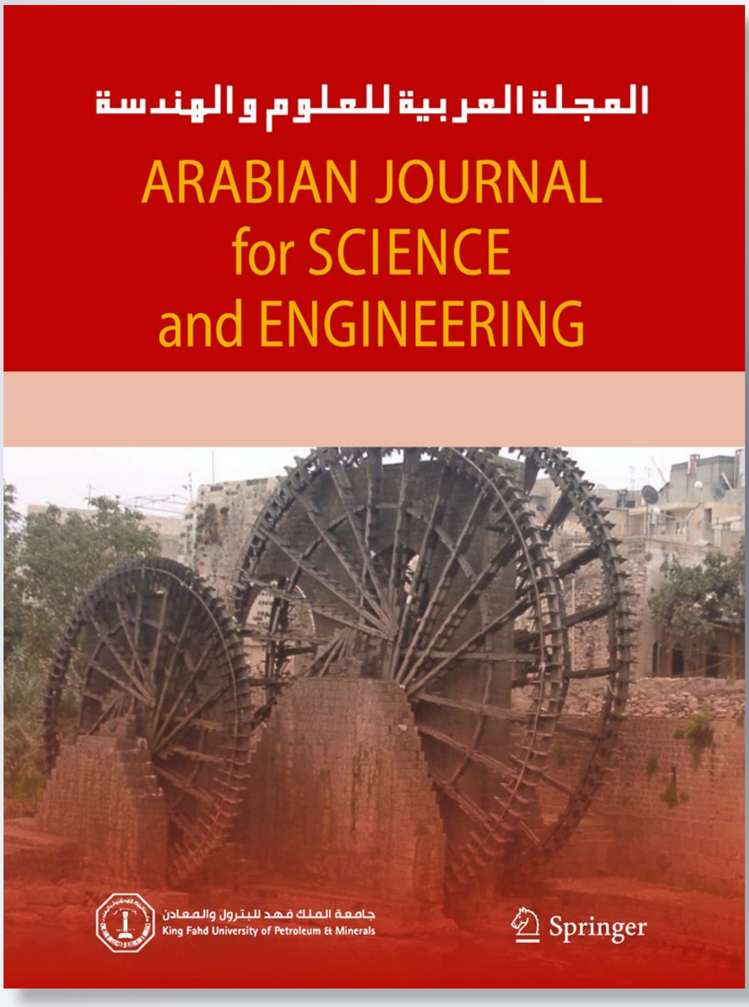
Parallel Data Migration Framework on Linux Clusters

Muhamed F. Mudawar & Mohammed K. AlGhuson

Arabian Journal for Science and Engineering

ISSN 1319-8025
Volume 36
Number 5

Arab J Sci Eng (2011) 36:785-794
DOI 10.1007/s13369-011-0073-5


المجلة العربية للعلوم والهندسة
ARABIAN JOURNAL for SCIENCE and ENGINEERING
جامعة الملك فهد للبترول والمعادن
King Fahd University of Petroleum & Minerals
Springer

Arab J Sci Eng (2011) 36:785–794
DOI 10.1007/s13369-011-0073-5

RESEARCH ARTICLE - COMPUTER ENGINEERING AND COMPUTER SCIENCE

Muhamed F. Mudawar · Mohammed K. AlGhuson

Parallel Data Migration Framework on Linux Clusters

Received: 7 October 2009 / Accepted: 18 March 2010 / Published online: 17 August 2011

Abstract A rapid growth in the storage capacity requirements at a computer center can lead to the installation of additional disk racks. The challenging task is not the installation, but to migrate old data to the new storage pools. A framework to parallelize the data migration process, using Linux clusters connected to Storage Area Network storage, is presented. A Linux tool to efficiently parallelize data migration, utilizing the High Performance Computing environment, is developed. Results show that using multiple nodes and multiple data copying streams per node achieves significant speedup factors over manual copying. The tool is demonstrated on four nodes using 178 data copying streams, achieving a speedup factor close to seven. The tool is scalable and capable of higher speedup factors with more available data moving nodes.

Keywords Parallel data migration · DMover tool · Linux cluster · Storage area network

الخلاصة

إن النمو السريع لحجم التخزين في مراكز الكمبيوتروالمعلومات أدى إلى زيادة عدد رفوف القرص الثابت التي يجري تثبيتها. والصعوبة ليست في تثبيت رفوف القرص الجديد، ولكن في ترحيل الكم الهائل من البيانات القديمة إلى مجمعات تخزين جديدة. وفي هذه الورقة، نقدم إطارا يستخدم مجمعات لينكس (Linux clusters) مرتبطة بشبكة تخزين (SAN storage) لتسريع عملية ترحيل البيانات بشكل متواز. وقد طوَرنا أداة لينكس (Linux tool) التي تستخدم بيئة الحوسبة عالية الأداء (High-Performance Computing) لترحيل البيانات بكفاءة. وأظهرت النتائج أن استخدام عقد متعددة (multiple nodes) وتيارات نسخ بيانات متعددة (multiple streams) لكل عقدة يحقق عوامل تسريع كبيرة على النسخ اليدوي. لقد أظهرنا استخدام الأداة في 4 عقد و 178 تيارا لنسخ البيانات وحققنا عامل تسريع بالقرب من 7. إن الأداة هي قابلة لتحقيق مستوى أعلى من عوامل تسريع نسخ البيانات اذا توفر عدد أكبر من العقد لنقل البيانات.

