Decentralized Data Offloading in High Performance Computing Centers using Scratch Space

P. Nishmi Irin, K. John Peter, I. Nancy Jeba Jingle

Abstract: ***This project addresses the issues associated with providing Decentralized Data Offloading service to HPC Centers. HPC centers are High Performance Computing centers that use Parallel Processing for running advanced applications more reliably and efficiently. The main concept of this project is the offloading of data from a HPC Center to the destination site in decentralized mode. This project uses the decentralization concept where it is possible for the end user to retrieve the data even when the center logs out. This is possible by moving the data from the center to the Scratch Space. From Scratch space the data is moved to the intermediate storage nodes 1..n and from the nth node the data is transferred to the destination site within a deadline. These techniques are implemented within a Production Job Scheduler which schedules the jobs and Bit Torrent tool is used for data transfer in a decentralized environment. Thus the total offloading times are minimized; data loss and offload delays are also prevented.***

Index Terms: ***High Performance Data Management, HPC Center Serviceability.***

I. INTRODUCTION

HPC centers are High Performance Computing centers that use Parallel Processing for running advanced applications more reliably and efficiently. The data sets are required to be offloaded to end-user locations. This project uses the decentralization concept where it is possible for the end user to retrieve the data even when the center logs out. This is implemented using the Distributed Hash Table Algorithm. As the complexity and size of applications increase, the offloading of data rates decreases. It is impractical to store all user data indefinitely at the center. The local storage in HPC centers, the scratch space, is used for job input, output, and intermediate data, which is typically on the order of terabytes. Scratch space is a space on the hard disk drive which is used for temporary storage and it is in the order of Terabytes. Scratch is built using a parallel file system that supports very high aggregate I/O throughput. To ensure efficient I/O and faster job turnaround use of scratch by applications is encouraged. HPC centers enforce purge policies to manage the precious scratch space, wherein data is deleted based on a time window, ranging from a few hours to a few days. As centers become crowded, the purge policies get more stringent to provide space for incoming jobs. The purge window is, therefore, a product of the center’s load, its provisioned storage, and its desire to maintain a certain level of serviceability.

Manuscript received on July, 2012.

P. Nishmi Irin, Computer Science, Vins Christian College of Engineering, Anna University, Nagercoil, India,

K.John Peter, Information Technology, Vins Christian College of Engineering, Anna University, Nagercoil, India.

I. Nancy Jeba Jingle, Computer Science, Vins Christian College of Engineering, Anna University, Nagercoil, India.

II. RELATED WORK

This project is enhanced to send messages through the intermediate nodes to reach the destination. The nodes can be monitored and directory sharing is also implemented. In concern to security, the project is enhanced to provide private conversation between the nodes using the Random walk algorithm. The message can be transferred directly from one node to another node, many nodes to one node, one to many nodes, many to many nodes. In one to one transfer, a participating node can share message with another node belonging to the same channel publically. Similarly many nodes of the same channel can share message to another node of the same channel. A single node can share a message with multiple nodes of the same channel publically. Multiple nodes of the same channel can share a message with another set of multiple nodes belonging to the same channel. All these transfers can be implemented by means of Distributed Hash Table Algorithm. However, in the case of Private communication the DHT algorithm fails and is less compatible. Hence Random Walk Algorithm is used for Private communication between the nodes.

In concern to security the message can be offloaded between any two nodes privately. This private communication is implemented to provide security for the messages and for Preventing Unauthorized Access. It uses RW Algorithm to implement private communication between the authorized nodes. Random Walk Algorithm is a conservative search algorithm which belongs to DFS Methods. Query source sends a query message called walker to one of its neighbors. The neighbor node sends the walker to all of its neighbors except source. Search cost is reduced. Coverage of RW grows with hop counts. Offloading rates can be reduced to about 90.4%. Network Traffic can be prevented. Prevents loss of data and avoids Offload delays. It provides security to message transfer between nodes. A streaming system usually has multiple channels, and peers may form multiple groups for content distribution. In this project, a distributed overlay framework for dynamic groups where users may frequently hop from one group to another while the total pool of users remain stable has been proposed. The mesh supports dynamic host joining and leaving, and will guide the construction of delivery trees.

