Data Partitioning Algorithm Based on Capability and Load for Distributed Intrusion Detection - Postprint

Authors: Zhang Runlian, Li Hao, Ye Zhibo, Zhao Xinhong

Date: 2018-05-20T00:00:00+00:00

Abstract

To address the issues of efficiency and detection rate in massive data parallel detection and processing for distributed intrusion detection in high-speed network environments, a capability and load-based data partitioning algorithm is proposed. This algorithm evaluates the data processing capability and load level of each data analysis node based on collected system performance metrics and operational status within the cluster. Based on the node' s capability and load adaptation factor, it determines the weight of each node' s capability to detect and analyze data in the cluster, thereby achieving dynamic data partitioning of massive data among data analysis nodes and allocating data granularity that adapts to their capability and real-time load. Simulation test results demonstrate that the algorithm exhibits favorable load balancing performance, reduces system detection time, and improves both the efficiency of data parallel processing and the detection rate.

Full Text

Preamble

Data Partitioning Algorithm Based on Capacity and Workload in Distributed Intrusion Detection Systems

Zhang Runliana,b, Li Haoa, Ye Zhiboa, Zhao Xinhongc

aGuangxi Key Laboratory of Trusted Software, bGuangxi Key Laboratory of Wireless Broadband Communication & Signal Processing, cGuangxi Key Laboratory of Cryptography & Information Security, Guilin University of Electronic Technology, Guilin, Guangxi 541004, China

Abstract: To address the efficiency and detection rate challenges of massive data parallel processing in high-speed network environments for distributed in-

trusion detection, this paper proposes a data partitioning algorithm based on node capacity and workload. The algorithm evaluates the data processing capacity and workload of each analysis node in the cluster by collecting performance metrics and runtime status. Based on a node' s capacity and load adaptation factor, it balances the weight of each node' s detection and analysis capabilities within the cluster to achieve dynamic data partitioning among analysis nodes, allocating data granularity that matches each node' s capacity and real-time load. Simulation results demonstrate that the algorithm achieves good load balancing, reduces system detection time, and improves the efficiency of parallel data processing and detection rates.

Keywords: distributed intrusion detection; data partitioning; data allocation; load balancing

0 Introduction

Distributed data collection and parallel detection processing are critical capabilities for Distributed Intrusion Detection Systems (DIDS) to handle massive data in high-speed networks. The granularity of parallel data—namely, the size of data subsets allocated to nodes for parallel processing—directly impacts system throughput and load balancing. Data partitioning algorithms divide massive data into different subsets according to specific strategies, and their effectiveness determines the rationality of data allocation, which in turn affects system load balance and overall performance.

Intrusion detection technology has become an essential component of active network security defense systems. To improve detection efficiency, researchers have proposed various optimizations: for rule-based detection methods, reference [1] presents improvements to string matching algorithms to increase system throughput; for anomaly detection methods, reference [2] employs SVM and classification techniques to enhance detection speed. To improve detection rates, reference [3] reduces packet loss in high-speed network environments by accelerating packet capture, filtering, and matching, thereby improving detection accuracy. To enhance system adaptability and scalability, reference [4] utilizes GHSOM neural networks for incremental learning of emerging attacks. For massive data processing, reference [5] adopts data partitioning methods and proposes a data stream sharding mechanism; reference [6] randomly partitions large datasets for distribution to independent neural networks for parallel learning; and reference [7] employs a distributed outlier detection algorithm to balance the workload across computing nodes.

Data partitioning algorithms are widely applied in parallel computing and large-scale data processing to maintain load balance and improve throughput. Reference [8] achieves data stream partitioning and parallel processing through constructed partitioning functions and algorithms. Reference [9] adjusts data assignment to Reducers using a multi-round allocation partition strategy to ad-

dress data skew problems caused by data partitioning in MapReduce. Reference [10] collects and evaluates node performance for data-intensive applications, adjusting the size and quantity of data allocation to improve efficiency.

This paper addresses the challenge of parallel detection and processing of massive data in DIDS by proposing a data partitioning algorithm based on node capacity and workload. By monitoring and collecting performance metrics and real-time operational status of nodes within the cluster, the algorithm evaluates each node' s data processing capacity and workload. It then balances the proportion of each node' s capacity and workload relative to the entire cluster to perform data partitioning and allocation, effectively leveraging node capabilities, maintaining system load balance, and improving system efficiency and detection rates.