M. F. Mudawar
King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia
E-mail: mudawar@kfupm.edu.sa

M. K. AlGhuson (✉)
EXPEC Computer Center, Saudi Aramco, P.O. Box 10265, Dhahran 31311, Saudi Arabia
E-mail: mohammed.ghuson@aramco.com

1 Introduction

Recent years have seen a growing interest in optimizing the data migration process. With rapid increases in server memory and CPU speeds, greater processor and node interconnection is required. At Saudi Aramco, the world's largest oil producing company, in its Exploration and Petroleum Engineering Computer Center (EXPEC), conventional serial seismic processing (CSSP) is performed on Linux clusters to which storage area network (SAN) storage [1,2,13] is attached.

When technology changes significantly in the cluster or storage hardware [11,12,16], or when a new approach is found to process CSSP jobs, new hardware usually replaces old technology. The necessary migration of terabytes of seismic data to the new storage requires very efficient handling. Current industry practice is to migrate data serially, a poorly distributed methodology in which the data pool is manually divided into sets with those sets assigned to multiple machines. Alternatively, the migration is sometimes done by an IT company such as IBM, HP or Brocade, at a cost depending on the data size. In any case, migration involves special hardware, software and a blackout time on the resources involved [3,4].

This paper introduces a tool that works on the file and directory level to divide the data to be migrated into equally sized threads; each is scheduled independently and executed automatically on the resources assigned to it by a resource allocation system. It gives storage administrators the capability to migrate data very efficiently and without additional cost. The tool uses Linux along with the high performance computing (HPC) environment [14,15] as the basic building blocks.

2 Related Work

A data placement subsystem supporting data staging in distributed applications in a grid environment has been developed [8,10]. This system has its own scheduler to deal with the characteristics of data jobs, which are different from computational jobs at such environments. The system introduces additional components to the HPC environment which conflict with the requirement NOT to change the way the current cluster functions. Moreover, this system focuses on grid distributed computing in which data needed for a computational job could be anywhere on that system, possibly geographically remote. On the other hand, the focus of our work is data migration to a new storage system from a decommissioned one with different characteristics.

Two tools of interest work on distributed grid environments [6,7]. The first, a pilot project called the Grid Data Management Pilot (GDMP) [7], asynchronously replicates large object-oriented data stores over a wide-area network to globally distributed sites. The second, called Kangaroo [6], offers a highly available and highly reliable service by sacrificing some consistency guarantees. As stated in [6]: "Although this would be unacceptable for a general-purpose local file system, it is sensible for distributed data analysis."

Grid distributed environment and development tools address a challenging set of needs. It makes a lot of sense to efficiently gather scattered data to be processed. However, if the data is not spread around but is in storage devices a few meters apart, then such tools are not applicable. In addition, most of these tools have particular objectives which make them inappropriate for our work. Of course, one has the option of ready-made solutions provided by large enterprises that provide proprietary solutions with their own hardware, software and engineers to perform the data migration.

However, one grid data application is interesting for our problem: "GridFTP" [5,9]. This protocol provides secure, efficient data movement in grid environments by extending the standard FTP protocol. In addition to standard FTP, GridFTP supports features offered by the grid storage system in use. However, although GridFTP provides good file recovery capabilities, it does have some weaknesses. For example, GridFTP requires that the client remains active until a transfer finishes. This means that one cannot use the rich set of recovery features of GridFTP where the client state has been lost. In the event of a client state loss, transfer must restart. However, most of GridFTP's limitations should not affect the actual data coping functionality, so it is a worthwhile tool.

3 Environment Overview

This section provides an overview of the EXPEC SAN environment with all the involved hardware. The environment consists mainly of Linux clusters, the storage systems and the SAN network middleware interconnect.

