International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 9, September- 2014

Parallelizing TCP/IP Offline Log Analysis and Processing Exploiting Multiprocessor Functionality

Chirag Kharwar
Department Of Computer Science & Engineering
Nirma university

***Abstract* —In the era of internet important informationtravel all over the world to your computer. There are so many applications of internet which are used by various utilities and programs in the operating systems. All of these packets travel through the computer from Network Interface Card (NIC). Network analysts capture these packets for offline analysis and then finally take decision based on the observations they have made. For this, the information of packet header and summary of the entire capture is important. This project is for the minimization of the time taken for the whole offline files to get processed. Here multiple-cores in the machine are used and single algorithm for the analysis is applied making it a SIMD parallelization. For this purpose first the entire capture is partitioned and then the partitions are given to different cores for processing and then final result is returned back to the main core which is running the main process. Experiments results are also included which shows that parallelization works better than normal implementations.**

Keywords—Packet Capture; Parallel Processing; Partitioning

I. INTRODUCTION

Internet is a big network where lots of new messages travels from one end of the world to the other end. Some of the messages are useful to us and others aren't. Due to large amount of messages passing from our NIC there may be potential problems that can arise like network slowdown, low throughput, high packet loss etc. Thus there should be some mechanism by which we can analyze so called headers of the protocols and conclude upon the cause of these problems. The application can be both in research as well as in security to analyze some of the threats in the network. For this purpose, packet sniffing is done. There are many packet sniffers available in the network world namely Wireshark, Omnipeek, TcpDump etc [4] [8] [9]. All are feature rich and provides easy to use interface for the users.

Processing time to create the view of all the packets which are there in the captured file increases as the size of capture increases. Thus, there is need to improve in this area of the sniffing i.e. offline analysis of the packets. State of the art technologies in the field of VLSI is improving day by day. Thus, even general purpose PCs are having multiple cores in them, but they can only be utilized if the applications are made to use all of them simultaneously. Thus, the need is that the offline analysis of the packet should be done in parallel utilizing all the cores available in the system so that the user can get results fast and make decisions based on them. The analysis process is having many applications in security also [2].

The rest of this paper is organized as follows: in section II I have discussed the existing sequential method by which the packets are sniffed and then stored for the offline analysis, in section III I have discussed some of the basic concepts of the parallelization on multicore processors so that reader can get idea about the method, in section IV I have shown my approach of doing the analysis in parallel, in section V I have compared the results of parallel program and sequential program, in section VI I will discuss the work that can be done for further improvement in the performance, finally in section VII I will be concluding my paper.

II. PACKET SNIFFING AND OFFLINE ANALYSIS

Packet sniffing process involves capturing the packets with the help of native or plugin libraries and then stores the content of the packet so that user can analyze it whenever needed [1]. This analysis involves so many filters which are used by the network administrator. Each time when some filter is applied the offline file is again traversed which takes lots of time if the size of the file is large.

The packets traverse from the network to the user's network and ultimately to the NIC of the user's PC. When packets reaches the local network, the router broadcasts all the packets that it receives to all the hosts in the local network. The hosts in the network checks whether the packet is intended for it, if it is the case then it takes it, otherwise it discards the packet. In my case program runs in promiscuous mode in which host will not discard any packet instead it will accept every packet.

Now these packets travel from the NIC to the CPU. Some drivers/libraries are needed to capture those from the NIC. For that libpcap (for Linux) and winpcap (for windows) are available. Using these libraries the packets are captured and then stored on to the disk for the later analysis. The packets are stored in the ".pcap" format which is a standard format. After storing the packets in the offline file the application can fetch it using the APIs available in the library, and finally it gets displayed to the user. Based on that user (here network analyst) can take decisions. Thus, the need is to process these packets as quickly as possible so that user can have more time

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 9, September- 2014

to take decision rather than waiting for the offline captured file to get processed completely. For this reason multi-core packet parallelization is applied [5] [6].

III. MULTIPROCESSING IN MULTICORE SYSTEMS

In a multiprocessing system, we divide the work into multiple number of sub modules or sub tasks which can be independently executed on different cores of the system. These tasks utilizes different cores of the machine as a result of that the program can achieve better performance in terms of execution speed. [7] Defines multi-processing as "A single OS instance controls two or more identical processors connected to a single shared memory and distribute the task among them." In case of the parallelization methods some of the standard methods like threshold method in which fix number determines the work load on each core. In one of the method the number can be changed dynamically which further increase the efficiency. Both the methods are having their application specific pros and cons. My motto is not to discuss them rather use it in my work. In the next section some of these logics are used to make the parallelization work on the captured files.