---

1 Distributed Intrusion Detection System Architecture

In distributed intrusion detection, system performance is enhanced through parallel data collection by multiple distributed sensor nodes. The collected massive data is then partitioned and distributed to multiple nodes for parallel detection and processing, thereby improving overall data processing capability. This paper employs a capacity and workload-based data partitioning algorithm for data distribution. The system architecture is illustrated in [Figure 1: see original paper], comprising three main components: data collection, control center, and data analysis/detection. The control center includes system monitoring, task scheduler, and alarm response modules.

As shown in [Figure 1: see original paper], the control center' s system monitoring module collects performance metrics and operational status from all data analysis nodes in the cluster, including CPU/memory/disk capacity and utilization rates, providing decision support for data partitioning. The task scheduler performs data partitioning based on the proposed algorithm, evaluating node capacity and workload. The alarm response module processes intrusion detection results from each data analysis node. The data analysis/detection component consists of heterogeneous nodes that detect the collected and allocated data, feeding detection results back to the alarm response module for processing.

---

2 Data Partitioning Algorithm Based on Capacity and Workload

This section presents the proposed algorithm, which dynamically partitions data for nodes based on evaluations of their capacity and workload.

2.1 Node Capacity Calculation

Node capacity is a critical factor in determining the data granularity allocated to each node. Primarily determined by hardware configuration, it can be measured by collecting relevant performance metrics through system monitoring. This paper considers node CPU, memory, and network bandwidth as the main factors affecting data processing capacity. The data processing capacity of node i, denoted as CAP(i), is calculated as follows:

$$CAP(i) = a_1 \times CPU_i + a_2 \times M_i + a_3 \times N_i$$

where a_1, a_2, and a_3 are influence factors for CPU, memory, and network bandwidth on data processing capacity, respectively, with $a_1 + a_2 + a_3 = 1$; CPU_i, M_i, and N_i represent the product of CPU count and MIPS rating, memory size, and network bandwidth of node i, respectively. Since different metrics have different units and value ranges, min-max linear function transformation is applied to normalize the collected metric data before calculation, ensuring convergence to a consistent interval. The CPU_i, M_i, and N_i in the equation are normalized values.

Node heterogeneity makes it difficult to establish a baseline for comparison during data partitioning and complicates control over data granularity size. To address this, we construct a node capacity factor that reflects the relative capacity among nodes, denoted as CL(i) for node i:

$$CL(i) = \frac{CAP(i)}{\min\{CAP(j)|1 \leq j \leq m\}}$$

where m is the number of nodes in the cluster.

2.2 Node Workload Calculation

Node workload is another important factor affecting the data granularity allocated to each node. The workload assessment for data analysis nodes is based on utilization rates of CPU, memory, and network bandwidth collected through system monitoring. The workload of node i, denoted as L(i), is calculated as:

$$L(i) = b_1 \times L(CPU_i) + b_2 \times L(M_i) + b_3 \times L(N_i)$$

where $L(CPU_i)$, $L(M_i)$, and $L(N_i)$ are the utilization rates of CPU, memory, and network bandwidth of node i, respectively; b_1, b_2, and b_3 are influence factors for CPU, memory, and bandwidth utilization on node workload, with $b_1 + b_2 + b_3 = 1$.

2.3 Data Partitioning and Allocation

To ensure rational data partitioning, the algorithm first checks each node' s workload status before allocation to avoid assigning data to overloaded nodes, which would increase latency. To evaluate node workload status, we set a light-load threshold α and an overload threshold β. Based on these thresholds, we construct a load adaptation factor that reflects the impact of node workload on data granularity, denoted as E(i) for node i:

$$E(i) = \begin{cases} 1 - \frac{L(i)}{\alpha} + 1, & L(i) \leq \alpha \\ 1, & \alpha < L(i) < \beta \\ \frac{1}{L(i)}, & L(i) \geq \beta \end{cases}$$

Based on the evaluations of node capacity and workload, we calculate the weight W(i) representing the combined capacity and workload of node i relative to all data analysis nodes in the cluster:

$$W(i) = \frac{CL(i) \times E(i)}{\sum_{n=1}^{n} CL(i) \times E(i)}$$

where n is the number of data analysis nodes in the cluster. In this equation, nodes with stronger capacity and lighter workload receive larger weights.

