EAI Endorsed Transactions
on Future Internet
Research Article EAI.EU

Implementation of Network Cards Optimizations in Hadoop Cluster Data Transmissions

Okta Nurika[1] , Mohd Fadzil Hassan[2] and Nordin Zakaria[3]

[1, 2, 3]High Performance Cloud Computing Centre (HPC[3]), Universiti Teknologi PETRONAS

Abstract

In this paper, the previously invented new methods of network card optimization are applied in a Hadoop cluster, where data transfers occur from the Master to the slave node. The slave node's network card setting is optimized subjective to the characteristics of the incoming data transmissions, which are indicated by the overall transmission size and packet size. The throughput comparisons between the optimized network card settings and the default setting conclude that the optimized versions always generate higher throughputs. Synchronously, the optimized settings also minimize CPU cycles utilization as they deploy timer-based polling (passive wait mode), in order to process the received data packets. This novel practice within Hadoop cluster may be replicated by other data cluster vendors, thus improving their data transfer's throughput and efficiency.

Keywords: Hadoop DataTransferProtocol; HDFS; Network Card Optimization.

Received on 16 December 2017, accepted on 18 December 2017, published on 21 December 2017

doi: 10.4108/_____

1. Introduction

This paper is an extension of the previous research about Genetic Algorithm (GA) assisted simultaneous multiple network cards optimization in a data centre [1], which is based on mathematical models described in [2]. The results of the simulations in [1] have been implemented in real physical network cards as published in [3] with the data transmissions sent using Hping3 software. As a diversification of implementations, the optimal network card settings for specific data transmissions would be sought and implemented in network cards residing in a Hadoop cluster. The Hadoop DataTransferProtocol would transmit the data from the storage server that could be the Master node itself to the slave node.

The throughputs of data transmissions from the optimized network cards and the default version are to be compared as a validation of methods proposed in [2], when implemented in a Hadoop data transfer infrastructure.

2. Hadoop Optimization Practices

According to our knowledge, network card based Hadoop data transfer/transmission optimization has not been done. Until recently, the optimizations within Hadoop focus on data compression/encryption, MapReduce, and Hadoop Distributed File System (HDFS).

Some researches related to data compression/encryption optimization in Hadoop cluster are presented by [4 - 10].

Another common Hadoop optimization area is within its MapReduce framework that distributes large scale data processing to slave nodes. A Hadoop with optimized MapReduce called HaLoop was developed and presented in [11] and [12]. Additionally, several other MapReduce optimization methods are accomplished by [13 - 20].

Furthermore, optimizations about HDFS within Hadoop ecosystem, which is based on Google File System (GFS) have also been conducted. For examples, optimized versions of HDFS have been developed by [21] and [22].

In a Hadoop cluster, there are regular data transmissions from the storage server or Master node to the slave nodes. This currently unoptimized segment of the cluster operation is the focus of this paper. The network card optimization methods explained in [1] are to be implemented in a Hadoop cluster to investigate if the intended higher throughputs in data transmissions could be achieved by the Hadoop DataTransferProtocol. The network card configuration of the slave node would be optimized according to the data transmission specifications (overall size and packet size). This proposed practice is expected to increase the data transmission throughput and concurrently reduces CPU cycles utilizations, as a beneficial effect of kernel interrupt minimization.

3. Hadoop Cluster Set-Up and Experiments

A working Hadoop cluster version 2.7.1 on Linux Ubuntu 16.04 LTS machines consisting of Master server and slave node was set to work as described in [23]. The network cards at both Master and slave nodes have 1Gbps of speed specification. Data transmissions of benchmark data would occur from the Master to the slave, by utilizing Hadoop command of 'hadoop fs -put'. The slave's network card was optimized according to the received data transmission characteristics. In this implementation, the network card setting could be configured either through the configuration files located under '/proc/sys/net/core' directory or using the 'sysctl' Linux command [24]. For example, in order to activate passive wait mode of 20 ms, both 'busy_read' and 'busy_poll' files inside '/proc/sys/net/core' directory must be changed to 20,000 because it accepts the polling duration in microseconds instead of milliseconds. Configuring these values via 'sysctl' Linux command would be to type 'sysctl net.core.busy_read=20000' and 'sysctl net.core.busy_poll=20000'. While for changing the watermark value is by altering the 'rmem_default' file under the same directory.

The ad-hoc chosen GA properties were based on the previous statistical analysis on GA convergence [1] that concludes population size of 50 and generation size of 100 to be having the highest average fitness value for all produced solutions, therefore they were taken as population size and generation size respectively. Mutation probability was 0.2 and crossover probability was maintained at 0.9 with single point crossover. Tournament selection method continued to be used.