IV. APPROACH OF PARALLELIZATION

My approach for the parallelization is mainly divided in to two parts which are as follows:

A. Partitioning Captured Packets

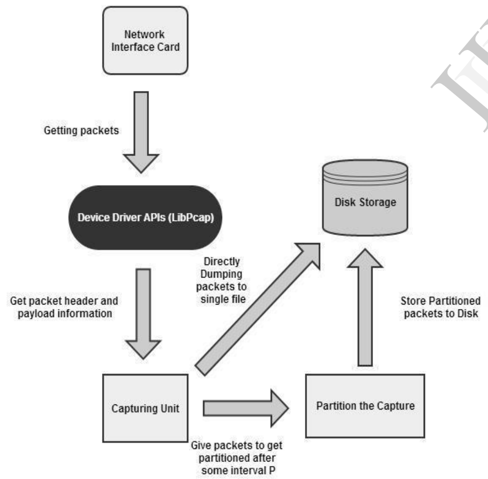

FIGURE 1. PARTITIONING PACKETS AND STORING TO DISK

As shown in figure 1 from the NIC the packets are captured using the standard libpcap or winpcap library and then it is send to the capturing unit where it is divided into parts, then files for all the individual parts are stored at some location on the disk. Here the partitions are made based on number of packets to be put in one partition. This variable is configurable by the user i.e. user can specify after how many packets new partition will be created, user can also specify size. Packets are then kept in the disk for the next stage of the application.

B. Parallel Analysis of Partitioned Files

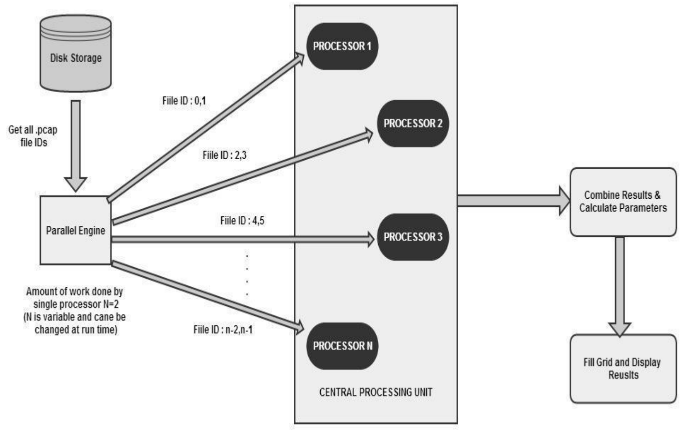

FIGURE 2. RETRIEVAL ASSUMING THRESHOLD VALUE TO 2

As shown in the figure 2 the parallel engine will take the files from the disk which are those files that are partitioned in the previous step. Now these files are individual, hence, similar processing can be applied on them in parallel. Basically, say if there are 4 cores in the system, the application forks 4 processes. As they are forked they are having different address space, in this space the program is copied and then for each process some fix/dynamic numbers of files are assigned. Here one assumption is that the processors are dedicated to do this task. Thus, as soon as the files are ready they will get the processor and hence they can run without much delay. This can be done by increasing the priority of the master process which controls all these activities. I will not discuss about how to increase the priority of process as it is implementation dependent/programming language specific.

The parallelization will work as follows:

I have used the threshold method which will decide the load on each core. For simplicity let us take threshold value of 2 i.e. at a time two files will be given to each core for processing.

As shown in the above figure first CORE 1 will process partition numbers from 1 to 2, CORE 2 will process partition numbers from 3 to 4 up to CORE N that will be processing partition numbers from n-1 to n. In this way all the cores of the CPU will be utilized and better performance can be achieved in terms of speed. The sequential numbers are taken for the simplicity, so that reader can easily get the concept. In terms of real implementation the partition given to each core is out of the control of the application. It is decided by the operating systems however user can control the number of partition which is given to the core using the threshold value.

V. Implementation Results

For implementing the above mentioned concept i have used java as programming language and Fork/Join which is a parallelization framework included since java 7. Netbeans has been used as an editor. For capturing the packets from the NIC JnetPcap library is used which is an open source library. This library internally uses winpcap (for windows) and libpcap (for linux) libraries which are generally used for capturing the packet information from the NIC. The program will run in promiscuous mode to capture all the network activities which is occurring in the network where host is located. As i have discussed earlier, for this experiment the threshold value that i have chosen is 10000 packets. Thus after 10000 packets new file is generated and is stored in the disk for the later processing. This value can either be modified by the user or can be chosen dynamically by the system.

TABLE I. WORKSTATION CONFIGURATION

Parameter	Value
Processor	Intel Core i5 4200U - 1.6 GHz - Dual core (with threading & Virtualization Enabled)
Processor Architecture	X64
RAM	4 GB
Number of Cores	4 (logical)
Operating System	Windows 8.1 Professional

Below table shows the experiment results:

TABLE II. EXPERIMENT RESULTS

Number of Packets	Sequential (sec)	Parallel(sec)
40000	0.587	1.138
80000	1.265	1.347
100000	3.424	1.863
200000	7.02	2.86
300000	15.334	4.206
350000	18.661	6.543
400000	20.248	8.675
500000	26.426	11.366

The actual performance of the system depends on the hardware capability of the workstation. Here sequential and parallel are tested on same hardware so that they can be compared.

