ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015

Different Technologies for Improving the Performance of Hadoop

Mr. Yogesh Gupta[1], Mr. Satish Patil[2], Mr. Omkar Ghumatkar[3]

Student, IT Dept,PVG's COET,Pune. Pune,India[1]

Student, IT Dept,PVG's COET,Pune. Pune,India[2]

Student, IT Dept,PVG's COET,Pune. Pune,India[3]

Abstract: Hadoop is a popular open-source implementation of the MapReduce programming model for Cloud Computing. However, it faces a number of issue to achieve the best performance from the underlying system. It include a serialization barrier that delays the reduce phase, repetitive merges and disk access, and lack of capability to leverage latest high-speed interconnects. We express some technologies which are helpful for improve the performance of Hadoop. The technologies are Hadoop-A, iShuffle, TCP/IP implementation of Hadoop-A, Hadoop OFE, Hadoop online are used for regret the performance of Hadoop.

Keywords: Hadoop, MapReduce, JobTracker, TaskTracker**,** MapTask, RDMA, InfiniBand, MOFSupplier, NetMerger, iShuffle, OFE.

I. INTRODUCTION

Hadoop[1] is an open-source implementation of MapReduce[2], currently maintained by the Apache Foundation, and supported by leading IT companies such as Google and Yahoo!.Hadoop implements MapReduce framework which use 2 types of components: 1.JobTracker 2.TaskTracker.JobTracker gives the command to the TaskTrackers to process the data in parallel manner. JobTracker have 2 main functions i.e. map and reduce. In that the JobTracker gives the charges of MapTask and ReduceTask to the TaskTracker. It also monitors their progress and handles the faults by re-executing the task.

There are different issues in MapReduce framework that a) Serialization between Hadoop shuffle/merge and reduce phase b) Repetitive merge and disk access c) unable to use RDMA interconnects. d) MapReduce computations often have "hot spots" in which the computation is lengthened due to inadequate bandwidth to some of the nodes. e) Hadoop-A is implemented based on InfiniBand, which restricts the usage of new algorithms on commercial cloud servers and prevents them from proving their contribution towards solving the disk I/O bottleneck. Several different techniques have been taken to accelerate Hadoop as follows.

Hadoop-A[3] is an acceleration framework that optimizes Hadoop with plug-ins which are implemented in C++ for fast data movement.

Hadoop Online presents a modified version of the Hadoop MapReduce framework that supports online aggregation, which allows users to see "early returns" from a job as it is being computed. The Hadoop Online Prototype (HOP) also supports continuous queries, which enable MapReduce programs to be written for applications such as event monitoring and stream processing.

Hadoop-OFE's approach to acceleration is orthogonal to the methods discussed above. Its goal is to improve the performance of MapReduce in Hadoop by utilizing OpenFlow as the interconnects between Hadoop nodes. One strategy is to make use of the QoS abilities of OpenFlow, which allows control over egress queues in an OpenFlow switch. This makes it possible for different flows to have different priorities over the bandwidth, and allows an application to control this priority setting. Thus applications can dynamically set different priorities to flows. In the case of Hadoop MapReduce there are distinct phases of execution that can benefit by prioritizing traffic on the network.

iShuffle [9], a job-independent shuffle service that pushes the map output to its designated reduce node. It decouples shuffle and reduce, and allows shuffle to be performed independently from reduce. It predicts the map output partition sizes and automatically balances the placement of map output partitions across nodes. iShuffle binds reduce IDs with partition IDs lazily at the time reduce tasks are scheduled, allowing flexible scheduling of reduce tasks.

The rest of the paper is organized as follows- Section (2)Overview of MapReduce Framework,(3) Design of Hadoop acceleration through network levitated merge,(4) TCP/IP implementation of Hadoop A,(5)iShuffle,(6)Hadoop OFE,(7) Conclusion

II. OVERVIEW OF MapReduce FRAMEWORK

Hadoop MapReduce is a pipelined data processing. Hadoop consist three main execution phases i.e. map, shuffle/merge and reduce. In a map phase the JobTracker selects a number of TaskTrackers {TT1, TT2, TT3…} and schedule them to run the map function. The mapping function in a map task converts the original records into intermediate result. These new records are stored as a MOF(i.e. Map Output Files). In the second phase when MOFs are available the JobTracker selects the TaskTrackers to run the reduce task. Typically, there is one segment in each MOF for every ReduceTask. So, a ReduceTask need to fetch such a segment for all MOFs{MOF1,MOF2...}. Globally these phase operation lead to an all-to-all shuffle on data segments among all the

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015

ReduceTask. Shuffle and merge of data segment by ReduceTask is called the copy phase of Hadoop. In the third or reduce phase each ReduceTask loads and process the merge segment using the reduce function. The final result is store in HDFS[4].