Finally, let D(i) denote the data subset (i.e., data granularity) partitioned and allocated to node i from the massive dataset M:

$$D(i) = M \times W(i)$$

---

3 Experimental Analysis

3.1 Experimental Environment

We implemented a distributed intrusion detection system model based on Hadoop and developed the capacity and workload-based data partitioning algorithm in Java. The algorithm is integrated into the control center' s task scheduler to perform data partitioning and allocation. The control center serves as the master node, communicating with slave nodes via MapReduce. Slave nodes include system monitoring nodes and data analysis nodes. The system is deployed across a campus network.

All nodes in the Hadoop cluster run Ubuntu 10.04. The master node is configured with an Intel Core 3.3 GHz CPU, 4 GB memory, and 500 GB disk. System monitoring nodes, alarm response nodes, and data storage nodes are each configured with an Intel Pentium 2.8 GHz CPU, 2 GB memory, and 250 GB disk. Data analysis nodes are heterogeneous.

This study focuses on improving system performance through parallel detection of massive data. Therefore, experiments primarily test the rationality of data partitioning, system load balancing after allocation to analysis nodes, and overall system performance improvement. We use the classic DARPA2000 dataset, downloading approximately 3 GB of data stored on the data storage node. To ensure consistent and valid detection results, Snort is installed and configured on all data analysis nodes to detect the data partitioned and allocated by the master node. System monitoring employs the open-source tool Sigar to collect runtime status from each data analysis node. The alarm response node processes intrusion detection alerts.

The algorithm parameters are set as follows: - In Equation (1), based on the impact of CPU, memory, and bandwidth on data processing and experimental results, we set $a_1 = 0.4$, $a_2 = 0.3$, $a_3 = 0.3$. - In Equation (3), $b_1 = 0.4$, $b_2 = 0.3$, $b_3 = 0.3$. - Based on empirical knowledge of light and heavy loads, we set the light-load threshold $\alpha = 0.3$ and overload threshold $\beta = 0.7$. - The data analysis node group consists of 8 heterogeneous nodes.

Experiment 1: Load Balancing Test

Node load imbalance leads to increased response latency and degraded system performance. Using the data partitioning and allocation described above, we tested the load balancing of three algorithms: our proposed algorithm, the method from reference [10], and consistent hashing. shows the distribution of data subsets when partitioning a 3 GB dataset across eight nodes.

** Data Distribution Results Using Different Partitioning Algorithms**

Algorithm	Node Data Distribution (GB)
Consistent Hashing	[Uniform distribution across nodes]
Reference [10] Method	[Performance-based distribution]
Our Algorithm	[Capacity and load-adaptive distribution]

The data shows that consistent hashing distributes data evenly without considering node capacity or workload, resulting in similar data subsets for all nodes. The method from reference [10] calculates each node' s computation time for the dataset and dynamically adjusts allocation accordingly, assigning data volumes that adapt to node performance. Our algorithm further considers real-time load changes in addition to node capacity, producing data subsets that better match node requirements.

After partitioning and allocation using these three algorithms, we compared their load balancing performance across eight nodes using the load balancing metric bLBM(t) [12]. The results are shown in [Figure 2: see original paper].

[Figure 2: see original paper] Load Balancing Comparison of Different Algorithms

[Figure showing bLBM(t) values over time for three algorithms]

[Figure 2: see original paper] demonstrates that consistent hashing exhibits the highest bLBM(t) values due to its lack of load adjustment mechanisms, which leads to uniform data distribution and causes low-performance and heavily-loaded nodes to receive excessive data, resulting in poor load balance. Our algorithm and reference [10] both incorporate load balancing adjustments, achieving more rational data distribution and significantly reducing node detection times. Our algorithm demonstrates slightly better performance with smaller fluctuations, indicating greater system stability.

Experiment 2: Detection Time Test

This experiment tested the time overhead for each node to detect its allocated data subset. We used eight heterogeneous nodes (S1-S8) with a performance ratio of 1:3.4:1.4:1.4:1.3:1:1.3:1.4:1.1. The total system detection times (ST) for each algorithm are shown in .