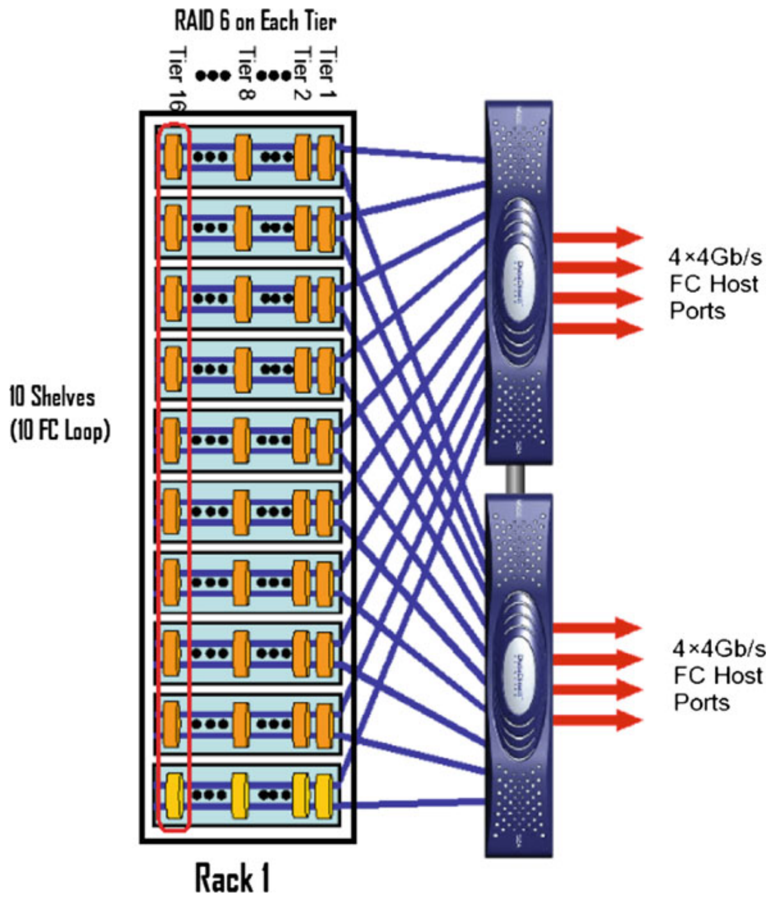


Fig. 1 IBM DCS9550 SAN storage

3.1 The Storage System

Figure 1 shows the SAN storage used in this work, comprising IBM DCS9550 racks. One storage rack contains 10 fiber channel (FC) shelves, each shelf accommodating up to 16 or 48 SATA/FC disks, depending on the shelf type, and a single or dual controller, which act as the gateway to all the other disk drives. The figure shows a rack with 10 shelves, 16 disks per shelf, and a dual controller. Each shelf has one 2 Gbps cable to each controller, which also has four 4 Gbps uplinks. The controllers provide cluster nodes access to the storage drives. This storage system design is such that the controllers are the bottleneck; with 10 shelves, the 20 Gbps will overwhelm a controller, which provides only 16 Gbps throughput to cluster nodes.

The first step in exporting the usable space from this massive number of disk drives on the IBM DCS9550 is to create tiers. Each tier is a collection of 10 disks, one from each shelf. Each tier is a redundant array of independent disks (RAID) group. The RAID created on top of each tier is RAID 6 or dual parity (DP) which can tolerate dual disk failures.

The smallest hardware unit that the controllers on this type of storage can deal with is a tier. However, the end system OS cannot interact with these tiers. As a result, one needs logical unit numbers (LUNs) on top of the tiers. The LUNs can be as small as a part of a single tier, or as large as 16 tiers. These LUNs get exported as regular attached SCSI disks which are presented to the end users as disks. After this, it is up to the end system users how to use these disks. They can create a local file system, or they can set up a distributed file system that spans multiple LUNs.

One well-known situation introduced with such a storage system is multi-pathing, meaning that a single LUN is seen by the end system OS multiple times. This is because of the multiplicity of uplinks used on the controllers. The advantage is that a single LUN can be accessed from different paths. However, the end system OS must be intelligent enough to recognize this issue. Otherwise, data corruption can occur. In ECC,

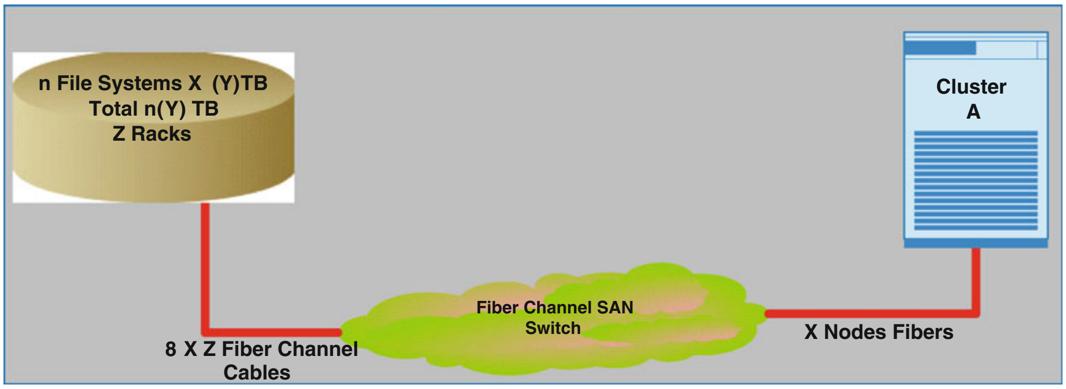


Fig. 2 ECC single-cluster SAN connectivity

the StoreNext FileSystem (SNFS) distributed file system is used. The storage system, shown in Fig. 3, consists of 76 storage racks, 2188 terabytes of disk storage and 32 file systems divided among five clusters.

3.2 Linux Clusters

The main computing resources are Linux clusters running the Red Hat OS and connected to SAN storage. A Linux cluster consists of rackable computers, called nodes, which have links collapsed into internal switches that provide connectivity among all the nodes and the uplink connections to the backbone's top-level switch. These nodes are typically divided into a master node and compute nodes. The master node runs the PBS resource management package and the Maui job scheduler. The compute nodes run the client software. Each node has an FC host bus adapter (HPA), which provides SAN and Ethernet connectivity.

