Query Bound Application Offloading: Approach Towards Increase Performance of Big Data Computing

[1]Rishikesh Salunkhe,[2]Dr. Naveen J,
[1]M.Tech Student,[2]Associate Professor,
[1]Bharati Vidyapeeth Deemed University,
[1]College of Engineering, Pune, India

Abstract- An offloading framework is used to offload source code among server and it's clients. Source code mainly distributed to I/O node and data node. The offloading mechanism assigns I/O bound and data bound modules to respective I/O node and data node. Data transmission decreased by offloading framework by partition I/O bound and data intensive applications. This system deals with data sharing over a client server application and offloading of executable code [2].
This approach improves performance of the communication between two systems and this performance is evaluated by implementing offloading framework on client server application. By executing five real time applications on heterogeneous and homogeneous environment, we get results which show that offloading framework increases overall system performance by factor up to 90.1% with 75.5% average. Also this testing shows that an offloading framework reduces network load introduced by client server application by factor between 35 and 68.

Index Terms— I/O intensive, offloading, data intensive, high performance computing.

I. INTRODUCTION

Vast network transmission is decreased by migrating data from storage node to I/O node using offloading environment in distributed computing environment.[1], [2] Network traffic load increases heavily by exchanging data among I/O node and storage node in client server application. The application processing data on computing node must be taken from storage node during data execution phase. The performance of the system may decreases by passing more amounts of data among computing node and storage node.[3]–[7] By using Ethernet these I/O problems becomes poor for client server application, where all nodes in the client server application share bandwidth.

Our offloading framework attached with API i.e. Application Programming Interface[1], [8], [9]and a execution time system allows developers to write new client server code without any offloading experience and also allows expanding existing code executing on client server application.[1], [6]

II. RELATED WORK

In this paper [2], [6], [10] author proposes new model for active storage system i.e. Dynamic Active Storage System which enhance data layout for operations and consider bandwidth requirement for that operation. Here author proposed mathematical module for bandwidth predictor data intensive application which preliminary find the required bandwidth for data intensive applications. Data distribution is done by strip data into multiple nodes. At the end author present the architecture and working style of Dynamic Active Storage (DAS).

Author conclude three points in this paper first offloading data intensive applications near to data node increase performance of overall system,[4], [5], [7], [11], [12] second bandwidth[11], [13] predictor measure required bandwidth run time for application and make offloading decision. This module can be tested on Big Data Hadoop framework.

In this paper[2] author propose new Multilevel Active Storage model for effective handling of data intensive operations because data intensive applications has large impact on system performance on HPC and Big data domain. MAS architecture effectively handles both read write as well as complex operations which has high computation requirements. I/O library improvement, file system level improvement runtime level improvement in MAS is achieved here. Results and analysis shows the reliability and performance improvement in case of data intensive applications. In future this Active storage programming model, runtime solutions is useful for big data.

In this paper [3] author proposed Everest framework for reducing the peak I/O request load and burst in data centre workload. Everest client is implemented by two library level and this framework is implemented at user level. Storage is implemented with four nodes as RPC server exporting. This method reduce burst data and peak load I/O by pooling idle disk bandwidth and offloading the write requests this can improve performance of server which face high peak load of I/O.

This paper [4] presents a module, which needs keeping record of each analytical I/O activity and their relative memory access on the storage node in a similar system, to originate a self adapting storage system. Finally, with the knowledge concerning subsist access patterns and also the similarity analysis of access patterns, it's not troublesome to predict the longer term I/O access operations for doable I/O improvement.

This technique is enforced in parallel system level to create a self standardization storage system. It makes I/O access operation very simple for I/O improvement. This research article has concentrated, implement and analyzed a autogenously adapting repository system by using the information associated with each logical I/O intensive procedure on the requestor side and real memory intensive procedures on the repository side i.e. server side. Final conclusion is that current execution of self-adaption repository system; a basic DAG is utilized to arrange all record of I/O events additionally. So as to enhance the perfection of I/O pattern forecasting, we have to account some specific theorems, like nonlinear prediction of chaotic statistic time for higher prediction correctness within close to future.

This research publication [5] proposed method that challenge for simplicity within the development of such not modified information analytics, exploitation analysis and enforcement model that exhibit the potential benefits and drawback of completely contrasting extent of the I/O ranking, jointly with on a system cipher's nodes vs. on individual "staging" nodes committed to survey tasks. Results disclose the significance adaptability in analytics assignment and differentiate the characteristics of analytics procedure that cause total distinct placement selections.

