2014 Sixth International Conference on Intelligent Human-Machine Systems and Cybernetics

A Real-time Anomalies Detection System based on Streaming Technology

Yutan Du, Jun Liu, Fang Liu
Beijing Key Laboratory of Network System Architecture and Convergence
Beijing University of Posts and Telecommunications
Beijing, China
duyutan0814@gmail.com

Luying Chen
Haohandata Technology Co.,Ltd
Beijing, China
cly@kuanguang.com.cn

***Abstract*—With the wide deployment of flow monitoring in IP networks, flow data has been more and more applied on abnormal traffic detection. In practice, anomalies should be detected as fast as possible from giant quantity of flow data, while, at present, some classical anomalies detecting methods can not achieve this goal. In this paper, we propose and implement a distributed streaming computing system which aims to perform real-time anomalies detection by leveraging Apache Storm, a stream-computing platform. Based on this efficient system, we can uninterruptedly monitor the mutation of flow data and locate the source of anomalies or attacks in real-time by finding the specific abnormal IP addresses. A typical application example proved the capability and benefits of our system and we also have a detailed discussion in performance measurements and scalability.**

Keywords-anomalies detection; Apache Storm; streaming computing ; real-time;

I. INTRODUCTION

A. Background and Problem Statement

Flow monitoring is an important field of research and has been widely deployed in IP network [1]. A flow is defined as unidirectional stream of IP packets that share a set of common properties; typically, the IP-five-tuple of protocol, source and destination IP addresses, source and destination ports are used [2]. The flow data contains a lot statistics about those observed flows, such as the number of octets and packets measured within a given time interval.

It has been shown that flow data can be used in anomalies detection. By discovering periodical changes and temporal trends of octets or packets count, it's obvious that we can detect and analyze traffic anomalies caused by network problems or illegitimate attack traffic [3]. In practice, it is important that the examination of the flow data occurs in a timely manner, enabling the network administrator find anomalies and to decide on appropriate countermeasures [4]. While in recent years, there occurs an explosive growth of interactive information in large scale network [5] [6], to deal with giant quantity of these flow data, some batch processing framework with big throughput, such as MapReduce [7] may be a feasible solution, but with poor instantaneity [8]. Those classical anomalies detection method meet a hardship when making real-time process and simultaneously dealing with giant quantity of source data. For this purpose, some emerging kinds of framework which providing real-time analysis and continuous computing can just meet our requirements. Typical one of them is Apache Storm [9]. Storm provides a distributed and fault-tolerant real-time computation method, makes it easy to process unbounded streams of data.

There are three key points that Storm can be used for real time anomalies detection in large-scale network:

- Distributed programming provides parallel processes, services mass data requirement and makes sure to be capable with increasing amount of source flow data.
- Continuous computing model could processes input streams in arbitrarily complex ways. This provides the feasibility of realizing anomalies detection algorithm in the framework. Flow data in streams can be disposed with filters, functions, aggregation and accesses complying with algorithm manner.
- Transfer mode makes these flow data streams storage in memory throughout their lives， and will not be written into disk unless artificial operation. Low latency guarantees real-time process possible.

B. Major Contributions

In this paper, we designed and implemented a scheme that makes the distributed real-time streaming technology applied on anomalies detection. Our system makes a flexible integration of Storm and distributed log collection system Flume [10], working as collector for flow data with TB level amount. Subsequent processing module chain provides a framework for user defined detection algorithm. It can dispose the received data with high-performance distributed, stream-oriented analyses to monitor the minutely changes and overall trends of octets or packets count. Based on some peculiar mutations, we can locate network anomalies by finding the specific abnormal IP addresses in real-time.

C. Paper Organization

This paper is organized as follows: Section II presents the architecture and design of the proposed system. Section III explains our key technologies. Evaluation results, including correctness verification and performance analysis will be shown in section IV. At last, section V presents the concluding remarks and a number of future enhancements.

