{Winner}(t)$, $\alpha(t)$ and $\eta_{i, Winner}(t)$ are also reduced as $t \rightarrow \infty$. The training process continues until the weights associated with the neurons are stabilized.

3.2 The RP-SOFM Algorithm for the Secure Data Placement

In this section, we present a secure data placement algorithm based on SOFM, denoted as RP-SOFM for short. In RP-SOFM, the dimension of an input vector is 1. Let $G = (V, E)$ denote the computer network graph in which the M partitions will be placed. The input layer of RP-SOFM contains N input values x_1, x_2, \dots, x_N where $N = |V|$. Each input value x_i represents a node index in G . Each output neuron of the RP-SOFM represents a node in G on which a partition is to be placed. Let w_j denote an output neuron. The total number of neurons in RP-SOFM is M . Initially,

output neurons are assigned with different indexes of the computer network nodes randomly. Then the neurons are trained to change their associations to the network nodes based on their proximities between the network nodes.

3.2.1 Competitive learning

In the competitive learning process, K neurons best-matching the input value are selected. The best-matching relationship in RP-SOFM is defined as the shortest path distance between the nodes in the computer network the input value and K neurons are representing (To speed the learning process, all pairs shortest paths between network nodes are computed and saved in advance). Let $d(a, b)$ denote the shortest path distance between a and b , $a, b \in V$. Define $d(a, B) = \min_{b \in B} d(a, b)$ as the distance between a and

set B , where $b \in B \wedge B \subset V$. The process to select K nearest neurons is the process to construct a minimum spanning tree (MST) for the sub-graph containing x_i and K nearest neurons. The order that the neurons join into the set U gives the rank of distances between x_i and the K neurons. The product of the sum of weights (distances) of all edges in the tree and the read weight of x_i , $ReadCost_{x_i}$, gives the total read cost of x_i under the data partition placement represented by the current status of the RP-SOFM.

3.2.2 Adaptive process

In RP-SOFM, each neuron maintains a distance list between all input values and itself. Let $DistList_{x_i, w_j}$ denote the distance between x_i and w_j maintained in the distance list. Initially, $DistList_{x_i, w_j}$ is set as the shortest path distance between x_i and w_j . During the adaptive process, $DistList_{x_i, w_j}$ between the current input x_i and all neurons w_j , $1 \leq j \leq M$, are updated according the following two factors.

The first factor is $f_R(x_i, w_j)$ that gives the distance deduction proportion due to the cost of x_i reading its K nearest partitions.

$$Prop_R(x_i) = \frac{ReadCost_{x_i}}{AveReadCost_{x_i} * K + AveWriteCost_{x_i} * M}$$

$$f_R(x_i, w_j) = \alpha^R(t)\eta_{w_j}^R(t)Prop_R(x_i) \quad (2)$$

where $ReadCost_{x_i}$ gives the K nearest partition read costs of x_i and is computed as in the previous subsection. $AveReadCost_{x_i}$ is computed as follows:

$$AveReadCost_{x_i} = ReadWeight_{x_i} * \sum_{\forall x_j \in V \wedge x_j \neq x_i} d(x_i, x_j) / N$$

where $ReadWeight_{x_i}$ is the read weight of x_i . Similarly $AveWriteCost_{x_i}$ is computed as follows:

$$AveWriteCost_{x_i} = WriteWeight_{x_i} * \sum_{\forall x_j \in V \wedge x_j \neq x_i} d(x_i, x_j) / N$$

where $WriteWeight_{x_i}$ is the write weight of x_i . Thus, $Prop_R(x_i)$ gives the proportion of current cost of reading K

partitions to the average cost for reading K partitions and writing M partitions. $\alpha^R(t)$ is the adaptation coefficient for $f_R(x_i)$. $\alpha^R(t)$ decreases as $t \rightarrow \infty$. $\eta_{w_j}^R(t)$ is the neighborhood coefficient kernel centered on the winning neuron and is defined as follows:

$$\eta_{w_j}^R(t) = \begin{cases} \exp\left(-\frac{(K - \text{Rank}(w_j))^2}{2(\sigma^R(t))^2}\right), & K * \Lambda^R(t) < \text{Rank}(w_j) \leq K \\ 0, & \text{otherwise} \end{cases}$$

where $\sigma^R(t)$ decreases as $t \rightarrow \infty$ and $\text{Rank}(w_j)$ gives the rank of distance between w_j and the winner neuron, e.g. $\text{Rank}(w_j)=1$ if w_j is the nearest network node to the winner. Note that the value of $\eta_{w_j}^R(t)$ for a nearer node is smaller than a distant node. This results in bigger distance deduction for a distant node. Intuitively, K partitions tend to close to each other in a good placement configuration. So “dragging” distant nodes with bigger strength (bigger distance deduction) tends to make these nodes closer. In addition, we control the number of neighbors by $\Lambda^R(t)$. Initially, $\Lambda^R(t)$ is set to 0 so that all K nearest nodes can be dragged. With the time increase, $\Lambda^R(t) \rightarrow 1$ so that fewer and fewer distant nodes can be dragged. This method accelerates the convergence of RP-SOFM.

