An Efficient Parallel Anomaly Detection Algorithm Based on Hierarchical Clustering

Ren Wei-wu, Hu Liang, Zhao Kuo, Chu Jianfeng
College of Computer Science and Technology, Jilin University, Changchun, China
E-mail: Chujf@jlu.edu.cn

Abstract—For the purpose of improving real time and profiles accuracy, a parallel anomaly detection algorithm based on hierarchical clustering has been proposed. Training and predicting are two busiest processes and they are parallel designed and implemented. Moreover, an abnormal cluster feature tree is built to dig anomalies from normal profiles. A series of experiment results on well-known KDD Cup 1999 data sets indicate that the improved algorithm has superior performance in both detection and real time.

Index Terms—parallel algorithm; hierarchy clustering; abnormal cluster feature tree; normal profiles

I. INTRODUCTION

With the number of intrusion and hacking incidents around the world on the rise, the importance of having dependable intrusion detection systems in place is greater than ever. An intrusion detection system is designed to detect several types of abnormal behaviors that can compromise the security and trust of a computer system.

Now the main intrusion detection technology is divided into two categories: misuse detection [1-2] and anomaly detection [3-7]. Misuse detection encodes the known attacks into the signatures and detects attacks whose signatures are known and have been encoded. Intrusion detection system based on misuse detection can not detect unknown attacks. The process of signing attacks is enormous cost for systems. Unlike misuse detection, anomaly detection builds the normality profiles on the basis of normal behaviors of users, often using machine learning or data mining techniques. In the process of detection, online traffic is matched with the normality profiles, and deviations are marked as anomalies. Since no knowledge of attacks is used to train the normality profiles, anomaly detection can detect previously unknown attacks. Therefore anomaly detection is hotspot in the field of intrusion detection.

Many popular technologies are applied in the field of anomaly detection. Clustering algorithm [21-23] is a successful application in the field of anomaly detection. Density clustering [9-12] and hierarchical clustering [12-15] are two outstanding representative kinds of clustering algorithms. Density clustering can build arbitrary shape cluster, but calculation is too complex. Hierarchical clustering has good efficiency and is easy to implement incremental algorithm, but only build spherical cluster. So profiles of hierarchical are generally less precise than profiles of density clustering.

Moreover, the growth of network flow makes the real time of anomaly detection algorithms be involved. The old serial algorithm can not meet the real time of detection, and the development of CPU has officially entered the era of multi-core. Along with the update multi-core technology, parallel algorithm must be a new way to solve the problem of real time.

Parallel algorithms [16-20] are valuable because of substantial improvements in multiprocessing systems and the rise of multi-core processors. There are two ways parallel processors communicate, shared memory or message passing. Shared memory processing needs additional locking for the data, imposes the overhead of additional processor and bus cycles, and also serializes some portion of the algorithm. Message passing processing use channels and message boxes but this communication adds transfer overhead on the bus, additional memory need for queues and message boxes and latency in the messages. Designs of parallel processors use special buses like crossbar so that the communication overhead will be small but it is the parallel algorithm that decides the volume of the traffic.

For the purpose of improving real time and profiles accuracy of hierarchical clustering, a parallel anomaly detection algorithm based on hierarchical clustering has been proposed in this paper. Training process and predicting process, which consume a lot of resources, are parallel designed and implemented. Moreover, an abnormal cluster feature tree is built to dig anomalies from normal profiles. It can compensate the lack of profiles accuracy of hierarchical clustering.

The rest of paper is organized as follows. In section 2, we introduce some basic concepts. Section 3 presents details of parallel algorithms. Section 4 presents our experiments results and analysis. Finally, we summarize our conclusions and future work in section 5.

II. BASIC CONCEPTS

The related basic concepts are introduced at first:

Definition 0: we assume that each sample point lies in the k-dimensional Euclidean space, and a cluster
\( C = \{ \vec{v}_i \mid i = 1 \ldots n \} \) is defined as the collection of \( \vec{v}_i \), \( \vec{v}_i \) is a vector which starts with the origin, and ends with the \( d_i \).

**Definition 1:** For a cluster \( C = \{ \vec{v}_i \mid i = 1 \ldots n \} \) and the \( k \)-dimensional vector \( \vec{v}_i \) in the cluster, its center is defined as:

\[
\vec{v}_0 = \frac{\sum_{i=1}^{n} \vec{v}_i}{n} \tag{1}
\]

The center describes the distribution of points in the cluster.