II. SYSTEM ARCHETECTURE

A. Data Collection

Flow data is a sequence of packets that have the uniform 5-tuple during a certain period. Those flow data are sorted in

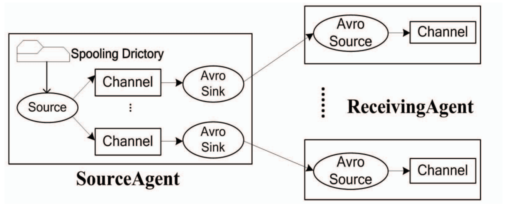

Figure 1. Source Collection Architecture

real time, and preserved in text form. Output file will be created each time unit and store in a directory. To collect the constantly produced flow data, availability and reliability should be met. In Fig. 1, a Flume agent called *SourceAgent* works to monitor the specific local directory. It does the scanner job and uses Avro protocol transferring flow records to those *ReceivingAgent* built in clusters. Flume provides automatic blocking mechanism [11], when records come so fast that the cluster calculating ability reach the limit, *SourceAgent* will slow down the scan speed accordingly.

B. Data Processing

Our system employs Storm stream-processing frame. When receiving the source flow records, *spout* will pre-process them and get required fields including: IP addresses, record time, uplink packets count and downlink packets count. (We choose each flow record's packets count rather than octets to detect the specific anomalies which are caused by a large number of short messages). Then multistage *bolts* cooperate with each other to implement our anomalies detecting algorithm.

Workload is partitioned by four modules. Fig. 2 illustrates our topology, flow data which have been pre-processed by *WorkSport* transfer into *Monitor* module, *TopN* module and *Storage* module synchronously.

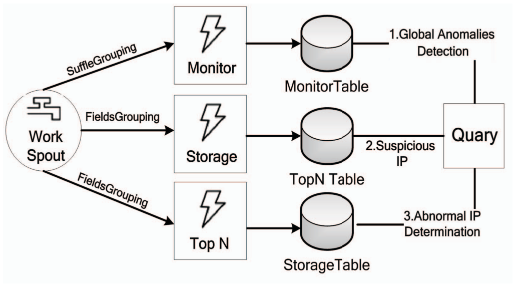

Figure 2. Data Processing Architecture

TABLE I. HBASE MONITOR TABLE STRUCTURE

Row key	Column: Qualifier (CF: count)
each time unit (eg.1392301200)	0: no abnormal 1: uplink abnormal 2: downlink abnormal 3: both way abnormal

TABLE II. HBASE TOP N TABLE STRUCTURE

Row key	Column: CF	
	Qualifier: Up	Qualifier: Down
each time unit (eg.1392301200)	N IP addresses: $IP_1,IP_2,...IP_{n-1},IP_n$	N IP addresses: $IP_1,IP_2,...IP_{n-1},IP_n$

TABLE III. HBASE STORAGE TABLE STRUCTURE

Row key	Column: CF	
	Qualifier: Up	Qualifier: Down
IP+","+each time unit (eg.123.456.78.9,1392301200)	up PKTs count M	down PKTs count N

Monitor module keeps whole packets count's variation trend throughout our system, it judges whether there occurred anomalies in each time unit by using the "K-Nearest Neighbor" (k-NN for short) [12] cumulative variation algorithm. TABLE I shows its stored format in "*Monitor table*".

TopN module finds out N IP addresses which generate the most quantity packets in each time unit. Those IP addresses made the greatest contribution to the overall packets count, so we mark them as *Suspects*. If an anomaly occurs, abnormal IP addresses should come from those *Suspects* in the corresponding time unit. TABLE II shows its stored format in "*TopN table*".

Storage module gets packets count of each IP address every time unit. We store the result in "*Storage table*" with the following format TABLE III shows. This information is used to determine the real abnormal IP addresses. After confirming the range of *Suspects*, we also need to sift. By inquiring those *Suspects* one by one, we can eventually know the "criminal" addresses responsible for anomalies.

Monitoring these three HBase [13] [14] tables in real-time, *query* module can get the final correspondence between anomalies occurring time and those abnormal IP addresses. Query process works uninterruptedly with two steps. Step 1, it checks *Monitor table* each time unit, if detecting anomalies at time unit T, it will immediately find the *Suspects* in TopN table using rowkey "T", and make a combination between T and each *Suspect* of query result as *Storage table* rowkey format "IP,time". Step 2, we scan the *Storage table* using step 1's outcome with a time range from T-x moment to T moment, fetch the packets count trend of this period and use "k-NN" algorithm to make sure if every *Suspect* IP address had problem in time unit T.