A simple enforcement architecture is employed to quantitatively elaborate the trade-offs in placement In future will focus on (1) to execution time resource management and pipeline reconciliation for I/O graphs and (2) to find out automatic placement of research data processing supported execution time enforcement observance.

In this paper [6] it is observed that parallel filing system (PFS) is commonly familiar save mean results and checkpoint/restart files in a very HPC i.e. high performance computing system. Many applications executing on Associate in tending HPC i.e. High Performance Computing system usually access PFSs at the same time important to decrease and varies I/O performance. In this research publication, we have to focus on our approach exposing and conjointly propose some primary testbed results gather throughout the study. This work allows deeper scanning of our approach in progress analysis in maintain inter-application interference in a PFS. As this method presents a method exploration of IO-Cop, we target to attain higher management over however the PFS answer to access demand from applications.IO-Cop will make easy the PFS modification its runtime behavior supported application access necessities or upgrade access management of the PFS. As it is observed that interferences usually originates from coinciding access by multiple application. As a result of this there are a unit that are restricted range of shared storage servers accessible on a PFS, applications typically share the storage servers.

III. IMPLEMENTATION DETAILS

Here, the development technicalities of query offloading architecture and elaborate how to execute offloading tasks[8], [10], [14] on client and server side is discussed below.

IV. CONFIGURATION

We have used static offloading method for offloading I/O intensive procedure to repository nodes. We need to follow some steps for running a client server application in our offloading framework. These steps are as follows:

1. Design and developed a client server program and recognize I/O intensive tasks to be offloaded to data nodes.
2. Developers should write configuration files for converting client server application into offloading versions which make use API
3. Make.exe files for data nodes when the executables are developed by compiled languages. But source files become executables files if application is implemented by interpreted language.
4. Executables and configuration files are copied to directories on I/O node and data node of application.
5. I/O intensive module starts on data node which is followed by I/O nodes. This sequence is mandatory as offloaded modules should give services to I/O intensive module in a starting phase.

V. OFFLOADING API

This version of the offloading framework gives API only for C and C++ programming language. We can implement same API in other language such as java or python as well.

Offloading environments has initialize and sets up by the init function in the first group. The role of I/O intensive module is decided by init which include program plays by finding a separate command line argument. Init deletes the separate argument after role decision which cannot further used.

All methods are making different in form of addresses after compilation. Similar functions may have distinct addresses in I/O intensive and data intensive modules. According to particular sequence to shifting offloading entries between a combination of I/O intensive and data intensive procedures, we prepare applications to command register methods to register functions and then shift method name instead of address.

Input parameters object are accepted by MARSHAL and UNMARSHAL in type of void * to adjust different type of object. Method names are converted to second set of interface to serialized and after all processed as regular string by using pointer. We have "offload call" function for offload calling. The network address is shown by the address where an offload module takes place. Offloading entrance is specified by the ""func_name" method. Instance of the offloading structure defines input and output parameter as "Ins" "outs".

VI. SHARING DATA

We will think on two different problems related to data sharing. One is how the global knowledge is shared between computing nodes and storage nodes. MARSHAL and UNMARSHAL technique is used to store global data instead of storing object points only and all this data should accessed by both the nodes managed by parameter given as input and final outcome. This is like object generated in address repository are complete distinct in two different components. Keeping track of address of method is done by function pointer. Managing global updates is task of I/O intensive procedures. The other problem is that how to split code segments. In this offloading architecture development, we connect all object cipher to every part, apart from of whether the ciphers are used or not; thus developer does not need to which procedure belongs to which part. We build map among addresses and functions for transferring a function entry thus putting procedure names in offloading requests and responses. Function names and addresses are resolved by computing nodes and storage nodes by using offloading framework.

VII. RESULTS

We choose PostgreSQL as an example, because it is a complicated application with a number of independent modules. Moreover, boundaries of I/O-intensive modules are highly dissimilar. We can easily separate PostgreSQL into computation node and offloading parts.

VIII. OVERALL PERFORMANCE EVALUATION

1. CPU Usage Evaluation

CPU utilization of the system is shown in following figure 4.1. The aim of this testbed was to gain performance effect of offloaded I/O intensive model on data nodes in offloading environment. This target was gained by examining performance of CPU usage of data node for Select query operation. It is very important that analysis if CPU usage because offload I/O intensive module may affect on other I/O modules or services which are executing on data nodes.