III. PROPOSED SYSTEM

In the proposed System, a more enhanced scheme for decentralized offloading scheme for job output data is designed. This project has been enhanced to share the entire directory of a particular node. So it is not necessary to download each file separately all the time from the scratch space. By the use of Directory sharing it is possible to download all the files at a time from the particular directory of a node. Hence the total offloading times can be reduced and performance can be improved significantly. The DHT algorithm alone cannot implement this concept. Hence the Distributed Hash Table algorithm is combined with the Random Walk algorithm to implement the directory sharing concept. In this project, a distributed overlay framework for dynamic groups where users may frequently hop from one group to another while the total pool of users remain stable has been proposed. The mesh supports dynamic host joining and leaving, and will guide the construction of delivery trees.

A decision making component is designed that factors in parameters such as a center's purge deadline, user delivery schedule and a snapshot of current network conditions between the center and the end user, to determine the most suitable approach to offload. Active monitoring is employed using the Network Weather Service (NWS), to make the data offload process react to bandwidth degradation, thus ensuring that a user-specified delivery constraint or a purge deadline can be met. Erasure coding schemes are utilized to ensure that the offload is fault tolerant. The offloading service components have been implemented and evaluated using both trace-driven simulations, using a realistic simulator, as well as actual tests using the Planet Lab test bed. Finally, the solution is integrated with a job submission system.

We design a new software component, Data Offload Manager, to capture the interactions and drive the offloading process. The Manager is integrated into the HPC center management software suite, and is provided with a number of critical center parameters and job descriptions to guide its operation. Compared to a direct transfer, our techniques have the added benefits of resilience in the face of end-resource failure and the exploitation of orthogonal bandwidth that might be available in the end-to-end data path.

A decision making component is designed that factors in parameters such as a center's purge deadline, user delivery schedule and a snapshot of current network conditions between the center and the end user, to determine the most suitable approach to offload. Active monitoring is employed using the Network Weather Service, to make the data offload process react to bandwidth degradation, thus ensuring that a user-specified delivery constraint or a purge deadline can be met. Erasure coding schemes are utilized to ensure that the offload is fault tolerant. Finally, the solution is integrated with real-world tools.

A.Channel Detection

Each user creates a node with a user name and a common password applicable to the entire channel. The users of another channel have to create their node using a different password applicable to their channel. If the password is wrong then the user cannot get access to the channel. Special directives are provided with which users can annotate their job scripts. These directives are parsed by the offload manager to extract and maintain a list of user-specified intermediate nodes. The data offload manager then submits the job to the scheduler. Thus the overlay of intermediate nodes becomes an integral part of the job and can be used for the delivery of the job's result data. End users can further qualify the intermediate node specification with usage policies, which specify the available storage and the load threshold at the intermediate node.

B. Channel Discovery

A new node is merged to available channel by entering the common password of that particular channel. If the password is correct then the node of that particular channel is discovered. If the password is wrong then the node belongs to another channel. The intermediate nodes are selected from among the participating sites that are interested in the data transfer. The distributed and decentralized communication substrate provided by structured p2p networks is utilized to locate Nis in a dynamic environment. A p2p overlay is used to arrange sites that intend to participate in the collaborative offload. Overlay provides reliable communication with other participants in the network. Nis use the overlay to advertise their availability to other nodes in the overlay using random broadcast. Nodes that receive these messages build local information about available nodes for offload. A given node can use its own policies and information about a remote node's capacity to make a decision regarding whether to use the remote node for the offload. Ns interacts with the center to sort the Nis with increasing latency from the center and with decreasing latency from Ns. The sorted set of nodes is provided to the center to utilize as the intermediate nodes, and becomes an integral part of the job's workflow.

C. Decentralized Transfer

Distributed Hash Table Algorithm is an algorithm that works in a decentralized environment and provides a look up service similar to Hash Table (key, Value) pairs**.** Any participating node can efficiently retrieve the value associated with the given key. DHT algorithm scales the no. of nodes, node arrivals, departures and failures with minimal disruption. Each peer node is identified by a (x, y) pair. A node wishing to join the overlay locates an initial node, referred to as a bootstrap node.