III. KEY TECHNOLOGIES

A. Integration of continuous input streams

Due to Storm's nontrivial character, topology must communicate with external systems for input, e.g. our topology *spout* takes flow data information from a external Avro connection provided by the Flume agents. So we need to build Flume agents in Storm *spout* components and make sure that they conclude an infallible and flexible connection. Storm cluster constituted by multiple nodes, for even distribution of whole data, we give equal *spout* tasks to each node by using pluggable schedule and it means every node needs to set as many Flume agents as *spout* task.

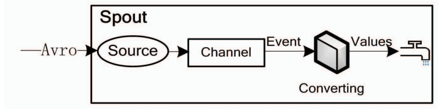

Figure 3. Spout Architecture

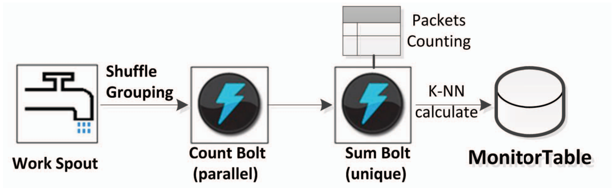

Figure 4. Topology of Monitor Module

To make a one-to-one correspondence between a Flume agent and a *spout* task, we redesign Storm spout's operating logic. In Fig. 3, when topology initializing, every *spout* task firstly establishes an Avro Flume agent binding with a unique port. After receiving flow records, local agent stores them in a memory channel. *Spout* monitors the channel continually, gets flow records and makes them a conversion from Flume's data model "Event" to Storm's tuple structure "Values". After this conversion, records will be sent to subsequent *bolt* calculation components in a batch pattern.

B. Real-time anomalies detection algorithm

For real-time processing, it's hard to store huge amount of experience values, so we choose the "k-NN" algorithm [11]. Fig. 4 illustrates *Monitor* module topology. It has two stages including count *bolt* and sum *bolt* (MC and MS for short).

MC could have a high level parallelism to deal with big throughput. It receives flow records emitted from *spout* using shuffle grouping to balance the pressure. Each task continuously updating the sum of receiving flow records' packets count, and emits them to MS. Emitting the update result when each record comes can make MS hold the accurate status, while cluster inner network load rising at the same time. MS summarizes MC's intermediate outcomes and encapsulates the k-NN algorithm. We maintain a table which keeps the received packets counting of each MC task. Each time unit, MS makes a summation of the whole packets count, then using k-NN algorithm to judge if anomalies occurred and store normal result for future verdict. HBase writing action occurs once per time unit, and MS's table should be cleared after writing operation.

C. Real-time TopN computing algorithm

We use the "Frequent algorithm" [15] which applies for top N query in data streams to dynamically get n IP addresses which produce most packets in each time unit.

Frequent algorithm is based on counting operations. For the given value N, we build N counter, when reading a new data x, it has three kinds of processing mode.

- If a counter maps to x, make this counter value added by 1.
- If no counter maps to x, but there is at least one unoccupied counter, then allot a counter for x and set its value as 1.
- If no counter maps to x, neither is one counter unoccupied, then make all counters' value subtracted by 1 and release counters with a value as 0.

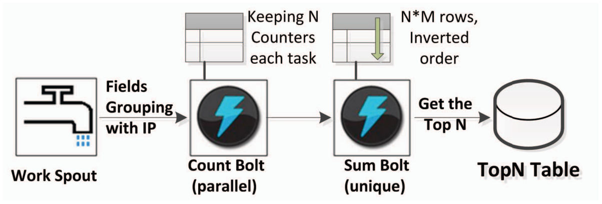

Figure 5. Topology of Monitor Module

Fig. 5 illustrates *TopN* module topology. It also has two stages including count *bolt* and sum *bolt* (TC and TS for short). TC receives flow records emitted from spout using fields grouping method to make sure that flow records with the same IP address can't be transfer to different tasks. Each TC task builds n counters, process coming records using the algorithm mentioned above. TC task provides its n counters to TS and TS with unique task maintains m*n (m is TC tasks count) counters. These counters are sorted in descending order and ranking result will be updated with every coming message emitted by TC. Though TC's counting may be less than the truth, the TS ranking correctness can be ensure when m*n is big enough. IP addresses map to the top n counters are just our target. HBase writing action occurs once per time unit, and all counters should be cleared after writing operation.