The GA assisted simultaneous multiple network cards optimization program was then run and the resulted optimal network card settings for every benchmark data transmission were compiled. They were finally implemented in Hadoop slave node's network card and the data transmission throughputs were compared against the ones resulting from default network card setting of the slave node. The throughputs were calculated by dividing the benchmark data transmission size over the duration of data transfer, and the unit would be converted from Bytes/seconds to Megabits/seconds (Mbps) to make it more familiar. The data transmission itself was done by Hadoop DataTransferProtocol. The duration of it was known by observing the Hadoop's log file of the slave node. The start of data transmission was indicated by the initial string of 'Receiving BP-' and the end of it was marked by the last string of 'PacketResponder'. The time information of the start and the end of data transmission were recorded so the duration of data transmission could be calculated. The benchmark data names, specifications, and their discovered optimal network card settings by GA are listed in the next table.

Table 1. Benchmark Data Specifications and Their Optimal Network Card Settings

Benchmark Data	Real Data Size (Byte)	Packet Size (Byte)	Optimal Network Card Setting
data1.txt	50,331,646,500	1500	Passive wait 30 ms
data2.txt	110,673,448,640	1500	Passive wait 140 ms
data3.txt	222,445,765,200	1500	Passive wait 30 ms
data4.txt	64,107,777,780	1500	Passive wait 130 ms

Table 2. Throughput Performances of Default Network Card Setting in Hadoop Cluster Data Transmissions

Benchmark Data	Default Network Card Setting Performances	
	Transmission Duration (seconds)	Throughput (Mbps)
data1.txt	914	440.5
data2.txt	1523	581.3
data3.txt	3106	572.9
data4.txt	880	582.8

Table 3. Throughput Performances of Optimized Network Cards in Hadoop Cluster Data Transmissions

Data	Optimal Network Card Setting Performances		Improvement Rate from Default (%)
	Transmission Duration (seconds)	Throughput (Mbps)	
data1.txt	825	488.06	10.8
data2.txt	1505	588.3	1.2
data3.txt	2878	618.3	8
data4.txt	842	609.8	4.6

The improvement rates contained in Table 3 display the higher throughputs from all the optimized versions of Hadoop slave node's network card, compared to the performances of the default setting in Table 2. Furthermore, since the optimized versions implemented timer-based polling, they also required less kernel interrupt generations, which led to lower power consumption and lower heat production. These conditions are supportive for the network hardware's longevity.

The overall improvement of Hadoop cluster's data transfer has proven the practicality and workability of the proposed optimization methods in a data cluster. Especially with the generic implementation technique that only requires minor network card's configuration change via several command lines, this brings potential for other data cluster vendors to apply the same network card optimization methods to improve their data transfer and maintain their network hardwares' shelf lives. This is also a positive effect for the data centre's financial preservation. The improved data transmission may subsequently accelerate the data analytic process, when the data need to be transferred to the slave nodes before analysis. Considering the regular occurrences of data replications from the master node to the slave node in a data cluster analytic system, network card optimization forms a catalyst part of the data cluster infrastructure.

4. Conclusions

An ad-hoc implementation of the proposed network card optimization methods in Hadoop data cluster, with GA properties set as recommended by the previous experiments' convergence analysis in [1] has been accomplished. The benchmarked data transfers became faster and consumed less CPU cycles. This opens the prospect of other data cluster systems to follow the similar optimization path, in order to increase data replication speed from the master to the slave node, which will ultimately expedite the data analytic phase.

Acknowledgements.
We are grateful towards Universiti Teknologi PETRONAS (UTP) High Performance Cloud Computing Centre (HPC3) for providing the necessary equipments to conduct the experiments.