Similar to $f_R(x_i, w_j)$, the second factor $f_W(x_i, w_j)$ gives the deduction proportion for the write cost of x_i to the M partitions represented by the current status of neurons.

$$f_W(x_i, w_j) = \frac{\alpha^W(t) \eta_{w_j}^W(t) \text{WriteCost}_{x_i}}{\text{AveReadCost}_{x_i} * K + \text{AveWriteCost}_{x_i} * M} \quad (3)$$

The items in $f_W(x_i, w_j)$ is defined similarly as those in $f_R(x_i, w_j)$, where WriteCost_{x_i} denotes the cost of x_i writing M partitions according to the current status of neurons. $\eta_{w_j}^W(t)$ is defined similarly as follows:

$$\eta_{w_j}^W(t) = \begin{cases} \exp\left(-\frac{(M - \text{Rank}(w_j))^2}{2(\sigma^W(t))^2}\right), & M * \Lambda^W(t) < \text{Rank}(w_j) \leq M \\ 0, & \text{otherwise} \end{cases}$$

The reason that dragging distant nodes with bigger strength is the same as that in $\eta_{w_j}^W(t)$ since intuitively in a good secure data placement, M partitions tend to close to each other. We also use $\Lambda^W(t)$ to make fewer and fewer distant nodes to be dragged.

Combining the two factors together, $\text{DistList}_{x_i, w_j}$ is deducted by the following rule:

$$\text{DistList}_{x_i, w_j} = \text{DistList}_{x_i, w_j} * (1 - f_R(x_i, w_j) - f_W(x_i, w_j)) \quad (4)$$

After the distance update, all neurons are checked to see if they should change the association relationship to the network nodes. That is, the distance between x_i and every other network node is compared with $\text{DistList}_{x_i, w_j}$ and find

the node, say a , whose distance is nearest to $\text{DistList}_{x_i, w_j}$. If $a \neq w_j$, change the value of w_j to a and set w_j 's $\text{DistList}_{x_i, w_j}$ to be $\text{DistList}_{x_i, a}$ between w_j and all x_i , $1 \leq i \leq N$; otherwise, no change for the association relationship occurs and the deduction to $\text{DistList}_{x_i, w_j}$ is accumulated for the next iteration. Note that since multiple neurons are checked to see if they should change their associations, three special cases may occur: 1) A neuron w_i intends to change its association to a network node a , but a is currently associated by another neuron w_j and w_j does not intend to change its association in this iteration. In this case, w_i simply does not change its association in current iteration. 2) Multiple neurons intend to change their association to the same network node. In this case, one neuron is arbitrary selected to change its association and other neurons remain unchanged. 3) A cycle is formed between multiple neurons who are changing their associations, e.g. a neuron w_i currently associating to b intends to change its association to a while another neuron w_j currently associating to a intends to change its association to b . In this case, no neuron is allowed to change its association since such changes have no effect to the association configuration of the whole RP-SOFM. In fact, during our simulation, this case never occurs since all neurons are dragged along the same direction to the input so that no cycle could be formed.

3.2.3 The RP-SOFM Algorithm

The whole RP-SOFM algorithm is listed as follows:

- Step 0. Initialize $\alpha^R(0)$, $\alpha^W(0)$, $\sigma^R(0)$, $\sigma^W(0)$, $\Lambda^R(0)$, $\Lambda^W(0)$, Z_1 (total iteration cycles), Z_2 (association relationship stable criterion). Let N denote the total number of nodes in the computer network graph G , M denote the total number of output neurons and K denote the minimum number of partitions a node should read. Initialize neurons w_j , $1 \leq j \leq M$, in the Kohonen output layer, i.e. randomly assign different indexes of the computer network nodes to w_j , $1 \leq j \leq M$.
- Step 1. Select a x_i as an input and present it to the output neurons w_j , $1 \leq j \leq M$.
- Step 2. Find the K nearest neurons to the input value.
- Step 3. Update $\text{DistList}_{x_i, w_j}$ for all w_j , $1 \leq j \leq M$. Check all w_j , $1 \leq j \leq M$, to see if the value of w_j (their association relationship to the network nodes) should be changed. If a w_j should be changed according to $\text{DistList}_{x_i, w_j}$ and can be changed according to the three special cases, change w_j to the new association and reset $\text{DistList}_{x_i, w_j}$ for the new value of w_j ; otherwise, no change occurs.
- Step 4. If not all x_i are presented in the current iteration, go to Step 1; otherwise, go to Step 5.
- Step 5. If the association relationship does not change within Z_2 iterations for all w_j , $1 \leq j \leq M$, or $t > Z_1$, stop;

Otherwise, set $t = t + 1$, update $\alpha^R(t)$, $\alpha^W(t)$, $\sigma^R(t)$, $\sigma^W(t)$, $\Lambda^R(t)$, $\Lambda^W(t)$, and go to Step 1.

4. A Heuristic Algorithm for the Secure Data Placement Problem

We further propose a heuristic algorithm for the secure data placement problem. Intuitively, K partitions tend to be close to each other and are centered to other nodes reading them while total M partitions also tend to be close to each other and are centered to other nodes. In this heuristic, we use the agglomerative clustering approach [14] to allocate partitions. Let $M = \alpha K + \beta$. The basic idea of the algorithm is to first evenly partition the network graph into α clusters, find the centroid of each cluster and then find K (or $K+b$, $b \leq \beta$) nodes within each cluster nearest to the cluster's centroid. In this way, the nodes within each cluster locally read the K partitions in the cluster so that the total read cost could be minimized. To be able to further reduce the total write cost, we then find the centroid of the whole network graph, adjust the centroid of each cluster closer to the centroid of the whole graph, finally adjust the K (or $K+b$, $b \leq \beta$) partition nodes within each cluster close to the adjusted centroid. The detailed process is described in the following:

Step 0. Initially, every node in the computer network graph is a cluster.

Step 1. Compute the average linkage distances between all clusters.

The average linkage distance between two clusters C_p and C_q is computed as follows:

$$D(C_p, C_q) = \frac{\sum_{x_i \in C_p, x_j \in C_q} d(x_i, x_j)}{|C_p| * |C_q|},$$

where $d(x_i, x_j)$ gives the shortest path distance between x_i and x_j .

Step 2. Let $D_{min} = D(C_x, C_y)$ denote the minimum average linkage distance of all clusters. Merge C_x and C_y into one cluster.

Step 3. Return to Step 1 until there are only α clusters left.

Step 4. Find the weighted centroid of the whole network graph according to the write weights of nodes, and then find the network node closest to the centroid.

a. Let W denote the weighted centroid of the network graph G . Find W as follows:

$$W = \frac{1}{|G|} \sum_{x_i \in G} \bar{x}_i, \text{ where } \bar{x}_i = WriteWeight_{x_i} * [d(x_i, x_1), d(x_i, x_2), \dots, d(x_i, x_N)]^T.$$

b. Find the network node closest to the centroid as follows:

$$x^* = \arg \min_{x_i \in G} \|x_i - W\|, \text{ where } x^* \text{ denotes the node closest}$$

to the centroid W .

Step 5. Find the weighted centroid of each cluster according to the read weights of nodes in each cluster. Then for each cluster, find the network node x_k^* within the cluster C_k closest to C_k 's centroid. Finally, adjust x_k^* closer to x^* .

a. Let CN_k denote the weighted centroid of C_k . Find CN_k for each C_k as follows:

$$CN_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} \bar{x}_i, \text{ where } \bar{x}_i = ReadWeight_{x_i} * [d(x_i, x_1), d(x_i, x_2), \dots, d(x_i, x_{|C_k|})]^T, \text{ where } \forall x_1, x_2, \dots, x_{|C_k|} \in C_k.$$

b. Find the network node x_k^* within C_k closest to the centroid CN_k as follows:

$$x_k^* = \arg \min_{x_i \in C_k} \|x_i - CN_k\|$$

c. Adjust x_k^* closer to x^* as follows:

i. Adjust the shortest path distance of x_k^* to x^* as follows:

$$d_{adjust} = d(x_k^*, x^*) \left(1 - \frac{\sum_{x_i \in C_k} WriteWeight_{x_i}}{\sum_{x_i \in C_k} ReadWeight_{x_i} + \sum_{x_i \in C_k} WriteWeight_{x_i}} \right)$$

ii. Find the network node x_k^{**} within C_k whose distance to x^* closest to d_{adjust} as follows:

$$x_k^{**} = \arg \min_{x_i \in C_k} \|d(x_i, x^*) - d_{adjust}\|$$

Step 6. Allocate additional b_k partitions to a cluster C_k according to the cardinality of C_k , $1 \leq k \leq \alpha$, where $\sum b_k = \beta$.

