The Improvement of Network Performance by Using the Technique of PF-RING Zero-copy to Optimize Spark Streaming

Feng LIU[1], Zhan-rong LI[2,*], Yang GAO[3] and Sheng-si LUO[4]

School of Computer, Electronics and Information in Guangxi University Nanning, China

E-mail: 290509953@qq.com, m13821843477@163.com

*Corresponding author

Keywords: Zero copy, PF_RING, Spark Streaming, Adaptive

Abstract. Regarding the problem of lower utilization rate of broadband and data throughput in the real-time computations when Spark Streaming receiving and treating a large number of TCP packet, we proposed a technique of effective zero-copy of IO kernel and optimiazation of user protocol stack. By comparing and analyzing the monitoring data, we can determine that the reason of low utilization rate of network under special circumstances is the large expanse of kernel system which induced by the big data flow frame dealing with a large number of network IO interruptions in unit time. We propose that the realization of protocol stack can be stripped from the kernel of operating system by combining PF_RING, this can both reduce the expanse of system management and avoid the additional data movement. The results demonstrate that comparing with the present open-source version of Spark-1.4.0, we can effectively reduce the time delay, decrease the jitter and improved the utilization rate of the Worker nodes by using the technique of PF-RING zero-copy to optimize Spark Streaming. And this technique can also meet the real-time computation's need when receiving and treating a large number of TCP packets.

Introduction

Recently, with the progresses of science and technology more than 2EB new data have been produced every day all over the world. The internet companies and scientific institutions have been found the niches which contained above and conducted production or development. Spark is a distributed memory computing systems which based on Hadoop. The Spark system suddenly rises relying on the excellent computing velocity and security. The memory computing framework of Spark has natural advantage in machine learning and data mining. The distributed memory has been extracted to the Resilient Distributed Datasets (RDD) [1] in Spark.

Spark supports the offline data calculation and real-time streaming data analysis. The Spark Streaming system can treat the real-time data stream with a high flux and fault-tolerant manner, and the complex operations (map, reduce, join, window) have been done to a variety of data sources (such as Kdfka, Flume, Twitter, Zero and TCP socket) in this system. The results will be saved in the external file system, database, or applied to practical production.

Due to the sliding window mechanism used in the data flow model and the

timeliness in data stream tuple, (each tuple will have a timestamp), Spark Streaming is suitable for the application of the multiple iterative operation. After the comparing with the effective experimental results, it is found that the cluster network bandwidth usage is only 40 to 85 MB/s in the Spark Streaming system when dealing with the data packets less than 100B in the case of no TCP incast in the network. So the cluster network optimization of Spark Streaming will make the distributed cluster processing improved in the production. At this stage, the conferences about the optimization of Spark which held in China or abroad are put the breakthrough point on the optimization of shuffle algorithm logic. In Ref. [2], the authors proposed a Spark join optimization way basing on the data structure of BloomFilter which focused on the large table equivalent links. The optimized Spark join way can reduce the expense of the network communication, thus reducing Shuffle phase time expense. In the literature [3], the authors put forward an equivalent link allocation algorithm and sharing time slice allocation policy basing on the consistent hash multi-channel data flow. This reduced the time expense in the phase of data reception and Shuffle.

The schemes for optimizing the Spark Streaming network performance are very limited now. Basing on the real-time Streaming data calculation of Spark Streaming, we can reduce the switching and the locality expenses by optimizing the Worker nodes of network card, CPU and thread scheduling under special circumstances. Using the kernel module which provided by PF_RING [5], the network performance of Spark Streaming can be optimized through the optimized techniques of IO zero-copy of kernel and the user protocol stack. In Ref. [4], the authors put forward the optimization of GPU more machine card.

Analysis of Spark Streaming Network Performance

Experimental Methods

By analyzing the high load experiment data of Spark Streaming, determining the utilization of Spark cluster network in the cases of high load, we can find the appropriate experimental parameters.

In order to achieve the test of high load, using the interface which provided by Java NIO, we can realize the operations of zero copy network IO of JVM, this technique greatly improves the performance of the client sends, thus realize the high load test of Spark Streaming.