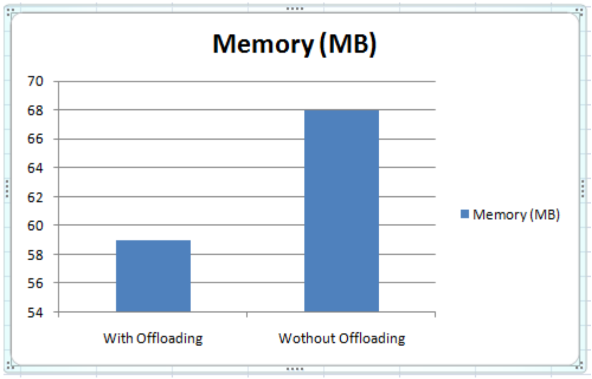

Figure 4.1 CPU Utilization

2. Time Evaluation:

Now we are in a position to evaluate time for with offloading and without offloading time taken by storage nodes to process the query. Time evaluation is done on SQL Select query with and without offloading. With offloading the query took 2 second and without offloading it took 5 second.

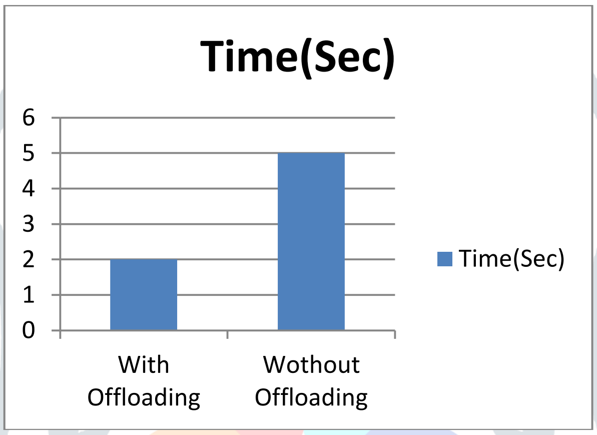

Figure I: Time Evaluation

IX. CONCLUSION

We present offloading architecture programming module which self-regulating offload I/O intensive procedures of applications to data node. This framework decrease network traffic burden occurred by data shifting among data node to computing node. Offloading module allows programmers to simple way of write new I/O module or dividing code to execute easily on clusters without burden of vast traffic load. Our testbed outcomes present that offloading module achieves two primary goals in homogeneous as well as heterogeneous environment. For this purpose, we used PostgreSQL to perform communication between client and server machine.

REFERENCES

[1] J. Naveenkumar, R. Makwana, S. D. Joshi, and D. M. Thakore, "OFFLOADING COMPRESSION AND DECOMPRESSION LOGIC CLOSER TO VIDEO FILES USING REMOTE PROCEDURE CALL," *J. Impact Factor*, vol. 6, no. 3, pp. 37–45, 2015.

[2] D. T. Jayakumar and R. Naveenkumar, "Active Storage," *Int. J. Adv. Res. Comput. Sci. Softw. Eng. Int. J*, vol. 2, no. 9, pp. 62–70, 2012.

[3] N. Jayakumar, T. Bhardwaj, K. Pant, S. D. Joshi, and S. H. Patil, "A Holistic Approach for Performance Analysis of Embedded Storage Array," *IJSTE - Int. J. Sci. Technol. Eng.*, vol. 1, no. 12, pp. 247–250, 2015.

[4] N. Jayakumar, S. Singh, S. H. Patil, and S. D. Joshi, "Evaluation Parameters of Infrastructure Resources Required for Integrating Parallel Computing Algorithm and Distributed File System," *IJSTE - Int. J. Sci. Technol. Eng.*, vol. 1, no. 12, pp. 251–254, 2015.

[5] J. Naveenkumar, R. Makwana, S. D. Joshi, and D. M. Thakore, "Performance Impact Analysis of Application Implemented on Active Storage Framework," *Int. J.*, vol. 5, no. 2, 2015.

[6] P. D. S. D. J. Naveenkumar J, "Evaluation of Active Storage System Realized through MobilityRPC," *Int. J. Innov. Res. Comput. Commun. Eng.*, vol. 3, no. 11, pp. 11329–11335, 2015.

[7] J. Naveenkumar and S. D. Joshi, "Evaluation of Active Storage System Realized Through Hadoop," *Int. J. Comput. Sci. Mob. Comput.*, vol. 4, no. 12, pp. 67–73, 2015.

[8] R. Salunkhe, A. D. Kadam, N. Jayakumar, and S. Joshi, "Luster A Scalable Architecture File System: A Research Implementation on Active Storage Array Framework with Luster file System.," *Int. Conf. Electr. Electron. Optim. Tech.* -, 2016.