A. Issues in the MapReduce framework

There are various issues in the existing Hadoop MapReduce framework these are (a)A serialization in Hadoop data processing (b)Repetitive merge and disk access(c)The lack of support for RDMA interconnects.

1. Serialization in Hadoop data processing:

Hadoop process the large datasets in pipelined manner. There are two phases which processed in pipelining architecture: 1.Map 2.Shuffle/Merge. After the initialization multiple MapTask start with the map function on the first set of data splits. Whenever the MOFs are generated from these splits then set of ReduceTask initiates the fetch partition through these MOFs. At every ReduceTask when total data size is more than a memory threshold then the smallest datasets are merged. In MapReduce programming model, the reduce phase does not execute until the map phase get executed with all data splits. MapReduce pipeline architecture has an implicit serialization. At every ReduceTask whenever the merge and shuffle operations are completed on each data splits then reduce phase initiates to process data segment using reduce function. Because of this serialization, reduce phase will be delayed.

2. Repetitive merge and disk access:

ReduceTask merge data segment when the number of segment or their total size grows over a threshold a newly merge segment has to be spilled to local disk due to memory pressure. In the existing merge algorithm in Hadoop leads to more repetitive merge therefore the extra disk is accessed. When more segments are arrive then the threshold will be broken. It is vital to choose a different policy for merge to minimize the additional disk accessed. An alternative merge algorithm is important for reduce the drawback i.e. repetitive merge and associated disk access for Hadoop.

3. Unable to use RDMA interconnects:

The existing Hadoop is not taking the advantage of high performance RDMA interconnect technology that have high performance computing such as InfiniBand[5].However the RDMA supports high bandwidth and less CPU utilization. To run the Hadoop on TCP/IP will not leverage the strength of RDMA.

III. DESIGN OF HADOOP ACCELERATION THROUGH NETWORK LEVITATED MERGE

As per the issues discuss in section 2, it's important to overcome it for improve the performance of Hadoop. The Hadoop-A(Hadoop Acceleration) is a technique which accelerate the Hadoop's MapReduce framework and overcome the limitation of it. An acceleration framework take the advantages of RDMA interconnect and different merge technique for boost up the performance of Hadoop framework.

A. Architecture of Hadoop-A

In figure1 Hadoop-A design two new user-configurable plug-ins are added in framework that is a) MOFSupplier b) NetMerger. These plug-ins are use the RDMA interconnect and different alternative merge algorithm. The MOFSupplier and NetMerger both are developed in C++ with Object-Oriented principles. The Acceleration Framework consists 3 techniques for the implementation as follows:

1. User-Transparent Plug-in: These plug-in are developed for user to enable or disable the acceleration for execution which is controlled by a parameter in the configuration file. The user-transparent in a two ways (1) No changes are introduced in scheduling and monitoring of TaskTracker and MapTask (2) No modification has been made into the submission and control interface between user program and JobTracker.

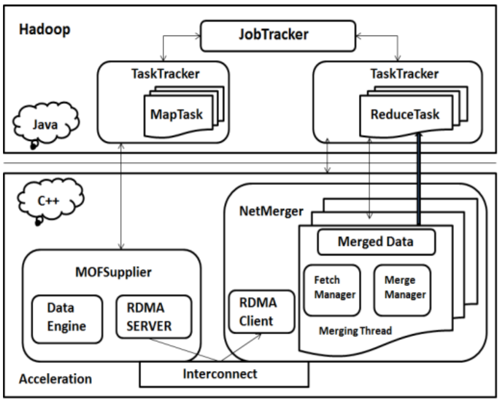

Fig 1: Hadoop Architecture

2. Multithreaded and Componentized MOFSupplier and NetMerger: The MOFSupplier consist RDMA Server which handles the fetch request and ReduceTask. It also consists the data engine that manages the index and data files for the MOFs are generated by local MapTasks.