3.3 SAN Connectivity

SAN switches provide interconnection between the nodes and the storage controllers. A SAN switch is a fiber channel protocol (FCP) switch that routes FC traffic. SAN switches have additional features like zoning, which masks a port from being seen from specific ports on the same fabric. Zoning limits the multi-pathing problem and isolates traffic on SAN switches.

The entire ECC SAN environment is used for seismic processing, with each process associated with a specific crew in the field. At any time, there can be a number of crews sending their seismic data for processing. Their data is independent, since it is coming from different fields. For this reason, there is no need to build a single SAN fabric. Instead, we have five separate SAN networks, each with a cluster of many nodes and storage of multiple racks, represented as multiple file systems. Figure 2 depicts the SAN connectivity to a single cluster.

3.4 Data Mover Nodes

Because of the separated SANs, there is no way for these clusters to communicate with each other. Consequently, there must be changes to this architecture for an efficient data migration process through the SAN switches. Because of this isolation and the requirement to use SAN connectivity to move data, data mover (DM) nodes were introduced, each with two dual-port fiber channel cards connected in a topology to cover each SAN network; see Fig. 3.

Each of the four DMs has four fiber channel connections; one is connected to the cluster A SAN switch, with the remaining three connected to the other SAN switches for load balancing and availability. Cluster A is connected to the storage system where the data is to be copied to. As a result, to migrate data from any storage to the new storage system the maximum available bandwidth is 4×4 or 16 Gbps.

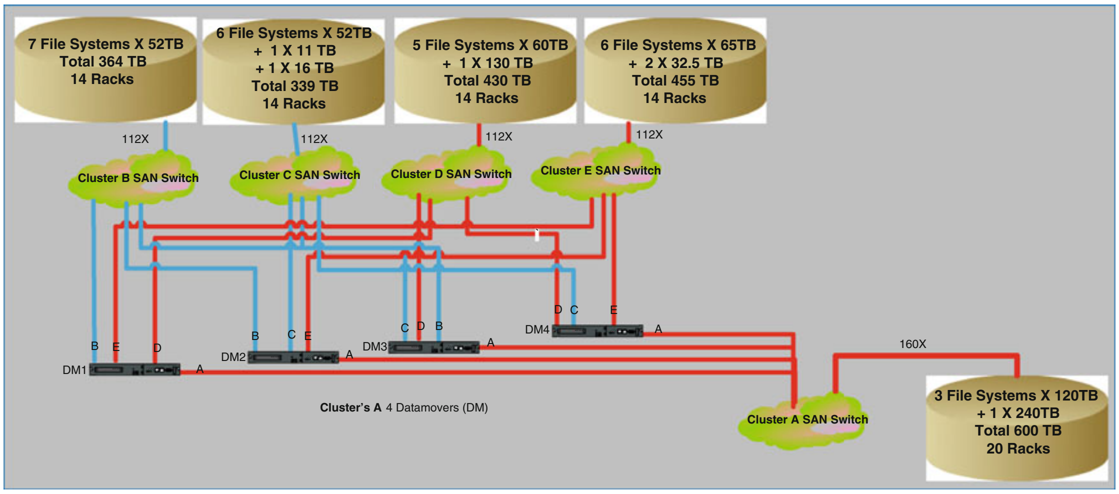


Fig. 3 Data mover nodes and their interconnection topology

4 The Data Mover Tool Architecture

The data mover tool, called DMover, is built on top of existing Linux OS tools, cluster resource allocation software, cluster job scheduling software and the *korn* shell scripting language. The underlying tool used for data transfer is the well known Linux/UNIX *rsync* tool. The two other tools are the portable batching system (PBS) package, responsible for cluster resource allocation and management, and Maui, an advanced scheduler for scheduling user jobs on the allocated resources based on particular policies. The policy used for DMover is to assign the first job to the first available node. Figure 4 shows the flowchart and the interactions of these tools in DMover.

4.1 Configurations

The DMover tool accepts a command line with a source, destination, mode, size and number of threads as parameters. It configures the Linux OS, PBS and Maui packages to assure a successful run of jobs. For a successful interaction between the master and the compute nodes, a trust key is used to avoid repeated password requests. The PBS package contains two main executables: the *pbs_server*, which runs on the master node, and the *pbs_mom*, which runs on all other compute nodes. A configuration file called *nodes* contains a list of hostnames for the compute nodes to be made available to the *pbs_server*, the number of processes that can be hosted by each hostname and the type of each compute node. The Maui package is an open source job scheduler used on clusters and supercomputers. It is capable of supporting an array of scheduling policies, priorities, reservations, and fair share capabilities, by which a job can be scheduled, queued and executed. It runs only on the master node. The main file to be configured is *maui.conf*. The *qmgr* command is a binary provided by the Maui package to configure a queue and assign available resources.