Step 7. For each cluster C_k , find $K+b_k-1$ nodes in C_k closest to each x_k^{**} . The method of finding closest nodes is the same as that in Section 2.

5. Numerical Results

This section evaluates the efficiency of the RP-SOFM algorithm and the heuristic algorithm to allocate M partitions for a given network graph. In our evaluation model, the underlying network topology is created by the Inet generator [3]. One reason that we choose the Inet generator is that it can generate the coordinates of each node so that we can draw the topology diagram and check the efficiency of algorithms visually. The original generator has the lower bound of the total nodes in the network. We removed the bound so that the graph with fewer nodes can be created. We also added the function for generating read and write weight for each node to the generator. The read and write weights are generated randomly with a pre-set ratio. In this paper, the parameters in RP-SOFM are set as follows: $K_1 = 200$, $K_2 = 12000$. Initially $\alpha^R(0)=0.1$, $\alpha^W(0)=0.9$, $\sigma^R(0)=0.1$, $\sigma^W(0)=0.9$, $\Lambda^R(0)=0$, $\Lambda^W(0)=0$. At every 100 iteration cycles, all parameters are updated as follows: $\alpha^R=0.1*\alpha^R$; $\alpha^W=0.1*\alpha^W$; $\sigma^R=0.1*\sigma^R$; $\sigma^W=0.9*\sigma^W$; $\Lambda^R = \Lambda^R + 0.02$, until $\Lambda^R = 0.99$; $\Lambda^W = \Lambda^W + 0.02$, until $\Lambda^W = 0.99$.

To evaluate the efficiency of both algorithms, we developed a program to exhausted search the optimal solution. However, as expected, this program can only run for a small graph (the number of nodes in the graph is around 20) with acceptable stop speed.

We use the total cost (total read cost + total write cost) as the metric to compare the efficiency of different algorithms. We vary several parameters: (1) N : the total node number in the network. (2) $Ratio$: the average ratio of read weight to write weight. (3) D : average connection degree of each node. (4) M : the total number of partitions to be placed. (5) K : the lower bound of partitions should be read by a network node.

5.1 The Efficiency of the RP-SOFM and the Heuristic Algorithms

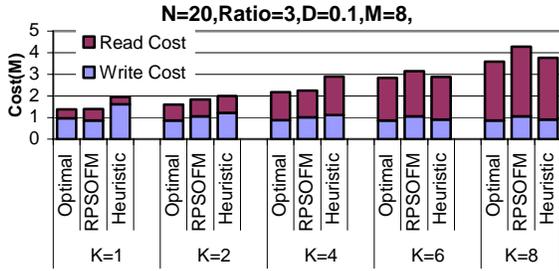


Fig. 1. Cost for different K ($N:20$, Ratio: 3, $D: 0.1$, $M:8$)

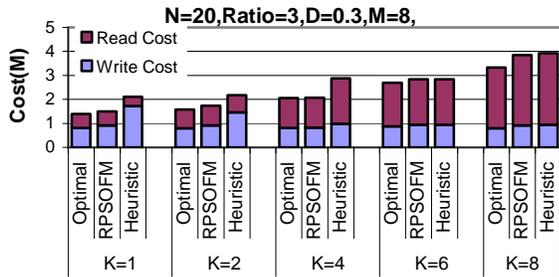


Fig. 2. Cost for different K ($N:20$, Ratio: 3, $D: 0.3$, $M:8$)

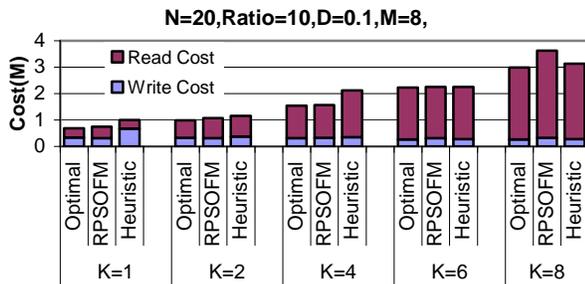


Fig. 3. for different K ($N:20$, Ratio: 10, $D: 0.1$, $M:8$)