Analysis of the Problems

By comparing with the experimental data, we found that the expense of kernel system is a little relatively big when receiving a small size packet. So we think that the network performance is closely relative to the size of packets, this is consistent with the test results. When packets are large to a certain size, due to the big memory management and local expense of Spark applying codes, the computing time which treating the data per time will increase, This limits the reception efficiency of Streaming data.

The Streaming batch interval setting is very important, because the batch time intervals directly determines balance between the delay of end-to-end and the

trade-off of the load flow throughput. For the settings of calculating window and sliding time of Streaming, when calculating window augment, the averaged receive bandwidth will decline. The reason for the above phenomena is that when calculating window becomes bigger, the buffer data volume will increase, thereby increasing the Spark computing tasks the data distribution of pressure, this leads to the extension of computed time, and also indirectly affect the amount of data reception and bandwidth.

Overall the network utilization of Spark Streaming is good, and the maximum utilization rate of network bandwidth can reach 80%, but there are some problems in multiple network card physical node queues in Spark cluster corresponding to multicore CPUs. During the usage of Spark Streaming we found that each Spark cluster Worker nodes of CPU is not absolutely balanced, there is 1 to 2 CPU consumption are about 20% larger than the rest of the CPU, Under the condition of high load there is 1 to 2 CPU is fully occupied, this will lead to the phenomenon of the computation delay of the Spark the cluster in the running process.

Irqbalance can be used for optimizing the allocation of interruption, and it will automatically collect system data and used it to analyze the mode. According to the load situation of the system, Irqbalance will make work status in the Performance mode or Power-save mode. In the experimental cluster of Spark, the calculation task is very heavy and no energy demand, but it need to take full advantage of the performance of each CPU. In a lot of packet system, the optimization of irqbalance has almost no effect, but also makes the CPU consumption imbalance, and this will cause the machine performance is not fully utilized.

Preliminary Optimization of the Performance

The Configuration of Spark Cluster Experiment

In Spark cluster, the Worker nodes are I350 queue network card and provide the 8 send queue and 8 receiving queue for the system. The CPU type of each physical node in the cluster of Spark is E3-1240 v3 3.40GHz*8, each CPU has 8 Cores. That is to say in Streaming runtime network traffic load to multiple CPUs. Through configuration data flow and affinity of nic queue, the network adapter queue bound to different CPUs. In theory, this method can improve performance and relieve the pressure of each CPU Cache. Each card queue is associated with a hard break, processing the interrupt response will be equally distributed on different CPUs.

Preliminary Network Optimization Method

Linux provides the Receive Packet Steering (RPS), the Transmit Packet Steering (XPS) and SMP IRQ affinity, and they are used to control the interrupt, nic queue, CPU affinity. Packets to interrupt equilibrium can be divided into hard interrupt load balancing and soft interrupt load balancing. The hard interrupts is suspend for the suspension of a single trigger to different CPUs. The shortcoming of this method is also very obvious, because the packets were randomly assigned to different cpus, leading to a lower packet cache of shooting. Different packet trigger different interrupt load balancing. Soft interrupt load balance is mainly the RPS, RFS, which provided by the system. RPS transfers the packets to the different CPU according to

certain rules before soft interrupt handling. This rule is based on packet load balancing, without considering the characteristics of flow, so there is still a problem of low hit ratio. Distribution of RFS and RPS is basically similar, but when distributing the packets to CPU, RPS is according to the flow distribution, it can ensure that the same stream of packets can be treated by the same CPU. This method can improve the cache.

1. The first optimization method: Close Irqbalance and manual bind the soft interrupt to the special CPU. Considering each machine 8 card queue corresponding 8 CPU in the Spark cluster, we can specify one CPU treat the interrupt request from one card queue. After One-to-one binding between CPU and nic queue, binding soft interrupt and hard interrupt, we can proceed the operation on the different nodes of the cluster in Spark, respectively.

2. The second optimization method: Close Irqbalance and manual binding soft interrupt to the special the CPU. By default the RPS/RFS, XPS function is not open, need to manually open, more than 1 for binding and CPU card queue and binding soft interrupt, respectively in Spark different nodes on the cluster configuration operation.