4.2 Thread Creation

DMover divides the source data into equal-sized streams to load balance the cluster nodes. It works by traversing a given source directory and, according to the mode of operation, builds threads which are submitted to the cluster's scheduler system. Figure 5 shows how DMover creates the threads, which are submitted as regular jobs to the clusters. For example, consider the command "DMover /root/ /tmp/ 2 10" which specifies /root/ as the source directory, /tmp/ as the destination, 2 as the mode and 10 as the number of threads. The first step, as shown in the figure, checks the operating mode, which is 2. It then checks the thread threshold, the maximum amount of data a thread can handle, by dividing the /root/ size by 10. It builds a list with the contents of the /root/ directory along with their sizes, grips the first entry and checks its size—whether it is greater than the threshold or not. If it is greater and this entry is a file, it creates a single large thread job and increments

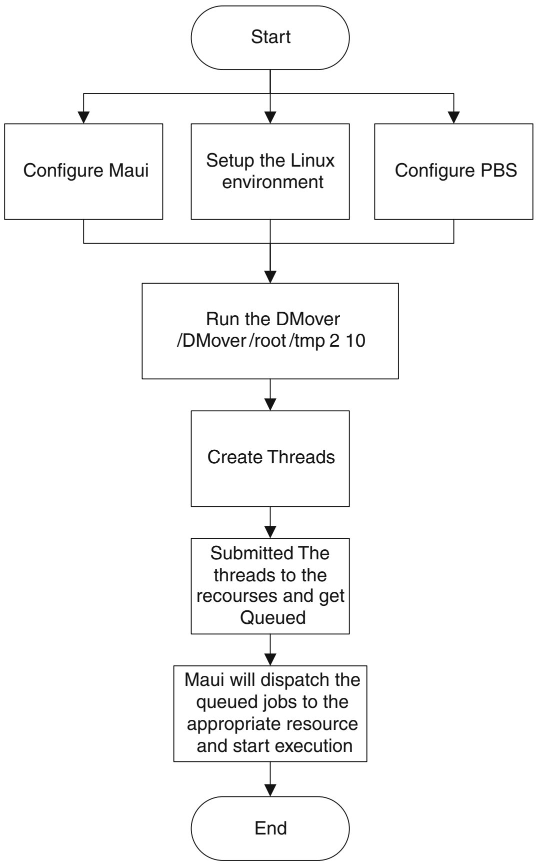

Start
Configure Maui
Setup the Linux environment
Configure PBS
Run the DMover /DMover /root /tmp 2 10
Create Threads
Submitted The threads to the recourses and get Queued
Maui will dispatch the queued jobs to the appropriate resource and start execution
End

Fig. 4 The data mover tool architecture

`Large Threads` by one. On the other hand, if this entry is a directory it appends the list with its contents along with their sizes. If this entry is not greater than the threshold it checks the `Current thread size` + this `entry size`. If the total size is less than the threshold, it appends the current thread and goes back to the list to pop the next entry. On the other hand, if the total is greater than the threshold it pushes this entry on top of the list, closes the current thread, increments `Threads`, pops the top entry in the list and adds it to a new thread.

This process continues until the list is empty. At that time, DMover finishes thread creation and begins submitting these threads to the master node as regular jobs where they are assigned resources. Then, the actual data migration begins.

5 Experiments and Performance Measurement

To test DMover, dump files were migrated among different file systems. The *lmdd* command was used to create the files as a representative sample of the seismic data files. The file sizes followed a normal distribution, ranged from 1 KB to 10 GB per file, with a total size of 4 TB for 67,500 files.

We used four nodes to perform the data migration with each node running multiple data copying threads. Three factors affect the migration time: the number of nodes participating, the number of threads running on a node and the number and size of the files transferred by each thread.

Figure 6 shows the results of five experiments. The first column shows the result of manual copying, which took 247 minutes to migrate 4 TB of data from source to destination. The second column was obtained using DMover on a single node running one process and one thread. Comparing the first two columns, it is clear that DMover does not yield any improvement in the case of a single node and a single thread. In fact, DMover required an initial setup overhead and took longer at 259 minutes. However, the results improve if multiple copying threads are involved.