The joining node sends a REGISTER with the joining node's Node-ID to the bootstrap node. The bootstrap responds with the information about the nodes that will be near to the joining node. Each peer maintains links to successors and to indexed peers.

They guarantee the connectivity of the network. Peer p first communicates with a random peer pB already in the network. Peer pB is called the bootstrapping peer and is responsible for finding the successors s1; . . . ; s4 of p. p locates the information that it should store. p uses its successors s1; . . . ; s4 to locate its neighbors and to obtain the required information. Peer p first transfers the information to the successor node s1. From s1 the data is transferred to the next successor s2…sn. From sn, the end user retrieves the data. Thus data is offloaded from center to the end user. A Data Offload Manager is designed to capture the interactions and drive the Offloading process. The Manager takes as input the guidelines regarding job specification from the job submission system. The specifications include the output data size, S, the job's data delivery schedule and JSLA. Then the Offload manager provides an offload schedule, Os. The Data offload Manager Shows the files downloaded in the Scratch Space and Shares with all nodes of that particular channel.

D. Bandwidth Management

Each participating node joins a clique, which is a group of sensors that measure bandwidth. A token is passed around which serves as an indication to a node to probe other nodes for available bandwidth. The replies are recorded not only at the node, but also at a central NWS repository. The token is then forwarded to the next node. The clique gives the center an estimate of the bandwidth available from it to different nodes. The center uses this information to decide whether a chosen fan out is sufficient to meet a particular SLA, or needs to be increased. If needed, additional nodes from the set of Nis can be chosen to increase the fan out and meet the SLA.

The Node Manager is responsible for maintaining Intermediate nodes Nis. The SLA Compliance module uses bandwidth predictions provided by NWS to guide the offload process in meeting the SLAs. The Erasure coding module transforms the data to be sent out into error-coded chunks. The Transfer Module is charged with pushing out the encoded chunks to the next level intermediate nodes. Finally, at the heart of the system is the Offload Manager that integrates all the modules and uses them to select different offload schedules and to enable the transfers.

E. Job scheduling

HPC centers utilize job management systems to ensure proper operation. Typically, the job submission system constitutes a user job script and a resource manager at the supercomputer center that schedules the jobs based on a queuing system. A way for specifying intermediate nodes and delivery deadlines as annotations is devised within a standard PBS script. These annotations are specified as directives, much like other PBS directives. The intermediate nodes can be further qualified with policy specification that captures usage constraints. These constraints include the amount of space available for offload on a node, and the node's availability.

More fine grained policies can be easily added. In an instrumented PBS script, an user specifies the stage out to a destination, the use of intermediate nodes with their space constraints, a port number where our transfer protocol is listening, and a delivery deadline. To handle the instrumented job script, we have implemented a parser that runs on the HPC center. When an annotated PBS script is submitted for execution to the job scheduler at the HPC center, it is intercepted by our parser that filters out directives specific to data offloading, and passes those details to the Offloading Service for data delivery. The remaining PBS script is then handed over to the PBS queue for standard processing. The Offloading Service is aware of the center's purge deadline and attempts to reconcile that with user delivery deadline and intermediate/landmark nodes to achieve a desired data transfer schedule. Once a set of intermediate nodes is selected using NWS, we use scatter-gather protocol to transfer the file from the center to the selected intermediate nodes.

The offload happens as follows: The Offload Manager creates a metadata "torrent" file for the subset of data to be transmitted to a set of chosen intermediate nodes. The Manager also provides tracking services so that the intermediate nodes may know what data has been transmitted to which node. Once the nodes receive the torrent file, they use the metadata information along with the tracker to "download" the data subset to their local storage. The process is repeated at all the intermediate node levels. The end host can also use appropriate torrent files to download the result data from the intermediate nodes, thus completing the offloading process.