IV. Experimental Results

A. Experimental Environment

Our experimental setup consists of 3 computer nodes (N1, N2 and N3 for short). They have the same hardware configuration with double Intel(R) Pentium(R) CPU G2030 @ 3.00GHz, 8GB of RAM, 500GB disk and 100Mbps network bandwidth. Fig. 6 shows the deployment scenarios.

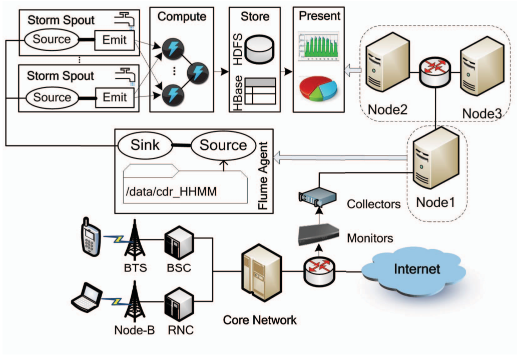

Figure 6. Experimental deployments

TABLE IV. Accuracy in Top N module

Time/ Accuracy	Top 10	Top 25	Top 50	Top 100
22:05	100%	100%	100%	96%
22:15	100%	100%	100%	100%
22:25	100%	100%	100%	97%
22:35	100%	100%	100%	96%
22:45	100%	100%	100%	98%
22:55	100%	100%	100%	98%

B. Correctness Evaluation

In this section, we present some results to analyze our system's function correctness. Experimental data is one hour (22:00-23:00) exported flow data in L province, China.

1) Real-time TopN Computing

We made a comparison between "*TopN table*" records and the authentic set for N IP addresses with most packets count. Accuracy is the metric and it is computed using

$$Accuracy = \frac{M}{N} \qquad (1)$$

M is the intersection of "*TopN table*" and authentic top N IP set. It is obviously that high accuracy presents good validity.

We pick six typical points to show the result. Accuracies are listed in TABLE IV. We just concentrate on coverage probabilities, while their ranking is not what we are concerned about. The accuracy was maintaining in 96 – 100%. This indicates that our real-time TopN computing scheme in Storm can achieve high accuracy.

2) Real-time anomalies detection

The left y-axis of Fig. 7 is packets count value which is calculated by all the raw flow records, while right y-axis is monitor module's detection result. In Fig. 9, up packets count rose unexpectedly at 22:20, while Monitor module's value changed to "1", meaning uplink anomalies occurred at the same time. Continue to observe, it's obvious that real value of packets count and our detection result showed great identity throughout the whole experimental time. Monitor table's value can reflect the truth in real-time, so we can conclude that our anomalies detection method that makes k-NN algorithm embedded in Storm topology works well.

3) Judging Correctness Evaluation

Up to this point, we have shown that the first found anomaly occurred at 22:20. At 22:20, our system exported 28 abnormal inside IP addresses records in upstream direction. By analyzing with raw flow records， we found that at 22:20, 26 members of these 28 IP addresses had huge flow records with the same outside IP 120.200.96.7. And just at that time unit, 120.200.96.7's packets count went up from 2538 to 17063945 which accounting for even 89 percent. According to some further research, we found the reason for this exceedingly proportion rising is that 120.200.96.7 received a huge number of consecutive, short UDP packets that transferred by 45 inside IP addresses around 22:20， while the top 26 with the most packets count among these 45 inside IP addresses were just these 26 which our system regarded as anomalies. Our system successfully detected this real existing abnormal UDP flow [16] by figuring out senders' IP. For the left two abnormal records, their inside or outside packets count rose drastically between 22:19 and 22:20, and this rising made themselves regarded as anomalies. However, we omit their detailed schema from this paper.