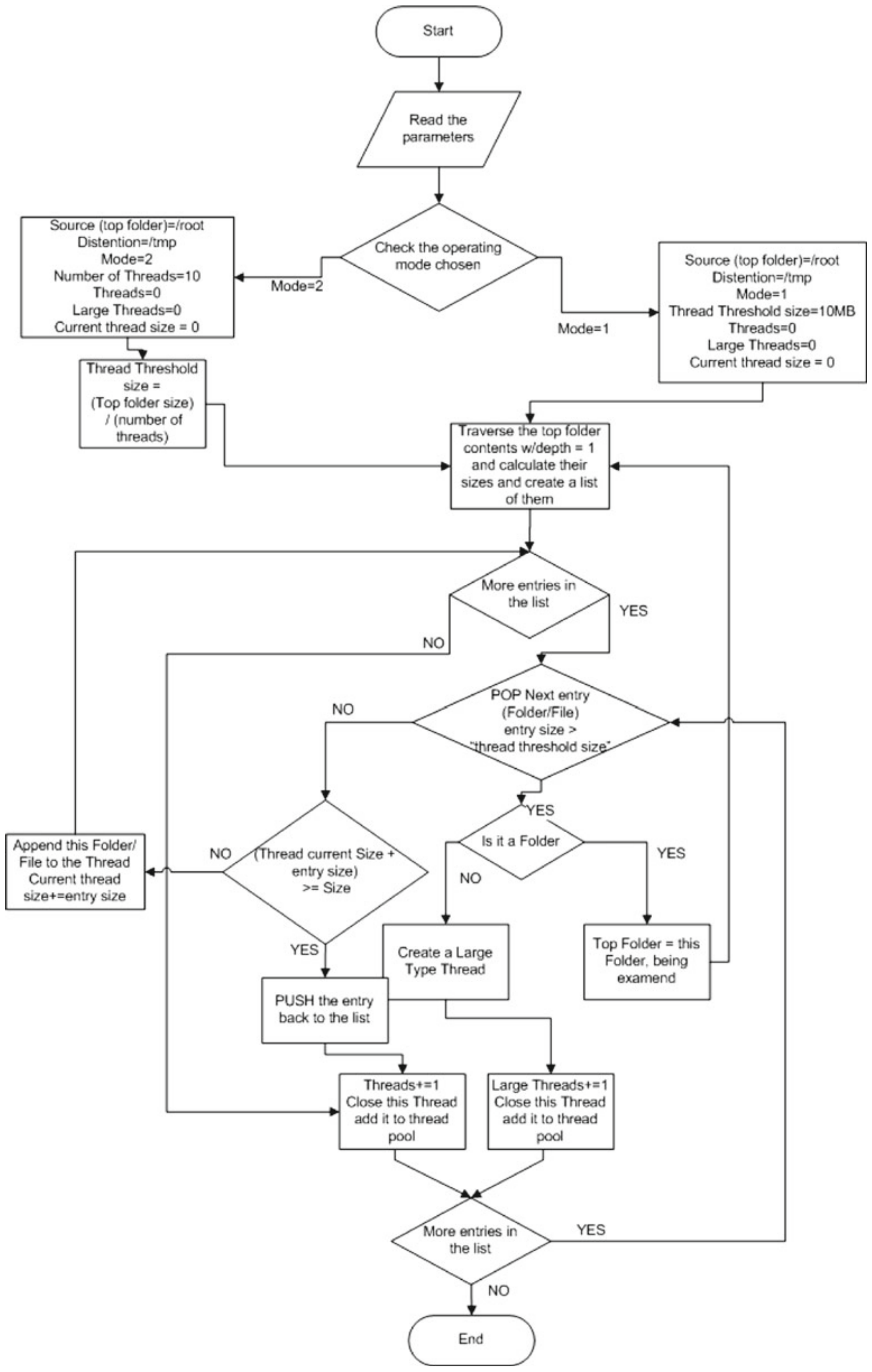

Start
Read the parameters
Check the operating mode chosen
Mode=2
Mode=1
Source (top folder)=/root
Distention=/tmp
Mode=2
Number of Threads=10
Threads=0
Large Threads=0
Current thread size = 0
Source (top folder)=/root
Distention=/tmp
Mode=1
Thread Threshold size=10MB
Threads=0
Large Threads=0
Current thread size = 0
Thread Threshold size = (Top folder size) / (number of threads)
Traverse the top folder contents w/depth = 1 and calculate their sizes and create a list of them
More entries in the list
YES
NO
POP Next entry (Folder/File) entry size > "thread threshold size"
NO
YES
Is it a Folder
YES
NO
(Thread current Size + entry size) >= Size
NO
YES
Append this Folder/ File to the Thread Current thread size+=entry size
Create a Large Type Thread
Top Folder = this Folder, being examend
PUSH the entry back to the list
Threads+=1 Close this Thread add it to thread pool
Large Threads+=1 Close this Thread add it to thread pool
More entries in the list
YES
NO
End