[9] S. Naveenkumar Rohan T, "Integrating Intrusion Detection System with Network monitoring," *Int. J. Sci. Res. Publ.*, vol. 4, no. 5, p. 4, 2014.

[10] M. N. Jayakumar, M. F. Zaeimfar, M. M. Joshi, and S. D. Joshi, "INTERNATIONAL JOURNAL OF COMPUTER ENGINEERING & TECHNOLOGY (IJCET)," *J. Impact Factor*, vol. 5, no. 1, pp. 46–51, 2014.

[11] S. D. J. M. J. Naveenkumar Farid, "A GENERIC PERFORMANCE EVALUATION MODEL FOR THE FILE SYSTEMS," *Int. J. Comput. Eng. Technol.*, vol. 5, no. 1, p. 6, 2014.

[12] S. D. J. Naveenkumar Jayakumar Farid Zaeimfar, "Workload Characteristics Impacts on file System Benchmarking," *Int. J. Adv.*

Res. Comput. Sci. Softw. Eng., vol. 4, no. 2, pp. 39–44, 2014.

[13] M. G. KAKAMANSHADI, M. J. NAVEENKUMAR, and S. H. PATIL, "A METHOD TO FIND SHORTEST RELIABLE PATH BY HARDWARE TESTING AND SOFTWARE IMPLEMENTATION," *Int. J. Eng. Sci.*, 2011.

[14] R. Salunkhe, A. D. Kadam, N. Jayakumar, and D. Thakore, "In Search of a Scalable File System State-of-the-art File Systems Review and Map view of new Scalable File system.," in *International Conference on El ectrical, Electronics, and Optimization Techni ques (ICEEOT) - 2016*, 2016, pp. 1–8.

[15] J. Namdeo and N. Jayakumar, "Predicting Students Performance Using Data Mining Technique with Rough Set Theory Concepts," Int. J. Adv. Res. Comput. Sci. Manag. Stud., vol. 2, no. 2, 2014.

[16] N. Jayakumar, "Reducts and Discretization Concepts, tools for Predicting Student's Performance," Int. J. Eng. Sci. Innov. Technol., vol. 3, no. 2, pp. 7–15, 2014.

[17] S. C. Chiu, W. K. Liao, and A. N. Choudhary, "Distributed smart disks for I/O-intensive workloads on switched interconnects," Futur. Gener. Comput. Syst., vol. 22, no. 5, pp. 643–656, Apr. 2006.

[18] C. L. Philip Chen and C. Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on Big Data," Inf. Sci. (Ny)., vol. 275, pp. 314–347, Aug. 2014.

[19] R. A. Kendall, M. Sosonkina, W. D. Gropp, R. W. Numrich, and T. Sterling, "Parallel Programming Models Applicable to Cluster Computing and Beyond," 2012.

[20] K. Qureshi and H. Rashid, "A performance evaluation of RPC, JAVA RMI, MPI and PVM," Malaysian J. Comput. Sci., vol. 18, no. 2, pp. 38–44, 2005.

[21] E. Riedel, C. Faloutsos, G. a. Gibson, and D. Nagle, "Active disks for large-scale data processing," Computer (Long. Beach. Calif)., vol. 34, pp. 68–74, 2001.

[22] E. Riedel and D. Nagle, "Active Disks - Remote Execution for Network-Attached Storage Thesis Committee :," Science (80-.)., no. December, 1999.

[23] C. Paridon and H. P. Corporation, "Storage Performance Benchmarking Guidelines - Part I : Workload Design May 2010 SSSI Member and Author :," no. May, 2010.

[24] D. Hoch, "Extreme Linux Performance Monitoring Part II Extreme Linux Performance Monitoring Part II," pp. 1–10, 2007.

[25] S. D. J. naveenkumar J, Farid Z, M joshi, "A generic performance evaluation model for the file systems," Int. J. Comput. Eng. Technol., vol. 5, no. 1, pp. 46–51, 2014.

[26] F. Gens, "The 3rd Platform: Enabling Digital Transformation," Idc, no. November, pp. 1–13, 2013.

[27] J. Naveenkumar, R. Makwana, P. S. D. Joshi, and P. D. M. Thakore, "Performance Impact Analysis of Application Implemented on

Active Storage Framework International Journal of Advanced Research in Performance Impact Analysis of Application Implemented on Active Storage Framework," Int. J. Adv. Res. Comput. Sci. Softw. engiinerring, vol. 5, no. 2, pp. 1–6, 2015.