3. Event-Driven Progress and Coordination: In this approach for synchronizing with Java-side components provide the event channel between MOFSupplier and NetMerger plug-ins and Hadoop framework. This is also used to coordinate activities and monitor progress for internal components of MOFSupplier and NetMerger.

B. Program Flow

(1) Fetching Header of Segments (S1, S2...) {H1(S1,<key,val>), H2(S2,<key,val>),..}
(2) Build Priority Queue (PQ) by using Key and Value of Segment until all Header arrived.
(3) Store root record as First Record(RR=H1)
(4) Fetch and Merge the Record concurrently which is not already merged.
(5) Deliver Merge data to Reduce Task

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
IJARCCE
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015

IV. TCP/IP IMPLEMENTATION OF HADOOP A

A TCP/IP Implementation of Hadoop Acceleration has two components MOFSupplier (Server) and NetMerger (Client). Multithreading technologies are used to manage memory pool, send/receive and merge data segments. A Map Reduce Framework has two file systems Google File System (GFS) [6] and Hadoop Distributed File System (HDFS) [4]. On the top of the Hadoop program, Apache Hive and pig are two applications for dealing with large amount of data in hadoop.

Figure 2 shows that the relationship between these components mentioned as earlier. Apache Pig [7] and Hive [8] are deals with the large amount of data. Hadoop can support applications running on large commodity cluster and Hadoop Distributed file system provides data storage mechanism. The TCP/IP implementation of Hadoop-A is used to improve the performance of Hadoop. It includes two components MOFSupplier and NetMerger connected with TCP/IP socket protocol via Ethernet.

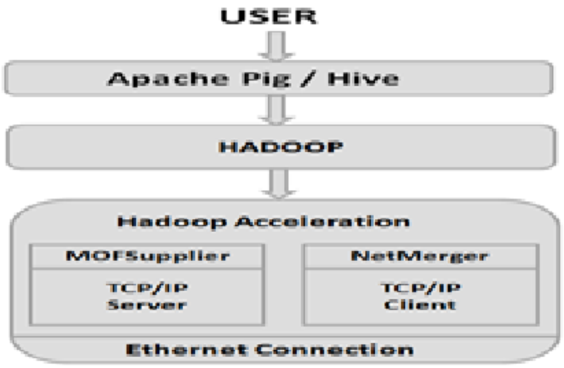

Fig 2 : Layered Architecture

A. *TCP/IP by Hadoop-A Architecture*

1 .Epoll in Linux kernel:

Epoll is an I/O event notification mechanism used in high performance network communication. It is used to replace traditional POSIX poll and select system calls. Here are some benefits of epoll over old poll/select mechanism: (1) the disadvantage of select is that the number of opened file descriptors (FD) is limited, which is sometimes not enough for the server; epoll does not have this limitation, and the largest number of FD can be opened, which is more larger than 2048; (2) another disadvantage of traditional select is that when you obtain a large set of sockets, due to network delay, only some of the sockets are active, but select/poll still scans all of the socket set linearly, which can lead to efficiency proportional penalties. (3) select, poll and epoll, all require the Linux kernel to provide information to the user space; as a result, avoiding useless memory copies is very important. Epoll solves this problem with the help of map via shared memory.

2. Multithreading:

As we know, disk I/O is always the bottleneck and data movement is expensive and time consuming. Consider the case where we only use one thread to read data from disk. When we get all the data we need in the memory, we send these data to the receiver. After the receiver gets this data, it does some calculation and writes data back sequence. We can make use of this multithreading technology to overlap the execution of this process. For instance, we can start a thread to read data from the disk, at the same time letting another thread send data. In the same way, we can also keep one thread receiving data while another thread computing the received data. For the purpose of increasing the speed of sending or receiving data over Ethernet.

3. Buffer allocation management:

One of the important resources in computing systems is memory. In MOFSuppliers, the program firstly allocates many buffers to a Memory Pool, once a Mapper write new Map Output data on the disk, when the disk read thread will get new empty buffer from memory pool to read data from disk. As long as NetMergers receive data, the receive thread get an empty buffer from Memory Pool and give this buffer with all data to merge thread.