**Definition 2:** for a cluster \( C = \{ \vec{v}_i \mid i = 1 \ldots n \} \) and the \( k \)-dimensional vector \( \vec{v}_i \) in the cluster, its radius \( R(C) \) is defined as:

\[
R(C) = \sqrt{\frac{\sum_{i=1}^{n} (\vec{v}_i - \vec{v}_0)^2}{n}} \tag{2}
\]

Radius \( R(C) \) describes the clustering degree of points in cluster (i.e., the mean distance from all the points to center \( \vec{v}_0 \)).

**Definition 3:** for two clusters \( C_1 \) and \( C_2 \), and their centers \( \vec{v}_0 \) and \( \vec{v}_0' \), the distance from \( C_1 \) to \( C_2 \) is defined as the Euclidean distance from \( \vec{v}_0 \) to \( \vec{v}_0' \).

**Definition 4:** Cluster Feature (CF) is a triple: \( CF = \{ n, \bar{s}, ss \} \), among which \( n \) is the number of vector \( \vec{v}_i \) in the cluster \( C \), and \( \bar{s} \) is the linear sum of all the vectors in the cluster \( C \) (i.e., \( \bar{s} = \sum_{i=1}^{n} \vec{v}_i \) ), and \( ss \) is the sum of squares of all the vectors in the cluster \( C \) (i.e., \( ss = \sum_{i=1}^{n} \vec{v}_i^2 \) ). CF describes the overall feature of cluster.

**Definition 5:** for two clusters \( C_1 \) and \( C_2 \), and their sum \( C_1+C_2 \) represents the merger of two clusters (i.e., all the vectors in two clusters are merged into one new cluster)

**Theorem 1:** for two clusters \( C_1 \) and \( C_2 \), and their CF \( CF_1 = \{ n_1, \bar{s}_1, ss_1 \} \) and \( CF_2 = \{ n_2, \bar{s}_2, ss_2 \} \), then CF of \( C_1+C_2 \) is \( CF_{1+2} = \{ n_1 + n_2, \bar{s}_1 + \bar{s}_2, ss_1 + ss_2 \} \).

**Proof:** the proof of theorem 1 is simple. Due to definition 4 and definition 5, theorem 1 is not difficult to show.

**Theorem 2:** cluster \( C \) and its \( CF = \{ n, \bar{s}, ss \} \), then its center can be represent as \( \vec{v}_0 = \frac{\bar{s}}{n} \)

**Proof:** due to definition 1 and definition 4, \( \bar{s} = \sum_{i=1}^{n} \vec{v}_i \) , then \( \vec{v}_0 = \frac{\bar{s}}{n} \).

**Theorem 3:** cluster \( C \) and its \( CF = \{ n, \bar{s}, ss \} \), then its radius can be represent as

\[
R(C) = \sqrt{\frac{ss - \bar{s}^2}{n}} \tag{3}
\]

**Proof:** due to definition 2, \( R(C) = \sqrt{\sum_{i=1}^{n} (\vec{v}_i - \vec{v}_0)^2} \).

After vector expansion: \( R(C) = \sqrt{\frac{\sum_{i=1}^{n} \sum_{j=1}^{d} (\vec{v}_{ij} - \bar{v}_{ij})^2}{n}} \),

Then \( R(C) = \sqrt{\frac{\sum_{i=1}^{n} \sum_{j=1}^{d} (\vec{v}_{ij}^2 + \bar{v}_{ij}^2 - 2\bar{v}_{ij} \bar{v}_{ij})}{n}} \),

Due to \( \bar{v}_0 = \sum_{i=1}^{n} \vec{v}_0' \),

Then \( R(C) = \sqrt{\frac{\sum_{i=1}^{n} \sum_{j=1}^{d} \bar{v}_{ij}^2 + \sum_{i=1}^{n} \bar{v}_{ij}^2 - 2\sum_{i=1}^{n} \bar{v}_{ij} \bar{v}_{ij}}{n}} \),

Due to definition 4: \( \bar{s} = \sum_{i=1}^{n} \vec{v}_i = n\bar{v}_0 \) and \( \bar{s} = \sum_{i=1}^{n} \vec{v}_i = n\bar{v}_0 \), and they are substituted in the above equation, then \( R(C) = \sqrt{\frac{ss - \bar{s}^2}{n}} \).

It can be seen by the above theorem. The center and radius of cluster can be calculated by its CF without having to know each specific vector. Therefore, the algorithm to run needs not to keep original data.