References

[1] Nurika, O., Hassan, M. F., Zakaria, N., and Jung, L. T. Genetic Algorithm Optimized Network in Cloud Data Centre. *Advanced Science Letters*. 22 (2016), 2705-2709.
[2] Nurika, O., Hassan, M. F., Zakaria, N., and Jung, L. T. Mathematical models for network card simulation and their empirical validations. *In Proceedings of the International Symposium on Mathematical Sciences and Computing Research (iSMSC)*. (2015), 66-71.
[3] Nurika, O., Hassan, M. F., Zakaria, N., and Jung, L. T. REAL THROUGHPUT MEASUREMENTS COMPARISON BETWEEN UNOPTIMIZED AND OPTIMIZED NETWORK CARDS. *Science International*. 29 (2016), 87-91.
[4] Xiang, L. -H., Miao, L., Zhang, D.-F., and Chen, F.-P. Benefit of Compression in Hadoop: A Case Study of Improving IO Performance on Hadoop. *In Proceedings of the 6th International Asia Conference on Industrial Engineering and Management Innovation*. (2016), 879-890.
[5] Lin, H.-Y., Shen, S.-T., Tzeng, W.-G., and Lin, B.-S. P. Toward data confidentiality via integrating hybrid encryption schemes and Hadoop distributed file system. *Advanced Information Networking and Applications (AINA) 2012 IEEE 26th International Conference* on. (2012), 740-747.
[6] Shen, Q., Yang, Y., Wu, Z., Yang, X., Zhang, L., Yu, X., et al. SAPSC: Security architecture of private storage cloud based on HDFS. *Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on*. (2012), 1292-1297.
[7] Park, S. and Lee, Y. Secure hadoop with encrypted HDFS. *In Proceedings of International Conference on Grid and Pervasive Computing*. (2013), 134-141.
[8] Haider, A., Yang, X., Liu, N., Sun, X.-H., and He, S. Ic-data: Improving compressed data processing in hadoop. *In Proceedings of High Performance Computing (HiPC) 2015 IEEE 22nd International Conference*. (2015), 356-365.
[9] Anuradha, D. and Bhuvaneswari, S. A Detailed Review on the Prominent Compression Methods Used for Reducing the Data Volume of Big Data. *Annals of Data Science*. 3 (2016), 47-62.
[10] Usama, M. and Zakaria, N. Chaos-Based Simultaneous Compression and Encryption for Hadoop. *PloS one* 12 (2017).
[11] Bu, Y., Howe, B., Balazinska, M., and Ernst, M. D. HaLoop: Efficient iterative data processing on large clusters. *In Proceedings of the VLDB Endowment*. 3 (2010), 285-296.
[12] Bu, Y., Howe, B., Balazinska, M., and Ernst, M. D. The HaLoop approach to large-scale iterative data analysis. *The VLDB Journal—The International Journal on Very Large Data Bases*. 21 (2012), 169-190.
[13] Zhang, Y., Gao, Q., Gao, L., and Wang, C. imapreduce: A distributed computing framework for iterative computation. *Journal of Grid Computing*. 10 (2012), 47-68.
[14] Zhang, Y., Gao, Q., Gao, L., and Wang, C. PrIter: a distributed framework for prioritized iterative

computations. *In Proceedings of the 2nd ACM Symposium on Cloud Computing*. (2011), 13.
[15] Zhang, Y. and Chen, S. i 2 mapreduce: incremental iterative mapreduce. *In Proceedings of the 2nd International Workshop on Cloud Intelligence*. (2013), 3.
[16] Elnikety, E., Elsayed, T., and Ramadan, H. E. iHadoop: asynchronous iterations for MapReduce. *In Proceedings of 2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*. (2011), 81-90.
[17] Guo, L., Sun, H., and Luo, Z. A data distribution aware task scheduling strategy for mapreduce system. *Cloud Computing*. (2009), 694-699.
[18] Fu, J. and Du, Z. Load balancing strategy on periodical mapreduce job. *Computer Science*. 40 (2013), 38-40.
[19] Abdullahi, A. U., Ahmad, R. B., and Zakaria, N. M. Proposed adaptive indexing for Hive. *In Proceedings of 2015 International Symposium on Mathematical Sciences and Computing Research (iSMSC)*. (2015), 226-231.
[20] Abdullahi, A. U., Ahmad, R., and Zakaria, N. M. Big data: Performance profiling of Meteorological and Oceanographic data on Hive. *In Proceedings of 2016 3rd International Conference on Computer and Information Sciences (ICCOINS)*. (2016), 203-208.
[21] Lai, L., Shen, L., Zheng, Y., Chen, K., and Zhang, J. Analysis for REPERA: A Hybrid Data Protection Mechanism in Distributed Environment. *International Journal of Cloud Applications and Computing (IJCAC)*. 2 (2012), 71-82.
[22] Li, X., Dai, X., Li, W.-j., and Cui, Z. Improved hdfs scheme based on erasure code and dynamical-replication system [j]. *Journal of Computer Applications*. 32 (2012), 2150-2153.
[23] Borthakur, D. HDFS Architecture Guide. Available: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. (2013).
[24] Terrehon, B. B. B., Jorge, N., Shen, F. Documentation for /proc/sys/net/*. Available: https://www.kernel.org/doc/Documentation/sysctl/net.txt. (2009).