4. Program Flow:

Figure 3 can give you a view of the flow of the program

(1) When a Reduce Task needs to fetch data from MapTask, it will send a fetch request to the NetMerger.

(2)NetMerger creates a connection with MOFSupplier and sends fetches request to MOF- Supplier.

(3) After receiving the request from NetMerger, MOFSupplier adds the request to the request queue, and notify DataEngine. Based on the request, DataEngine searches its Data Cache which is read from disk by the disk read thread.

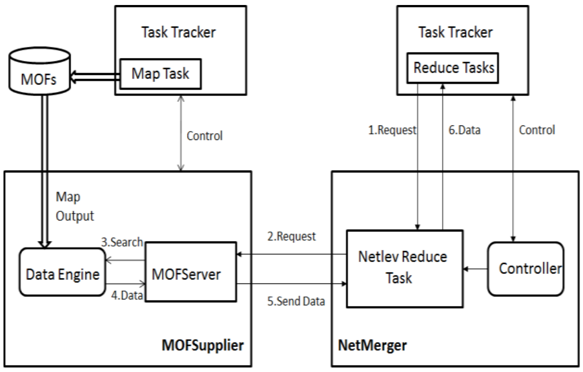

Fig 3: Program Flow

(4) If the required data has been found, DataEngine sends the data back to MOFServer.

(5)MOFServer invokes some send threads to send data back to the NetMerger.

(6)NetMerger uses many threads to receive data and gives received data to Merge Thread to do computation. As soon as computation has been done data will be sent to the Reduce Task.

V. ISHUFFLE

In the Hadoop, the delay in job completion, the coupling of the shuffle phase and reduce tasks which leaves potential parallelism between multiple waves of map and reduce is unexploited, fails to address data

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015

distribution skew among reduce tasks, and makes task scheduling inefficient. In this work, we propose to decouple shuffle from reduce tasks and convert it into a platform service provided by Hadoop. The iShuffle[9], a user-transparent shuffle service that pro-actively pushes map output data to nodes via a novel shuffle-on write operation and flexibly schedules reduce tasks considering workload balance.

A. iShuffle Design

iShuffle, a job-independent shuffle service that pushes the map output to its designated reduce node. It decouples shuffle and reduce, and allows shuffle to be performed independently from reduce.

1. Overview :

Figure 4shows the architecture of iShuffle. iShuffle consists of three components: shuffler, shuffle manager, and task scheduler. The shuffler is a background thread that collects intermediate data generated by map tasks and predicts the size of individual partitions to guide the partition placement. The shuffle manager analyses the partition sizes reported by all shufflers and decides the destination of each partition.

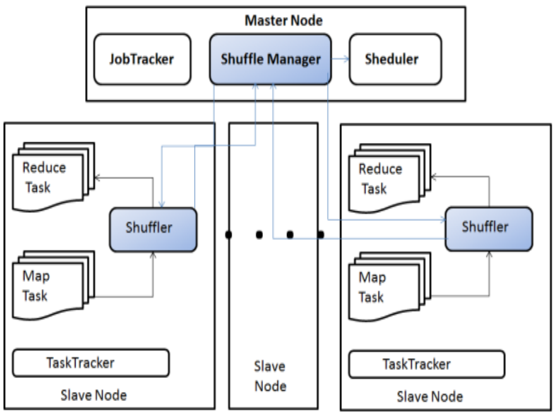

Fig 4: Architecture of iShuffle

User-Transparent Shuffle Service - We design shufflers and the shuffle manager as job-independent components, which are responsible for collecting and distributing map output data.

Shuffle-on-Write - The shuffler implements a shuffle-on-write operation that proactively pushes the map output data to different nodes for future reduce tasks. Every time such data is written to local disks. The shuffling of all map output data can be performed before the execution of reduces tasks.

Automated Map Output Placement- The shuffle manager maintains a global view of partition sizes across all slave nodes. An automated partition placement algorithm is used to determine the destination for each map output partition. The objective is to balance the global data distribution and mitigate the non-uniformity reduce execution time. The task scheduler in iShuffle assigns a partition of a reduce task only when the task is dispatched to a node with available slots. To minimize reduce execution time, iShuffle always associates partitions that are already resident on the reduce node to the scheduled reduce.