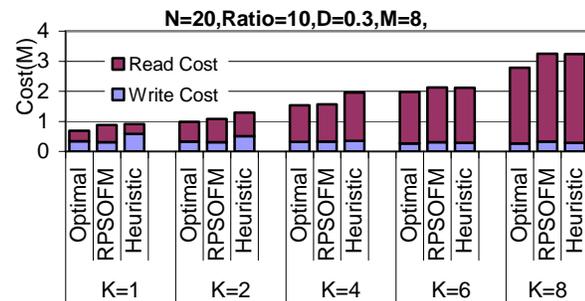


Fig. 4. Cost for different K ($N:20$, Ratio: 10, $D: 0.3$, $M:8$)

Figure 1-4 show the cost comparisons under different parameter combinations. From the result, we can see that almost all cases, the solutions found by the RP-SOFM and the heuristic algorithms are very close to the optimal solution. When K is smaller than or equal to $M/2$, the RP-SOFM is better than the heuristic algorithm while when K becomes larger, the heuristic algorithm becomes better than the RP-SOFM algorithm. The reason could be attributed to the fact that when K is relatively big, more neurons will be dragged by each input due to the read cost factor $f_R(x_i, w_j)$ so that the K partitions allocated may not be locally close to each other. While when K is relatively small, the heuristic algorithm tends to partition the graph into relatively many clusters. These clusters may not be evenly partitioned so that although the read cost is relatively small, the write cost could be very high since total M partitions are not close to each other. Note that when $K=4$, the heuristic algorithm gives worse results comparing to the RP-SOFM. This could be attributed to the fact that the two clusters partitioned by the heuristic algorithm are very unbalanced resulting in higher read costs. So, this experimental result suggests that the two algorithms could be further improved by controlling some parameters, e.g. for the heuristic algorithm, we should adaptively control the total number of clusters and the total number of nodes within each clusters so that the graph can be evenly partitioned.

5.2 The Scale-up of the RP-SOFM and the Heuristic Algorithms

We further run the RP-SOFM and the heuristic algorithms for the graph containing 100 nodes to see how well they scale up with the increasing number of nodes. Figure 5-8 show the cost comparisons under a graph containing 100 nodes. Due to the same reason as explained above, the results are in the same pattern as that of figure 1-4. From another aspect, the solution differences between the two algorithms are relatively not big. This suggests that both algorithms can give good solutions for a larger graph.

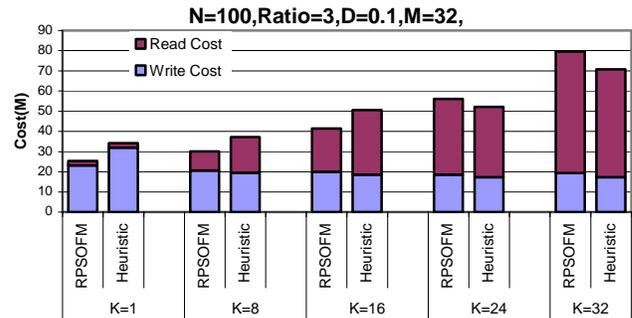


Fig. 5. Cost for different K ($N:100$, Ratio: 3, $D: 0.1$, $M: 32$)

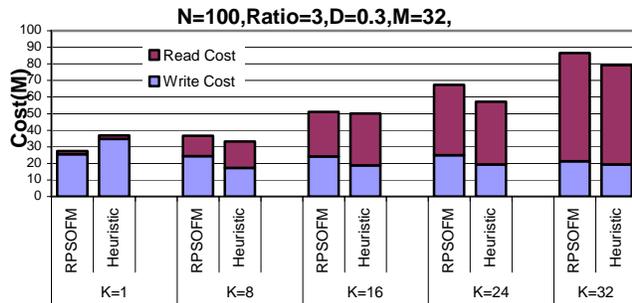


Fig. 6. Cost for different K (N:100, Ratio: 3, D: 0.3, M: 32)

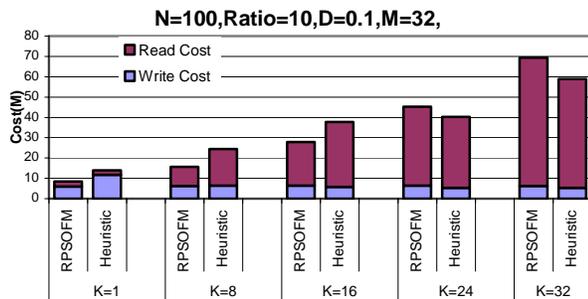


Fig. 7. Cost for different K (N:100, Ratio: 10, D: 0.1, M: 32)