C. Performance and Scalability analyze

1) Performance Analyses

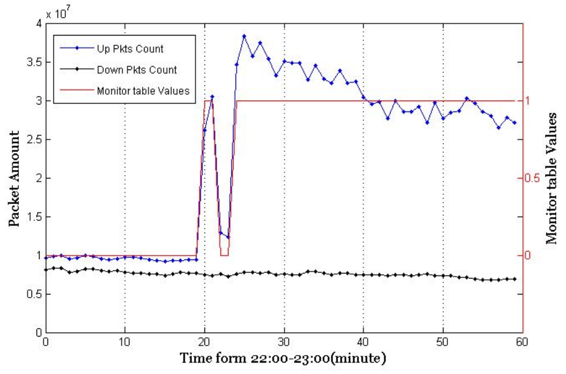

Figure 7. Comparison between Monitor Module's Result and Real Packet

For a distributed streaming system, many factors can have an effect on performance. We monitored our system status with memory, disk IO, networks IO and CPU.

Storm topology has no disk IO unless we manually take the operation reading or writing disk， so our system disk IO are HBase writing and reading. In TABLE V, disk IO is under 20% while the system attains the maximum processing ability. So disk IO is not a key factor, neither is network IO with a transmission rate slower than 3Mb/s, because our network bandwidth is 12.5Mb/s.

Now we concentrate on CPU performance, in TABLE V, when adding input records count, nodes' CPU idle lowered. The y-axis of Fig. 8 is CPU idle percentage, when the records count grows up to 0.8 million every minute, CPU had just a few seconds around 51 without full load status. When this became to 0.85 million， CPU has no idle and attained to ultimate state ， while at the same time accumulation of unprocessed flow data occurred at source dictionary because of the Flume blocking mechanism. This shows that CPU confined system maximum calculate ability.

Memory is also an important resource. By making a comparison between the third and fourth column, we can see when the RAM increasing, system performance had no obviously improvement except that frequency of JVM garbage collection decreased. CPU also stayed in ultimate state and the real processing time did not shorten. So in our experiment, memory did not become the bottleneck of performance.

TABLE V. SYSTEM PERFORMANCE ANALYSES

Records count per minute (*100000)	4 (1G RAM)	7 (1G RAM)	8 (1G RAM)	8 (4G RAM)
Java GC in 20 minutes (young/full)	N2(302/26) N3(291/19)	N2(365/30) N3(277/30)	N2(485/32) N3(441/33)	N2(102/14) N3(93/17)
Average Disk IO	9.5%	16.7%	19.9%	19.7%
Source node Network IO	2.2Mb/s	2.56Mb/s	2.39Mb/s	2.42Mb/s
Average CPU Idle	49.93%	19.76%	7.17%	7.42%
Idle> 50% per minute	28s	12s	2s	2s

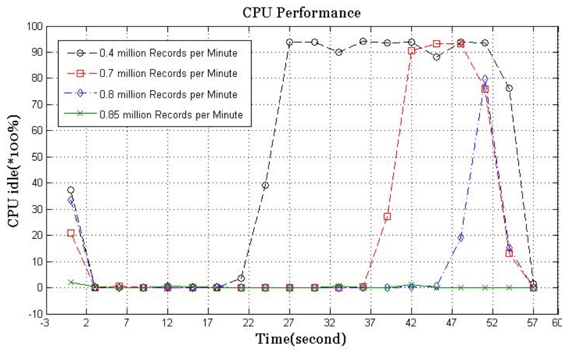

Figure 8. CPU Free Period Situation

TABLE VI. SCALABILITY OF TASK PARALLELISM

Num of Nodes	Maximum processing records (*10000 per min)	Maximum processing records Per Node (*10000 per min)
2	80	40
3	118	39.3
4	152	38
5	182	36.4

According to those result, we can conclude that CPU properties dominate performance. Capability promoting needs support of CPU with higher frequency or more cores.