IV. RESULT

A realistic simulator is developed for the offloading process, which models both job execution and data offloading. A number of large jobs are first scheduled in the order they arrive, until a majority of the machine's resources is allocated. Next, smaller jobs are scheduled. However, such larger jobs can leave a small but significant number of cores idle. Back filling helps to avoid this by assigning smaller jobs to the idle cores. It utilizes a number of different traces to provide an accurate model of the system. The job traces provides the arrival time, start time, total job execution time, and the resources used.

Additionally, the traces also contain the amount of physical memory and virtual memory used by a job. The bandwidth traces provide pair wise bandwidth measurements for the 50 sites over duration of 96 hours. It provides an output trace with information about overall scratch space usage and the time it would take to offload the required data for a given job. This information can then further be used to determine any delay in meeting job scheduling deadlines. Each simulated node is assigned a measured trace. Since there are more nodes in the simulator, some nodes will have duplicate bandwidth traces. Nodes running for longer than 96 hours simply loop through their associated trace. It provides an output trace with information about overall scratch space usage and the time it would take to offload the required data for a given job. This information can then further be used to determine any delay in meeting job scheduling deadlines. It maintains a pool of nodes arranged in a configurable topology to use as intermediate nodes.

Nodes are randomly selected to facilitate the simulated offload. If a node is used for multiple offloads at the same time, the bandwidth is equally divided between the offloads. Moreover, it can also capture varying storage capacities of the nodes and can alter offloading paths based on the capacities. Here, we are mainly concerned with moving the data from the center to the first-level intermediate nodes only. The main driver is a Job tracker that reads the logs, and selects an appropriate action for the simulator to take. At each job arrival, the tracker places it in a wait queue. The job input data staging is then started. The staging process may take many simulator ticks depending on the size of the input data, but once the process completes the job is moved to a run queue. The job will wait there until sufficient compute resources to run the job become available. Once the job completes its execution, it moves to the offload queue. Finally, it also provides accounting and statistics about the offload process, such as the scratch space used and the data read, as well as other vital statistics.

V. CONCLUSION

Since distributed offloading is highly competitive, it raises new research questions in terms of the strategic placement, and selection, of intermediate nodes between an HPC center and end-user destinations. Advanced Networking and Parallel Processing techniques can be implemented along this project to make it more accurate. This Project can be simulated to return the total scratch space usage and the nodes can be monitored to produce a better Offloading rates. This project is enhanced to provide Private Communication between the Nodes to provide one to many communication and many to one communication using the Random walk Algorithm. The data offloading rates have been significantly reduced by using the Distributed Hash table algorithm and the offloading delays are minimized. The entire directory can be shared easily. Thus the Offloading rates are reduced and the transfer speed is increased.

REFERENCES

1. F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," in Proc., 2002.
2. J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke, "GASS: A Data Movement and Access Service for Wide Area Computing Systems," in Proc., 1999.
3. M. Gleicher, "HSI: Hierarchical Storage Interface for HPSS," in Proc., 2010.
4. J.W. Cobb, A. Geist, J.A. Kohl, S.D. Miller, P.F. Peterson, G.G. Pike, M.A. Reuter, T. Swain, S.S. Vazhkudai, and N.N. Vijayakumar, "The Neutron Science Teragrid Gateway: A Teragrid Science Gateway to Support the Spallation Neutron Source: Research Articles," in Proc., 2007.
5. M. Christie and S. Marru, "The Lead Portal: A Teragrid Gateway and Application Service Architecture: Research Articles," in Proc., 2007.
6. R. Wolski, N. Spring, and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing," in Proc, 1999.
7. H. Monti, A.R. Butt, and S.S. Vazhkudai, "Timely Offloading of Result-Data in Hpc Centers," in Proc., 2008.
8. T. Kosar and M. Livny, "Stork: Making Data Placement a First Class Citizen in the Grid," in Proc., 2008.
9. DMOVER: Scheduled Data Transfer for Distributed Computational Workflows, 2008.
10. Sherwood, R. Braud, and B. Bhattacharjee et al, "Slurpie: A Cooperative Bulk Data Transfer Protocol," in Proc.,2004.