2 .Shuffle-On-Write :

iShuffle decouples shuffle from a reduce task and implements data shuffling as a platform service. This allows the shuffle phase to be performed independently from map and reduce tasks. The introduction of iShuffle to the Hadoop environment presents two challenges: user transparency and fault tolerance. Besides user-defined map and reduce functions, Hadoop allows customized partitioner and combiner. To ensure that iShuffle is user-transparent and does not require any change to the existing MapReduce jobs, we design the Shuffler as an independent component in the TaskTracker. It takes input from the combiner, the last user-defined component in map tasks, performs data shuffling and provides input data for reduce tasks. The shuffler performs data shuffling every time the output data is written to local disks by map tasks, thus it name the operation shuffle-on-write.

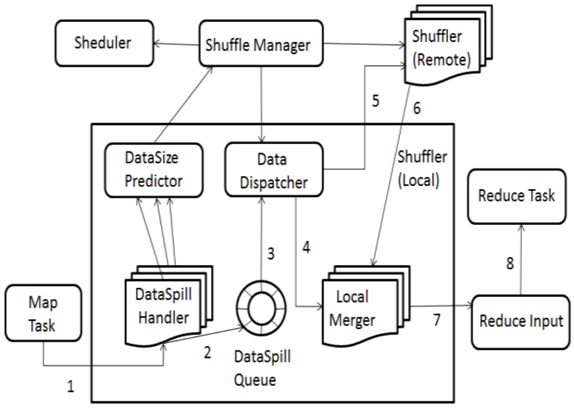

Fig 5: Workflow of shuffle write

Map output collection - The shuffler contains multiple DataSpillHandler, one per map task, to collect map output that has been written to local disks. Map tasks write the stored partitions to the local file system when a spill of the in-memory buffer occurs. It intercepts the writer class IFile. Writer in the combiner and add a DataSpillHandler class to it. While the default writer writing a spill to local disk, the DataSpillHandler copies the spill to a circular buffer, DataSpillQueue, from where data is shuffled/ dispatched to different nodes in Hadoop.

Data shuffling - The shuffler proactively pushes data partitions to nodes where reduce tasks will be launched. Specifically, a DataDispatcher reads a partition from the DataSpillQueue and queries the shuffle manager for its destination. Based on the placement decision, a partition could be dispatched to the shuffler on a different node or to the local merger in the same shuffler.

Map output merging - The map output data shuffled at different times. It needs to be merged to a single reduce input file and sorted by key before a reduce task can use it. The local merger receives remotely and locally shuffled

DOI 10.17148/IJARCCE.2015.4576

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015

data and merges the partitions belonging to the same reduce task into one reduce input. To ensure correctness, the merger only merges partitions from successfully finished map tasks.

VI. HADOOP OFE

In recent years, data intensive programming using hadoop and MapReduce is more increased. Hadoop's implementation of MapReduce in a multi rack cluster is dependent on the top of the rack switches and of the aggregator switches connecting multiple racks. In Hadoop OFE, combine the OpenFlow (OF) enabled switches and a modified JobTracker in Hadoop that is OpenFlow. Hadoop OFE is used for improving the performance of Hadoop. Hadoop-OFE on standard ethernet can provide good performance to Hadoop over specialized interconnects.

The performance improvements by Hadoop-OFE, performing experimental studies using: 1) the MalStone Benchmark [10]; and, 2) an open source Hadoop based application (Matsu) [11] for processing satellite images to detect floods and other phenomena.

A. Hadoop-OFE Design

Hadoop-based applications are widely available in market, the Map and Reduce phases of the computations are required different network requirements. Also, many Hadoop applications are in iterative manner because of that it requires different network requirements for different phases of the iteration. In principle, if the network topology of the cluster can be required to support these requirements, greater efficiency could be achieved when processing data with Hadoop.

Following figure 6 and figure 7 shows a Hadoop cluster with and without OpenFlow networking. To explain the benefits of OpenFlow, consider the following example.

As shown in figure 7, JobTracker is modified to get the OpenFlow Controller to change the properties of flow paths dynamically, depending upon the execution stage of a job. During a Map phase, the flow-path between systems A, B and system F (which holds input data) can be assign higher priority for passing the data required by job. Likewise, during a Reduce phase the flow-path between systems A, B and E (which performs Reduce) assigns higher priority.