Fig. 5 Thread creation algorithm

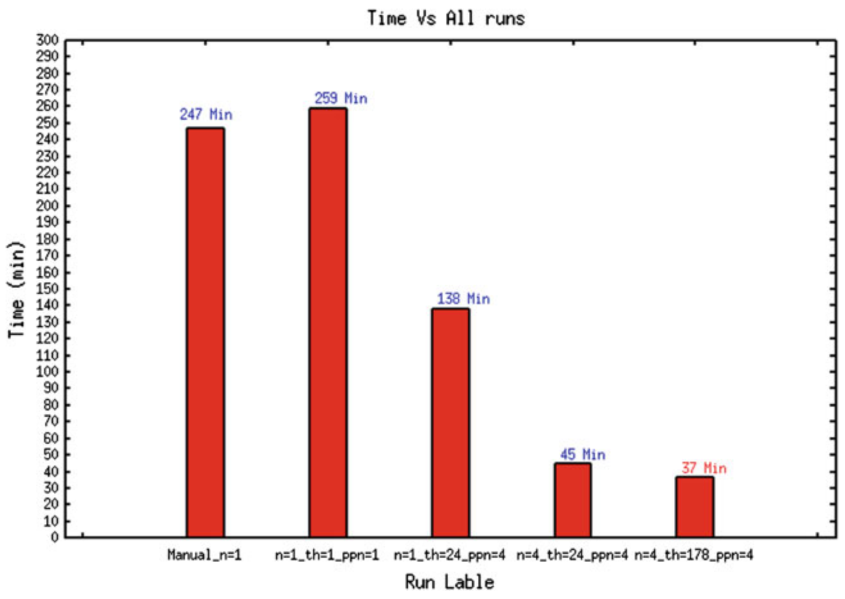


Fig. 6 Results of five data migration experiments

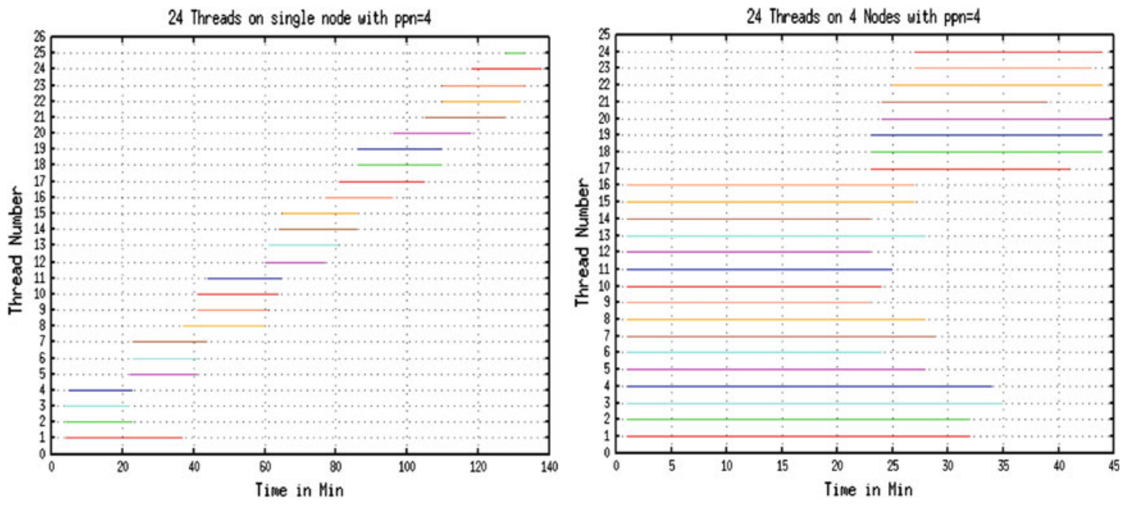


Fig. 7 One versus four nodes executing 24 threads with four threads per node at one time

The third column in Fig. 6 shows the results for multithreading and multiple processes per node. The same set of data files was divided into 24 threads executing on the same node. Comparing the second and third columns, the same migration job was finished 121 minutes earlier and ran 1.88 times faster because of the multiple copying streams.

It should be clear that the actual streaming does not start from minute zero since DMover must traverse the source data files and build the threads. This time is pure overhead since all resources are idle during the setup time.

Migration time can be improved further with more nodes. The fourth column in Fig. 6 shows the execution with 24 threads on four nodes and with four threads per node at one time. The time is reduced from 138 to 45 min. Figure 7 shows the timing and the progress of the 24 threads on one node as well as on four nodes.

The time to copy 4 TB of data was reduced from 138 to 45 min when the 24 threads ran on four nodes rather than one. Ideally, the time should have decreased to 36.75 min, given that the DMover setup overhead was estimated at 3 min:

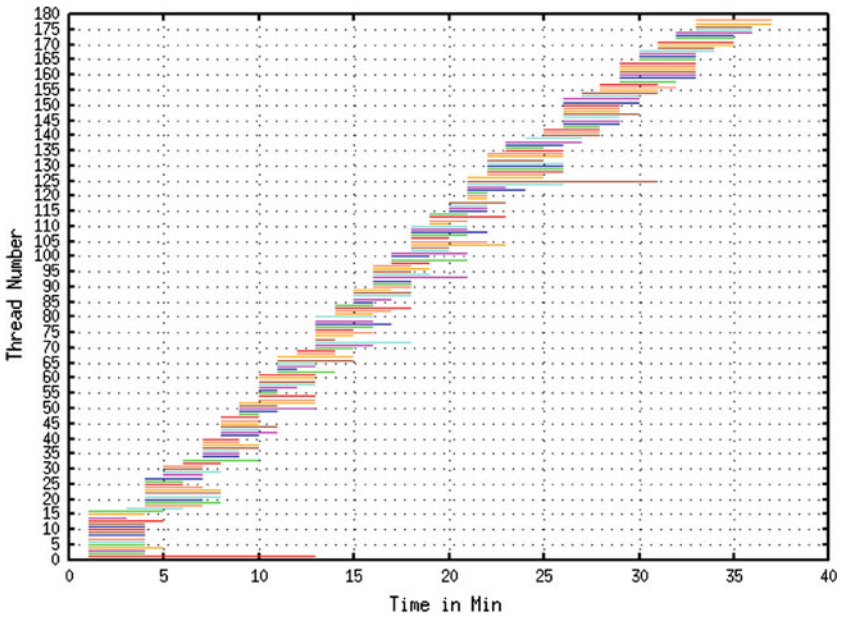


Fig. 8 Four nodes executing 178 threads with four threads per node at one time

$$\frac{138\,\text{min} - DMover\_Overhead\ (3\,\text{min})}{Nodes(4)} + DMover\_Overhead\ (3\,\text{min}) = 36.75\,\text{min} \qquad (1)$$

As clearly shown in Fig. 7, the threads and nodes are not balanced. Some nodes took longer to execute their threads while others were idle and executed fewer threads. The resource utilization and load balancing can be improved if we increase the number of threads and divide the source data into smaller threads. This keeps the nodes busier toward the end of the migration process. Figure 8 shows the effect of increasing the number of threads to 178 on four nodes, which reduces the time to 37 min, as shown in the fifth column of Fig. 6.