** Detection Time Test Results**

Algorithm	Total Detection Time (ST)
Consistent Hashing	[Highest value]
Reference [10] Method	[Intermediate value]
Our Algorithm	[Lowest value]

The results show that consistent hashing produces the longest detection time due to its unbalanced data distribution. Both reference [10] and our algorithm achieve load balancing adjustments, resulting in more reasonable data allocation and significantly reduced detection times. Our algorithm achieves slightly lower detection times due to superior load balancing.

Experiment 3: Detection Rate Test

Based on the data partitioning and allocation described above, we tested the detection rate of each node on its allocated data subset. The results indicate that consistent hashing achieves the lowest detection rate of approximately 87.3%, primarily because its packet-to-virtual-node mapping produces small partitioning granularity that severely damages session integrity, thereby reducing detection rates. Neither reference [10] nor our algorithm considers data integrity preservation; however, both achieve similar detection rates of approximately 99% due to fewer partitioning operations causing less damage to data integrity.

4 Conclusion

To address the challenge of massive data parallel detection in distributed intrusion detection systems, this paper proposes a data partitioning algorithm based on node capacity and workload. By evaluating node capabilities and real-time load conditions, the algorithm balances each node' s actual status and data allocation relationships within the cluster, assigning more data to nodes with strong processing capabilities and light workloads. The algorithm performs data partitioning and allocation according to actual node performance and load, fully leveraging node capabilities while preventing performance degradation caused by assigning excessive data to weak or overloaded nodes.

Simulation results demonstrate that the algorithm' s data partitioning and allocation align with node capacities and real-time load states, achieving good load balancing. This effective load balancing reduces system parallel processing time overhead, thereby improving system performance and detection rates. Future work will focus on ensuring data integrity during partitioning to further improve detection rates.

References

[1] Choi Y H, Jung M Y, Seo S W. A fast pattern matching algorithm with multi-byte search unit for high-speed network security [J]. Computer Communications, 2011, 34 (14): 1750-1763.

[2] Erfani S M, Rajasegarar S, Karunasekera S, et al. High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning [J]. Pattern Recognition, 2016, 58: 121-134.

[3] Song H, Lockwood J W. Efficient packet classification for network intrusion detection using FPGA [C]// Proc of the 13th International Symposium on Field-Programmable Gate Arrays. New York: ACM Press, 2005: 238-245.

[4] Yang Yahui, Huang Haizhen, Shen Qingni, et al. Research on intrusion detection based on incremental GHSOM neural network model [J]. Chinese Journal of Computers, 2014, 37 (5): 1216-1224.

[5] Kruegel C, Valeur F, Vigna G, et al. Stateful intrusion detection for high-speed networks [C]// Proc of IEEE Symposium on Security and Privacy. Washington: IEEE Computer Society, 2002: 285-294.

[6] Liu Yanheng, Tian Daxin, Yu Xuegang, et al. Large-scale network intrusion detection algorithm based on distributed learning [J]. Journal of Software, 2008, 19 (4): 993-1003.

[7] Wang Xite, Shen Derong, Bai Mei, et al. BOD: An efficient distributed outlier detection algorithm [J]. Chinese Journal of Computers, 2016, 39 (1): 36-51.

[8] Gedik B. Partitioning functions for stateful data parallelism in stream processing [J]. The VLDB Journal, 2014, 23 (4): 517-539.

[9] Wang Zhuo, Chen Qun, Li Zhanhuai, et al. An incremental partitioning strategy based load balancing method for MapReduce [J]. Chinese Journal of Computers, 2016, 39 (1): 19-35.

[10] Rosas C, Sikora A, Jorba J. Dynamic tuning of the workload partition factor and the resource utilization in data-intensive applications [J]. Future Generation Computer Systems, 2014, 37 (6): 162-177.

[11] Karger D R, Lehman E, Leighton T, et al. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web [C]// Proc of International Symposium on Theory of Computing. 1997: 654-663.

[12] Chen Yijiao, Lu Xicheng, Shi Xiangquan, et al. A session-oriented adaptive load balancing algorithm [J]. Journal of Software, 2008, 19 (7): 1828-1836.

Note: Figure translations are in progress. See original paper for figures.