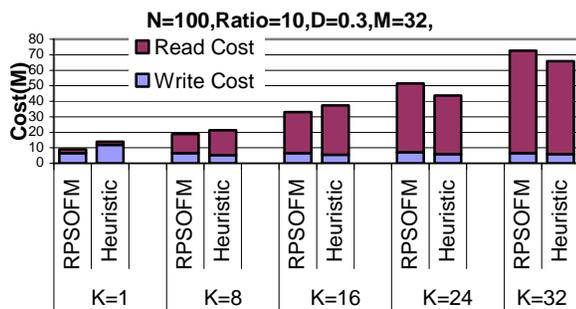


Fig. 8. Cost for different K (N:100, Ratio: 10, D: 0.3, M: 32)

6. Conclusion

In this paper, we considered the allocation of partitioned data objects in distributed systems. We defined the problem as finding M nodes in the network to host the data partitions so that the total cost is minimized to read K partitions and update M partitions by all nodes in the system. This problem is NP-hard. We then propose a SOFM-like algorithm and a heuristic algorithm to solve this problem. Numerical studies are conducted extensively and the results show that both algorithms are effective for this problem. The numerical results also show both algorithms could be further improved. We plan to continue to improve both algorithms and test them on different network topologies.

References

[1] Y. Deswarte, et. al. Intrusion tolerance in distributed computing systems. *IEEE Symposium on Research in Security and Privacy*. 1991.
 [2] R. Gennaro. "Theory and practice of verifiable secret sharing," Ph.D-thesis, MIT, 1995.

[3] C. Jin, Q. Chen, S. Jamin, "Inet: Internet Topology Generator," Tech. Rep. CSE-TR-433-00, EECS Department, University of Michigan, 2000.
 [4] K. K. Kalpakis, et. al. "Optimal placement of replicas in trees with read, write, and storage costs," *IEEE Transactions on Parallel and Distributed Systems*. Vol 12, No. 6. 2001.
 [5] T. Kohonen, "Self-organizing maps," 2nd Edition, Springer Verlag, 1997.
 [6] H. Krawczyk. Distributed fingerprints and secure information Dispersal. Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing (PODC'93), 1993.
 [7] H. Krawczyk. Secret Sharing Made Short. Crypto'93.
 [8] S. Lakshmanan, M. Ahamad, and H. Venkateswaran. Responsive security for stored data. *IEEE Transactions on Parallel and Distributed systems*. Vol 14, No. 9, 2003.
 [9] M. Malkin, et. al. Building intrusion tolerant applications. *DARPA Information Survivability Conference & Exposition I*. 2000.
 [10] A. Mei, L. V. Mancini, and S. Jajodia. Secure dynamic fragment and replica allocation in large-scale distributed File systems. *IEEE Transactions on Parallel and Distributed systems*. Vol 14, No. 9, 2003.
 [11] M. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of ACM*. Vol. 36, No. 2. 1989.
 [12] A. Shamir. "How to share a secret," *Communication of the ACM*. Vol. 22, 1979.
 [13] M. Tu. A data management framework for secure and dependable data grid. *Ph.D Dissertation, UT Dallas*. <http://www.utdallas.edu/~tumh2000/ref/Thesis-Tu.pdf>. 2006.
 [14] J. Vesanto and E. Alhoniemi, "Clustering of the Self-Organizing Map," *the IEEE Transactions on Neural Networks*, Vol. 11, No. 3, May 2000, pp. 586-600
 [15] O. Wolfson and A. Milo. "The Multicast Policy and Its Relationship to Replicated Data Placement," *ACM Trans. Database Systems*. Vol.16, No.1. 1991.
 [16] J. Wylie, M. Bakkaloglu, V. Pandurangan, M. Bigrigg, S. Oguz, K. Tew, C. Williams, G. Ganger, and P. Khosla. Selecting the right data distribution scheme for a survivable storage system. *Technical Report CMU-CS-01-120*. Carnegie Mellon University. 2000
 [17] J. Xu, et. al. "Optimal replica placement on transparent replication proxies for read/write data," *Proc. the 21st IEEE Int. Performance, Computing, and Communications Conference (IPCCC '02)*. 2002