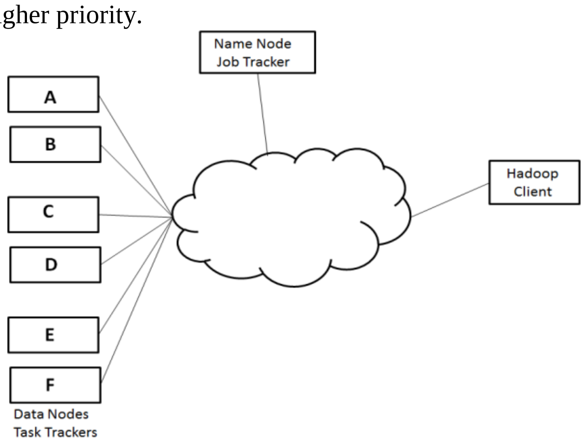

Fig 6 :.Hadoop cluster

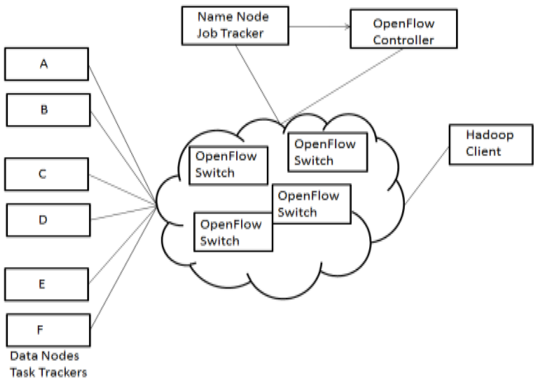

Fig 7: Hadoop cluster with OFE interconnectivity

VII. CONCLUSION

As per the above points ,the Hadoop-A through Network Levitated Merge doubles the data processing throughput of hadoop and reduce CPU utilization by more than 36%.The iShuffle reduce the Job selection by 30.2% than existing Hadoop. Hadoop-A by TCP/IP achieve the good scalability and also 26.7% execution time outperforms than Hadoop. Hadoop-OFE on standard Ethernet can provide good performance to Hadoop over specialized interconnects, like InfiniBand.

ACKNOWLEDGMENTS

Thanks to our project guide **Prof. S. A. Mulay**, Head of Department **Prof. G. V. Garje** and all staff members of Information Technology department of PVG's COET, Pune.

REFERENCES

[1] Apache Hadoop Project. http://hadoop.apache.org/.
[2] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. Sixth Symp.on Operating System Design and Implementation (OSDI), pages 137–150, December 2004.
[3] Yandong Wang, XinyuQue, Weikuan Yu, Dror Goldenberg, DhirajSehgal, LiranLiss. Hadoop Acceleration Through Network Levitated Merge, SC11, Seattle, WA.
[4] onstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), pages 1–10.
[5] Infiniband Trade Asso. http://www.infinibandta.org.
[6] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, "The Google File System," pub. 19th ACM Symposium on Operating Systems Principles, Lake George, NY, Octo- ber, 2003.
[7] hristopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins, "Pig latin: a not-so-foreign language for data processing" In SIGMOD08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pages 10991110, New York, NY, USA, 2008. ACM.
[8] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang 0002, Suresh Anthony, Hao Liu, and Raghotham Murthy, "Hive - a petabyte scale data warehouse using hadoop," In ICDE, pages 9961005, 2010.
[9] YanfeiGuo, JiaRao, and Xiaobo Zhou "iShuffle: Improving Hadoop Performance with Shuffle-on-Write" in 10th International Conference on Autonomic Computing (ICAC '13)
[10] Collin Bennett, Robert L. Grossman, David Locke, Jonathan Seidman and Steve Vejcik, MalStone: Towards a Benchmark for Analytics on Large Data Clouds, The 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2010), ACM, 2010.
[11] Daniel Mandl, Stuart Frye, Pat Cappelaere, Robert Sohlberg, Matthew Handy, and Robert Grossman, The Namibia Early Flood Warning System, A CEOS Pilot Project, IGARSS 2012.