The results can be better analyzed from the graph shown below:

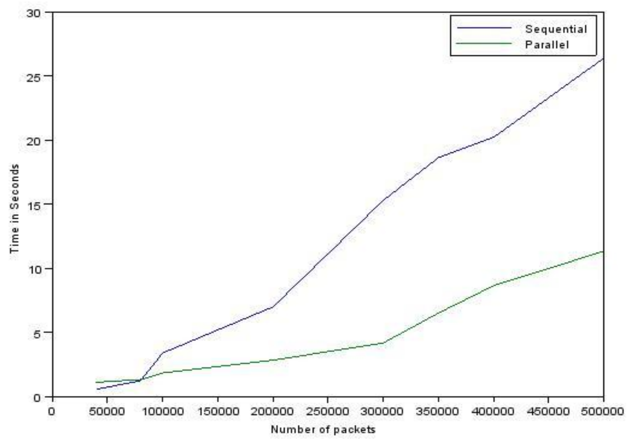

FIGURE 3. COMPARISON OF SEQUENTIAL AND PARALLEL PROGRAM

As we can see when numbers of partition are more i.e. when the size of the capture increases then the parallel program completely outperforms the serial version which is over parallel one in the initial stages. The growth of the sequential program is near to exponential when size of the capture increases, whereas in case of parallel it is comparatively much better. More satisfactory results can be achieved if the CPU is having more number of cores. That is because the SIMD parallelization is directly proportional to the number of cores available in the CPU. The performance results may differ according to the hardware change. Even on similar hardware due to state of the operating systems results may get changed but it will always work better than the sequential counterpart. More steadiness can be brought in by changing the priority of the java process to highest/real-time (on windows) so that the process can get the cores faster and process the packets in real time. Or else some dedicated machine can be used, which only do the offline processing tasks. This solution is specific to the implementation and is not included in the proposal.

VI. Future Work

As a future work we can apply this technique to the following two paradigms:

1. GPU Based:

2. Some implementation platforms like CUDA can be used to parallelize this concept instead of than fork/join framework I have used. GPUs are many core machines thus having much more processing power and are also dedicated to a particular task (in this case it is offline processing). Thus we can imagine how drastically it will change the performance of the system.

2. Reducing packet loss using multi core:

- The reduction in packet loss which is caused by limited capability of a single core to process a packet in time. E.g. on a gigabit Ethernet say you will require 1 ns to process the packet, but the CPU is capable of taking 3 ns time to process it and hence the other packets comes in that time gets lost due to the unavailability of CPU. Instead we can give those packets to different cores for processing in this way loss in the packets due to limited functionality of the CPU can be reduced.

VII. CONCLUSION

In this paper I have investigated the sequential packet processing, its drawbacks due to the sequential nature of the programs that processes these packets offline and compared it with the parallel version of the same which runs on the multi core architecture. I conclude that the parallel version is utilizing the CPU cycles more effectively and hence decreasing the time it takes to process the packets in offline mode. Due to the reduction in time the analyst can take decision fast and can detect the potential threats that can occur in the network and can solve it fast and/or in time. Talking about the results I conclude that the parallel version takes more time compare to sequential counterpart in cases where the size of the trace is small. This is due to the fact that the parallel version has to create threads and then assign it to different cores on the system. This process consumes some time (included in form of overhead) and hence if the data is small to process then the sequential version is working better. As we can see in the graph also when the size of the trace increases the parallel version becomes more and more fast compare to the nearly exponential growth of the sequential algorithm. Hence we can say the parallel version is much better than the sequential version which is currently applied in all the well-known sniffers. Thus, the mechanism can be embedded in the sniffers to make the offline processing faster.

VIII. REFERENCES

[1] S. Ansari, R. S.G. and C. H.S., "Packet Sniffing: A Brief Introduction," in *IEEE*, 2002.

[2] S.-Y. Phang, H. Lee and H. Lim, "Design and Implementation of V6SNIFF: an Efficient IPv6 Packet Sniffer," in *Third International Conference onConvergence and Hybrid Information Technology*, 2008.

[3] M. Abdul Qadeer, A. Iqbal, M. Zahid and M. Siddiqui, "Network Traffic Analysis and Intrusion Detection using Packet Sniffer," in *Second International Conference onCommunication Software and Networks*, 2010.

[4] S. Wang, D. Xu and S. Yan, "Analysis and Application of Wireshark in TCP/IP Protocol Teaching," in *InternationalConference on E-Health Networking, Digital Ecosystems and Technologies*, 2010.

[5] Y. Li, L. Shan and X. Qiao, "A Parallel Packet Processing Runtime System On Multi-Core Network Processors," in *AParallel Packet Processing Runtime System On Multi-Core Network Processors*, 2012.

[6] Y. Li and X. Qiao, "A Parallel Packet Processing Method On Multi-Core Systems," in *10th International Symposiumon Distributed Computing and Applications to Business, Engineering and Science*, 2011.

[7] C. Janssen, "Techopedia," Techopedia, [Online]. Available: http://www.techopedia.com/definition/3393/multi-processing. [Accessed 24 April 2014].

[8] WildPackets, "OmniPeek Network Analyzer," wildpackets.com, 2012.

[9] tcpdump, "tcpdump & libpcap," tcpdump, 2010. [Online]. Available: http://www.tcpdump.org/. [Accessed 10 8 2014].