**Cluster Feature tree (CF tree):** CF is organized in a tree structure. A CF tree represents the present user’s profile. CF tree is height balanced tree. For each leaf node, its height is the same. There are three parameters: the maximum number of branches \( B \), threshold \( T \) and the maximum number of cluster \( M \). The node of CF tree is not same with the other nodes. The value of its which is pointed by \( pChild \). In particular, the leaf node of \( pChild \) does not represent the merger of clusters but specific clusters. Each cluster in the leaf node must satisfy the threshold \( T \) (i.e., the radius of each cluster must be less than \( T \)). The number of clusters in the whole leaf nodes must be less than \( M \). To sum up, the tree which satisfies the above conditions can be called Cluster Feature tree (CF tree).
NCF tree: NCF tree is a CF tree which is composed of CFs derived from normal behaviors. NCF tree represents user’s profiles of normal behaviors.

ACF tree: ACF tree is a CF tree which is composed of CFs derived from abnormal behaviors. ACF tree represents the known type of attack.

In the process of generating profiles, ACF tree is constructed, in other words, the vector \( v \) is considered as a cluster which only contains one vector. And the following vectors are dynamically inserted into the CF tree by procedure Insert(CF,p). The pseudocode of procedure Insert(CF,p) is described as follows:

```
Procedure Insert(CF, p)
    Procedure WorkingThreadRun ()
    
    initial root is an empty CF
    create a empty CF tree;
    receive traffic(thread will sleep until the coming of traffic);
    extract \( \bar{V} \) from traffic and generate CF;
    Insert(CF, root); initial root is an empty CF
    if CF tree is greater than M
    collect all the clusters of leaf nodes;
    send the above clusters to main thread;
    clean CF tree;
    
    Procedure WorkingThreadRun ()
    
    Traverse all the items of node p, find the nearest cluster CF;
    if(p is leaf node) insert(CF, p); if(merge CF into CF, and new cluster still satisfies T)
    merge CF into CF, according to Theorem 1;
    else add a new item [CF, NULL] in node p;
    else insert(CF, pChild) //recursive call
    merge CF into CF ;
    if(pChild, violate B) call SplitNode(pChild,)//split pChild, into pChild1 and pChild2;
    delete item[CF, pChild] from node p;
    create CF1, and CF2, they are separately cluster mergers of pChild1 and pChild2;
    create two new items [CF1, pChild1] and [CF2, pChild2];
    insert two new items into node p;
```

Initially, CF tree only has an empty root. With the coming of new vector \( \bar{v} \), a temporary CF={1, \( \bar{v}, \bar{v}^2 \)} is constructed, in other words, the vector \( \bar{v} \) is considered as a cluster which only contains one vector. And the following vectors are dynamically inserted into the CF tree by procedure Insert(CF,p). The pseudocode of procedure Insert(CF,p) is described as follows:

```
Procedure Insert(CF, p)
    
    Traverse all the items of node p, find the nearest cluster CF;
    if(p is leaf node) insert(CF, p); if(merge CF into CF, and new cluster still satisfies T)
    merge CF into CF, according to Theorem 1;
    else add a new item [CF, NULL] in node p;
    else insert(CF, pChild) //recursive call
    merge CF into CF ;
    if(pChild, violate B) call SplitNode(pChild,)//split pChild, into pChild1 and pChild2;
    delete item[CF, pChild] from node p;
    create CF1, and CF2, they are separately cluster mergers of pChild1 and pChild2;
    create two new items [CF1, pChild1] and [CF2, pChild2];
    insert two new items into node p;
```
It can be drawn from the above pseudocode that the function of Insert(CF,P), as a recursion, needs to check whether the constraint B is satisfied or not when it returns. Therefore, the constraint B is split into two from bottom to top. If the root is against B, the tree would grow itself.

The above algorithm calls the sub procedure SplitNode(p), and the pseudocode shows details of SplitNode(p) as follows:

```plaintext
Procedure SplitNode(p)
    // Traversing node p and find the two farthest nodes CF_i and CF_j;
    // Generate two new nodes p1 and p2;
    // Insert separately [CF_i,pChild] and [CF_j,pChild] into node p1 and node p2;
    // for all the CFs of node p except CF_i and CF_j;
    if( dis(CF_i,CF_j) ≤ dis(CF_i,CF_j) )
        insert [CF_i,pChild] into p1;
    else
        insert [CF_j,pChild] into p2;
    return p1 and p2;
```

It can be drawn from the above algorithm that with the coming of new data, the number of leaf node increases until the constraint M is violated. Then CF tree needs to be rebuilt.