2) *Scalability Evaluation*

Scalability is the key parameter that is used to evaluate the efficiency of a parallel algorithm. To evaluate scalability of our system, we changed the number of cluster nodes as 2, 3, 4, and 5 to study the processing capacity for detecting work. The middle column of TABLE VI is the maximum processing ability calculated when CPU cores reach their ultimate state. The ideal scalability should mathematically be a linear growth that means doubling the number of computing nodes doubles the processing capacity. The right column shows node average processing ability, it remain the same value indicates the ideal scalability. From the result, we can see that our method approaches closely to linear scalability. The reasons for not matching the ideal is that extra task scheduling, and communications between *spout* and *bolt* take up more system resources, which are internal mechanism residing in Storm execution environment.

Although our parallel algorithm extend little slower than the linear scalability, the total execution amount of flow data records, which are 1.82 million for 1 minute can satisfy the performance requirement as a stream-oriented processing program running over only 5 working nodes.

V. CONCLUTION

This paper proposes a real-time anomalies detection system based on streaming technology. We designed and implemented a novel anomalies detection algorithm on the streaming computation freamwork. It offered a solution that can detects network anomalies and accurately locate the source of anomalies at IP level. Experimental results prove that the proposed system achieves accurate real-time anomalies detection from mass flow data in a scalable way. For future work, we will focus on enhancing the system to provide multiple data source processing capability.

ACKNOWLEDGMENT

This work was supported by 111 Project of China (B08004) and the Fundamental Research Funds for the Central Universities (2013RC0114).

REFERENCES

[1] F. Dressler and G. Munz. Flexible flow aggregation for adaptive network monitoring. IEEE LCN Workshop on Network Measurements 2006, Tampa, Florida, USA, Nov. 2006.

[2] Gerhard Munz and Georg Carle. Real-time Analysis of Flow Data for Network Attack Detection. Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on, 21-25 May 2007, Munich, Germany:100-108.

[3] Anukool Lakhina, Mark Crovella, Christiphe Diot. Characterization of network-wide anomalies in traffic flows. Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, 25 Oct 2004:201-206.

[4] Paul Barford, David Plonka. Characteristics of network traffic flow anomalies. IMW '01 Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement, 01 Nov 2001: 69-73.

[5] J. McHugh. Sets, Bags, and Rock and Roll: Analyzing Large Data Sets of Network Data. European Symposium on Research in Computer Security 2004 (ESORICS 04), Sophia Antipolis, France, Sept. 2004.

[6] Ling Huang, XuanLong Nguyen et al. Communication-Efficient Online Detection of Network-Wide Anomalies. 26th IEEE International Conference on Computer Communications. 6-12 May 2007, Anchorage, AK:134-142.

[7] Jeffrey Dean, Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. Communications of the ACM - 50th anniversary issue: 1958 – 2008, Volume 51 Issue 1, January 2008 ,NY, USA:107-113.

[8] Youngseok Lee, Wonchul Kang, Hyeongu Son. An Internet Traffic Analysis Method with MapReduce. Network Operations and Management Symposium Workshops (NOMS Wksps) . 19-23 April 2010:357-361.

[9] Twitter Storm Homepage, http://storm.incubator.apache.org/.

[10] Apache Flume Homepage, http://flume.apache.org/.

[11] Chengwei Wang, Infantdani Abel Rayan, Karsten Schwan. Faster, larger, easier: reining real-time big data processing in cloud. Middleware Conference. 13 Dec. 2012.

[12] Xiaohui Yu, Ken Q. Pu, Nick Koudas. Monitoring k-Nearest Neighbor Queries Over Moving Objects. ICDE 2005, 05-08 April 2005:631-642.

[13] Apache HBase Homepage, http://hbase.apache.org/.

[14] George L. HBase: the definitive guide[M]. O'Reilly Media, Inc. 2011.

[15] Richard M. Karp, Scott Shenker, Christos H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. ACM Transactions on Database Systems, Volume 28 Issue 1, March 2003:51-55.

[16] C. Gates and D. Becknel. Host Anomalies from Network Data. IEEE Systems, Man and Cybernetics Information Assurance Workshop, West Point, NY, June 2005.