Analysis for the Preliminary Optimization Data of Spark Streaming

By comparing the test results of the Spark Streaming, it shows that for the packet size of 50B, the averaged bandwidth of network is 42379KB/s and the throughput of Streaming per 2 second is 88054603B in the case of no optimization. After using the optimization method two, the averaged bandwidth of network becomes 50120KB/s and the throughput of Streaming per 2 second becomes 91340778B. The utilization rate of network bandwidth is upgraded about 18.3%, the averaged 2s throughput promotion is about 3.3MB. When the packet size of 100B, the averaged network bandwidth of Streaming is 64084KB/s and the averaged 2s throughput of Streaming is 133989058B in the no optimization case. After using optimization method two, the averaged network bandwidth of Streaming is 69799KB/s, the averaged 2s throughput of Streaming is 138522140B. The utilization rate of network bandwidth is upgraded about 8.9%, the average 2s throughput promotion is about 4.5MB. The above data show that the utilization rate of Spark Streaming network is not increased 50% as expected. The results for Optimized method two are better those of method one. On the whole the increase of network utilization for the Spark Streaming is less than 20%. It can be found during the test that after optimization the occupancies of 8 CPUs on each physical node are relatively averaged comparing with before optimization. In order to ensure the accuracy, the data which used to comparison are all measured in a period and the averaged data are used finally.

The optimization of Spark Streaming by Custom protocol stack

PF_RING custom Protocol Stack

The designed intention of Linux system is convenient to the control of telephone and telegraph, it is not suitable for dealing with the large-scale network packet. The breakthrough point for the improvement of performance is the direct management of network data, memory and CPU scheduling by the applied procedures. In order to

realize the above operation it needs to bypass the Linux kernel stack. The performance of the kernel stacks which developed by Wind River and 6 Wind Gate companies has been increased 500% at least comparing with those of the Linux UDP/TCP.

PF_RING is a patch which developed by Luca Deri and it uses to increase the efficiency of the kernel processing packets and balance the applied procedures. PF_RING is a new type of socket of the network, and it can greatly improve the speed of the packet capture. The basic principle of the PF_RING is to store the receive packets from network card in a circular buffer. There are two interfaces for this cache, and one is responsible for writing the data packets from network card, the other is providing an interface which used to reads and receives data packets by mmap function. PF_RING reduces the number of memory copy, and distributes the received data to different circular buffers. Using the above methods, you can also increase the capture performance of the packet through the cluster.

Spark Streaming with PF_RING building Custom Protocol Stack

Install and operate the PF_RING in Worker nodes of the Spark cluster. Under the current conditions, the data packets can only memcpy to PF_RING, and it not through the standard Linux path. PF_RING allows the users to deal with the custom packets through the plug-ins, because PF_RING writing the data interface by c++ language, and this is not compatible with the Spark. So by connecting the Spark with PF_RING library by the custom RPC script which written by thrift language, the Spark Streaming can make the customized data packets of PF_RING as the data source.

Experiments using the DNA module of PF_RING, the rate of packet capture and transmission can be significantly increased by loading PF_RING module and DNA driver with consumer Libzero mode. , the kernel layer of Linux is bypassed, and Spark obtains the data packets from PF_RING ring buffer directly. The Libzero library provides two main components: DNA Cluster and DNA Bouncer. Specifying the send interface of the data packets, it can realize the collection of the data packets and ensure the data sharing of Streaming in the form of zero-copy.

The comparison and Analysis of the Optimized Data

Comparisons of Network Bandwidth

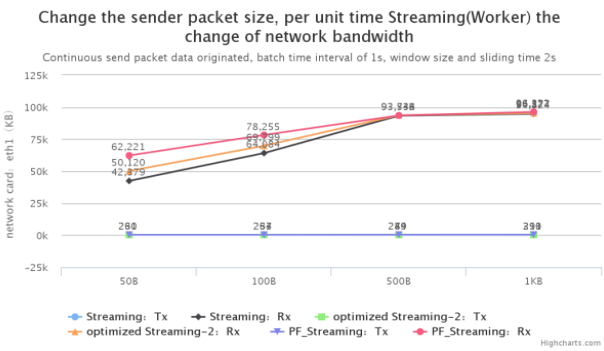

Figure 1. PF_Ring optimized and the default network bandwidth

In Figure 1, the data transmitter send data successively, a single data size is about

50B ~ 1KB, the time interval of batch for Streaming is 1s, the calculation window size and sliding time of Streaming is 2s. Tx represents the averaged values of sending flow by card (unit: KB), Rx represents the averaged values of receiving flow by card (unit: KB), Streaming represent no optimizating, optimized Streaming-optimization 1: using the first optimized method, PF_Streaming: using the PF_RING method.