The process of rebuilding CF can be described as follows: first, the constraint T is relaxed. Then CFs of the entire leaf nodes are collected and CF tree is cleaning. Next the collected CFs are inserted into the new CF tree by calling the procedure of Insert(CF, p). Finally the CF tree is rebuilt.

Threshold T is one of three parameters of CF tree. To relax T can merge clusters and can reduce the number of leaf nodes. There are still two other parameters to be explained: B and M. B represents the maximum number of branches, in other words, node can have the maximum number of child nodes. Obviously, the larger B is, the lower the height of CF tree is. So if B is too small, the height of CF tree is high. And CF tree has a small number of CF. Such a CF tree can usually not satisfy the fundamental demand of clustering. If B is too large, the height of CF is low. A larger number of CFs will be gathered in few nodes. Such a CF tree usually leads to the performance problems of detection. Especially, the process of searching CF will traverse all the CFs in nodes. The real time of algorithm is difficult to guarantee.

B. Merging Algorithm

As shown in the Fig 3, the main thread receives sub CF tree and merge it into main CF tree. This process call the above procedure insert(CF, root).

The pseudocode of merging algorithm is described as follows:

```plaintext
Procedure MainThreadRun()
    Create an empty CF tree;
    while(receive CF from working thread){
        for(each CF_i){
            Insert(CF_i,root);
            // root of main tree
            if(main tree violate M)
                relax T; rebuild main CF tree;
        }
    }
```

C. Parallel Predicting Algorithm

In the parallel predicting algorithm, multiple parallel predicting threads extracts \( \bar{v} \) from the network traffic and predicts \( \bar{v} \) with the help of profiles. Parallel predicting threads only read the profiles and do not write them. Multiple parallel predicting threads have no impact on CF tree. There is no mutual exclusion problem among predicting threads. Moreover, due to the nature of intrusion detection system, security administrator does not usually train and predict at the same time. As a result, multiple predicting threads and main thread hardly access the main CF tree at the same time. But in order to avoid this pitfall, main thread is protected by read lock and predicting threads are protected by read lock.

The predicting thread can be described as follows:

```plaintext
Procedure PredictThreadRun()
    while(new traffic){
        receive new traffic //thread will sleep until the coming of traffic;
        extracts \( \bar{v} \) from traffic;
        if(PredictACF(\( \bar{v}, root \)) is abnormal){
            alert abnormal;
        }else if(PredictNCF(\( \bar{v}, root \)) is abnormal){
            Alert abnormal;
        }
        Record normal;
    }
```

At first, PredictACF(\( \bar{v}, root \)) is called to determine that \( \bar{v} \) is the labeled attack type or not. If procedure of predictACF(\( \bar{v}, root \)) directly returns abnormal; the detection of \( \bar{v} \) is over. Otherwise, \( \bar{v} \) needs to be further determined by calling PredictNCF(\( \bar{v}, root \)). The details of procedure PredictACF(\( \bar{v}, root \)) are shown as follows:

```plaintext
Procedure PredictACF(\( \bar{v}, p \))
    traverse all items of node p, and find the nearest CF_i of \( \bar{v} \);
    if( p points to leaf node)
        if( the distance between \( \bar{v} \) and CF_i < K*R(CF_i) ) //k is a constant slightly greater than 1
            return labeled attack type of CF_i
        else
```
return normal;
PredictNCF(\vec{V}, \text{root});
}/ else
  return PredictACF(\vec{V}, pChild);\}

The details of procedure PredictNCF(\vec{V}, \text{root}) are shown as follows:

Procedure PredictNCF(\vec{V}, p)
{
  traverse all items of node p, and find the nearest CFi of \vec{V};
  if( p points to leaf node)
    if( the distance between \vec{V} and CFi > K*R(CFi)) // k is a constant slightly greater than 1
      return abnormal;
    else
      return normal;
  } else
    return PredictNCF(\vec{V}, pChild);\}

The number of predicting threads can be dynamically adjusted according to the real time network traffic.

IV. EXPERIMENTAL RESULT AND ANALYSIS

A. Dataset

KDD CUP 1999 data set which is deprived from 1998 DARPA Intrusion Detection Evaluation program held by MIT Lincoln Labs, is employed to study the utilization of machine learning for intrusion detection by numerous researchers. The dataset includes all kinds of simulated intrusion actions in the complicated network environment, where each connection instance contains 41 features. In this paper the KDD CUP 1999 data set have been selected as the simulated traffic source of our experiments. 100,000 connection instances, as the training dataset, are extracted randomly from the file kddcup.data_10_percent. 100,000 connection instances, as the predicting dataset, are extracted randomly from the file corrected.