6 Discussion and Conclusion

Using DMover on four nodes, without human interaction, achieved a speedup factor of 247/37 = 6.68 over manual copying. Increasing the number of threads from 24 to 178 balanced the load and improved the utilization of resources. In the experiment, only four nodes were configured for data movement. For better performance, more data moving nodes are necessary. However, such a change in the HPC environment is not always feasible at data centers such as EXPEC.

Looking closer at Figs. 7 and 8, some threads took longer than others. Analysis of those threads makes clear that the number of files matters. So it is not only the size of the data for which each thread is responsible, but also the number of files copied that must be considered. To obtain a smooth graph, there must be a way to apportion the large files onto threads to maintain a load balance. A more complex model is necessary to split large files into pieces. The approach of file splitting is worth investigating. Of course, the benefit of splitting/merging large files should be weighed against the associated overhead.

References

1. Hulen, H.; Graf, O.; Fitzgerald, K.; Watson, R.: Storage Area Networks and the high performance storage system. In: Nineteenth IEEE Symposium on Mass Storage Systems (2002)
2. Eisler, M.; Corbett, P.; Kazar M.; Nydick, D.S.: Data ONTAP GX: A scalable storage cluster. In: Proceedings of the 5th USENIX Conference on File and Storage Systems (FAST'07), San Jose, CA (2007)
3. Golubchik, L.; Khuller, S.; Kim, Y.; Shargorodskaya, S.; Wan, Y-C.: Data migration on parallel disks. In: Proceedings of European Symposium on Algorithms (2004), pp 689–701. LNCS 3221, Springer (2004)
4. Zissimos, A.; Doka, K.; Chazapis, A.; Koziris, N.: GridTorrent: Optimizing data transfers in the Grid with collaborative sharing. In: Proceedings of the 11th Panhellenic Conference on Informatics (PCI2007). Patras, Greece, May 2007

5. Allcock, W.; Bresnahan, J.; Kettimuthu, R.; Link, M.; Dumitrescu, C.; Raicu, I.; Foster, I.: The Globus striped GridFTP framework and server. In: Proceedings of Super Computing 2005 (SC05), November 2005
6. Thain, D.; Basney, J.; Son, S.C.; Livny, M.: The Kangaroo approach to data movement on the grid. In: Proceedings of Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10), San Francisco, California, August 7–9 2001
7. Samar, A.; Stockinger, H.: Grid data management pilot (GDMP): a tool for wide area replication. In: Proceedings of IASTED International Conference on Applied Informatics (AI2001), Innsbruck, Austria, February 19–22, 2001
8. Kosar, T.; Livny, M.: A framework for reliable and efficient data placement in distributed computing systems. J. Parallel and Distrib. Comput. **65**, 1146–1157 (2005)
9. Lim,S.; Fox, G.; Kaplan, A.; Pallickara, S.; Pierce, M.: GridFTP and parallel TCP support in naradabrokering. In: Proceedings of International Conference on Algorithms and Architectures for Parallel Processing, pp. 93–102 (2005)
10. Chen, L.; Zhu, Q.; Agrawal, G.: Supporting dynamic migration in tightly coupled grid applications. In: Proceedings of ACM/IEEE SC 2006 Conference, November 2006
11. Cao, M.; Tso, T.; Pulavarty, B.; Bhattacharya, S.; Dilger, A.; Thomas, A.: State of the art: where we are with the Ext3 filesystem. In: Proceedings of the 2005 Ottawa Linux Symposium, 2005
12. Eisler, M.; Corbett, P.; Kazar, M.; Nydick, D.S.; Wagner, J.C.: Data ONTAP GX: a Scalable storage cluster. In: Proceedings of FAST' 07, 2007
13. Clark, T.: Designing storage area networks: a practical reference for implementing fibre channel and IP SANs, Addison-Wesley Professional, Reading (2003)
14. Schmuck, F.; Haskin, R.: GPFS: A shared-disk file system for large computing clusters. In: Proceedings of the First Conference on File and Storage Technologies (FAST), pp. 231–244, January 2002
15. Shepard, L.; Eppe, E.: SGI InfiniteStorage Shares Filesystem CXFS: A high-performance, multi-OS filesystem from SGI, Technical Report, Silicon Graphics, 2006
16. Carns, P.; Ligon III, W.B.; Rajeev, R.B.; Thakur.: PVFS: A Parallel File System for Linux Clusters. In: Proceedings of the 4th Annual Linux Showcase and Conference, pp. 317–327 (2000)