Comparisons of Throughput

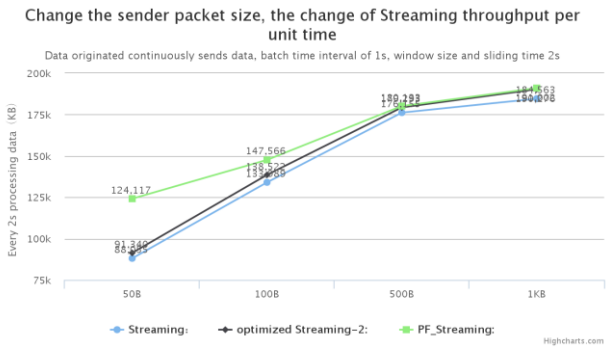

Figure 2. PF_Ring optimized and by default throughput

In Figure 2, the data transmitter send data successively, a single data size is about 50B ~ 1KB, the time interval of batch for Streaming is 1s, the calculation window size and sliding time of Streaming is 2s, the averaged processing data volume every 2s (unit: KB) and Streaming: no optimization, Streaming-optimiztion 1: using the first optimized method, PF_Streaming: using the PF_RING method to optimize.

Analysis and Conclusions

After receiving the data packets from card, the drive of PF_RING transmit them to the pre-allocated memory by the Libzero user mode. At the same time, updating the pointer of the data packets in the lock ring queue, the polling custom script can apperceive the received data packets and send them to the applied procedures of Spark Streaming as the data source. If treating the processes of receiving packets according to the worker nodes before optimization, the card notify the protocol stack by the interrupt mode to deal with the data packets. The protocol stack will first check the validity of the data packets, and then judge if the target of data packets is the socket of this machine. If the above conditions are satisfied, the data packets will be copied and transferred to the user socket. The path of the above process is very fussy and there is a copy process from kernel to the applied layer.

By comparing with the experimental data, the PF_RING custom stack can improve and optimize the network performance of Spark Streaming. This method is effective not only for the Spark Streaming, but also for the architectures which dealing with the vast big data. But the PF_RING also has shortages. Now most cards support the technique of MSI_X, and the technique of RSS realize the load balancing between each RX queue. It can fully utilize the multi-core advantage of the CPU to increase the velocity of treating the data packets. But the PF_RING can not make full use of advantage, because the custom driver still need a polling network card queue and it can not access to all card queue at the same time.

Conclusions

The Combination of Spark Streaming with PF_RING make the entire protocol stack separate from the kernel of operating system. The above operation can not only reduce the overhead of system management, but also avoid the additional data movement. At the same time, the PF_RING library can directly operate with the card data. From the analysis of the published investigations, the custom protocol stacks can significantly increase the throughput of a simple application, but the improvement this performance are based on a set of software and hardware optimization solution.

This paper discusses the effect of several potential network optimization technologies on the promotion of the performance of network. By the tests and optimizations for the typically distributed memory computing platform Spark and a distributed Streaming platform Spark Streaming, we found that the simple network optimization has certain limitations to improve overall application performance. Specific performance has the following three aspects: Firstly, the dependence of CPU intensive application on the network performance is low. Secondly, facing the throughput of network applications, the dependence on delay performance is low. Finally, the underlying network optimization is difficult and the compatibility is poor.

References

[1] Di JiaAn, deep understanding of the Spark core idea and source code analysis[M]. Mechanical industry press, 2015.

[2] Optimiziting Big Data Equi-join in Spark and Its Application in Analysis of Network Traffic data[D], South China University of Technology, 2015.

[3] Research on the Key Technologies of Large-Scale Real-Time Data Streams Joins[D], Tsinghua university, 2015.

[4] Yan ZhiJie. GPU more machine and more card Machine Learning Middleware[J]. BDTC, 2015.

[5] PF_RING,http://www.ntop.org/products/packet-capture/pf_ring/