B. Optimal Parameter

There is essentially no difference between serial execution result and parallel execution result. So the optimal parameters of main CF tree are selected by referring to the optimal parameters of serial procedure. As is shown in the Fig 4 and Fig 5, only when B=20 and M=300, detection rate is relatively high and false positive rate is relatively low in the tolerable range.

Threshold TM of main CF tree can relax itself when the number of clusters is great than M. Due to relaxing T, some clusters are merged and the number of leaf nodes decreases. The initial value of TM of main CF tree is set to 0, and TM will reach an adaptive value with the growth of CFs. The final TM is equal to 0.8.

Every CF tree has its parameters. But sub CF tree has no direct impact on detection performance. So there is no need to select the optimal parameters for sub CF tree. The parameters of sub CF tree still need to take some basic precautions: (1) B of sub CF tree should be same with B of main CF tree. It helps to be convenient to insert sub CF tree into main CF tree; (2) M of sub CF tree should be less than M of main CF tree, which can reduce times of splitting node from bottom to up; (3) TS of sub CF tree should be a constant which is less than TS of main CF tree. Therefore the parameters of sub CF tree are set as follows: B=15, M=200 and TS=0.5.

The maximum number of concurrent threads in thread pool has various default values according to different CPU. This value can be modified according to the actual need. The maximum number of concurrent threads in thread pool should be increased with the growth of real time network traffic. But too many threads in thread pool
will also lead to the incremental communication overhead between threads and the incremental concurrency control cost. So the optimal number of threads in thread pool should be selected.

As is shown in the Fig 6, parallel predicting procedure which runs in the Dual Core reaches the maximum speedup when the maximum number of threads is equal to 30 and the parallel predicting procedure which runs in the Quad Core reaches the maximum speedup when the maximum number of threads is equal to 50.

C. Performance Comparison

There are three indexes for measuring performance: Speedup, detection rate and false positive rate.

Speed up is the ratio of running time $T_{sp}$ of serial procedure to running time $T_{pp}$ of parallel procedure when the core number of processor is $n$:

$$S_n = \frac{T_s}{T_p}$$

Different sizes of network traffic are simulated by different number of connection instances. The basic executable unit of procedure is a connection instance. The serial procedure can only process one connection instance at a time, and the parallel procedure can process multiple connection instances. The number of processing connection instances is determined by the number of idle threads in thread pool.

As shown in the Fig 7, with the increasing number of connection instances, the speedup of parallel training algorithm which runs in the Dual Core processor reaches the maximum when the number of connection instances is equal to 60000. Then its speedup decreases slightly. And the speedup of Quad Core has continued to rise. It can be drawn that the parallel training algorithm running in Quad Core processor has the best real time performance in the face of a large amount of network traffic. Moreover, it is noticeable that when the number of connection instances is small, parallel procedure spent more time than the serial procedure.

V. CONCLUSION

With the advantage of multi-core, two busiest processes in the detection: training and predicting are split into many sub tasks and are carried out at the same time instead of one after the other. The parallel processing of training and predicting has the same excellent detection performance with serial processing, and it also has better real time performance than serial processing. Moreover, ACF tree can compensates for the loss of profiles accuracy which is derived from the spherical cluster.

In the future, the message communication mechanism between threads will be replaced by the network communication mechanisms. Parallel threads will run in
different hosts. We will focus on a distributed anomaly detection algorithm.

ACKNOWLEDGMENT

This work was supported in part by the National High Technology Research and Development Program of China under Grant No. 2011AA010101, the National Natural Science Foundation of China under Grant No. 61103197 and 61073009, the Key Programs for Science and Technology Development of Jilin Province of China under Grant No. 2011ZDGG007, the Youth Foundation of Jilin Province of China under Grant No. 201101035.

REFERENCES


Kuo Zhao received the B.E degree in Computer Software in 2001 form Jilin University, followed by M.S degree in Computer Architecture in 2004 and Ph.D. in Computer Software and Theory from the same university in 2008. He is currently associate professor in the College of Computer Science and Technology, Jilin University. His research interests are in operating systems, computer networks and information security. He is the corresponding author of this paper.

Jianfeng Chu (chujf@jlu.edu.cn) is currently an associate professor in the College of Computer Science and Technology, Jilin University. He received his Ph.D. degree from Jilin University. His research interests include network security, mobile security, cloud security and security of Internet of Things.