IOWA STATE UNIVERSITY
Digital Repository

Electrical and Computer Engineering Conference Papers, Posters and Presentations

Electrical and Computer Engineering

2-2-2019

LDM: Lineage-Aware Data Management in Multi-tier Storage Systems

Pratik Mishra
Iowa State University

Arun K. Somani
Iowa State University, arun@iastate.edu

Follow this and additional works at: https://lib.dr.iastate.edu/ece_conf

Part of the Electrical and Computer Engineering Commons

Recommended Citation

Mishra, Pratik and Somani, Arun K., "LDM: Lineage-Aware Data Management in Multi-tier Storage Systems" (2019). *Electrical and Computer Engineering Conference Papers, Posters and Presentations*. 66.
https://lib.dr.iastate.edu/ece_conf/66

This Conference Proceeding is brought to you for free and open access by the Electrical and Computer Engineering at Iowa State University Digital Repository. It has been accepted for inclusion in Electrical and Computer Engineering Conference Papers, Posters and Presentations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

المنارة للاستشارات

www.manaraa.com

LDM: Lineage-Aware Data Management in Multi-tier Storage Systems

Abstract

We design and develop LDM, a novel data management solution to cater the needs of applications exhibiting the *lineage* property, i.e. *in which the current writes are future reads*. In such a class of applications, slow writes significantly hurt the over-all performance of jobs, i.e. current writes determine the fate of next reads. We believe that in a large scale shared production cluster, the issues associated due to data management can be mitigated at a way higher layer in the hierarchy of the I/O path, even before requests to data access are made. Contrary to the current solutions to data management which are mostly reactive and/or based on heuristics, LDM is both deterministic and pro-active. We develop *block-graphs*, which enable LDM to capture the complete time-based data-task dependency associations, therefore use it to perform life-cycle management through tiering of data blocks. LDM amalgamates the information from the entire data center ecosystem, right from the application code, to file system mappings, the compute and storage devices topology, etc. to make oracle-like deterministic data management decisions. With trace-driven experiments, LDM is able to achieve 29–52% reduction in over-all data center workload execution time. Moreover, by deploying LDM with extensive pre-processing creates efficient data consumption pipelines, which also reduces write and read delays significantly.

Keywords

Lineage, Storage, Hadoop, Hard Disk Drives HDD, Solid State Drives SSD, Data management

Disciplines

Electrical and Computer Engineering

Comments

This is a post-peer-review, pre-copyedit version of a conference proceeding published as Mishra, Pratik and Arun K. Somani. (2020) "LDM: Lineage-Aware Data Management in Multi-tier Storage Systems." In Arai K., Bhatia R. (eds) *Advances in Information and Communication. FICC 2019. Lecture Notes in Networks and Systems*, vol. 69. Springer, Cham. The final authenticated version is available online at DOI: 10.1007/978-3-030-12388-8_48. Posted with permission.

This conference proceeding is available at Iowa State University Digital Repository: https://lib.dr.iastate.edu/ece_conf/66

المنارة للاستشارات

www.manaraa.com

LDM: Lineage-aware Data Management in multi-tier storage systems

Pratik Mishra and Arun K. Somani

Department of Electrical and Computer Engineering,
Iowa State University, Ames IA 50014, USA,
mishrap@alumni.iastate.edu, arun@iastate.edu.

Abstract. We design and develop **LDM**, a novel data management solution to cater the needs of applications exhibiting the *lineage* property, i.e. *in which the current writes are future reads.* In such a class of applications, slow writes significantly hurt the over-all performance of jobs, i.e. current writes determine the fate of next reads. We believe that in a large scale shared production cluster, the issues associated due to data management can be mitigated at a way higher layer in the hierarchy of the I/O path, even before requests to data access are made. Contrary to the current solutions to data management which are mostly reactive and/or based on heuristics, **LDM** is both deterministic and pro-active. We develop *block-graphs*, which enable LDM to capture the complete time-based data-task dependency associations, therefore use it to perform life-cycle management through tiering of data blocks. LDM amalgamates the information from the entire data center ecosystem, right from the application code, to file system mappings, the compute and storage devices topology, etc. to make oracle-like deterministic data management decisions. With trace-driven experiments, LDM is able to achieve 29% to 52% reduction in over-all data center workload execution time. Moreover, by deploying LDM with extensive pre-processing creates efficient data consumption pipelines, which also reduces write and read delays significantly.

Keywords: lineage, storage, Hadoop, Hard Disk Drives HDD, Solid State Drives SSD, data management.

1 Introduction

Extracting high performance from the storage system is the most important challenge in designing computing systems today. In large data intensive applications, the movement of data from and to storage to compute engine may overshadow the processing time for data [1] [2]. The storage devices attached directly (or locally) to the compute nodes have limited capacity and are expensive due to their proximity. Therefore, data is typically stored in storage hierarchy and are required to be moved over the network to the compute nodes for processing. A higher volume of data movement over I/O channels is resource (memory, network, and storage), time and energy intensive. Overall, this scenario makes data

للاستشارات المنارة

www.manaraa.com

storage and management in data centers a challenge as it has direct impact on the efficiency of computing.

Data centers today cater to a wide diaspora of applications which process multiple data sets for multiple jobs in a multi-user environment concurrently. They also deploy storage systems organized in multiple heterogeneous tiers, which is necessary to achieve cost-performance-capacity trade-off [3] [4] [5]. Dedicating physical resources for every application is not economically feasible. Resource sharing causes contention affecting the efficiency and performance [3] [6] [7]. Despite advanced optimizations applied across the various layers along the odyssey of data access, the data management layer remains volatile [8] [9]. Data are scattered over multiple files located at multiple storage nodes[1] and replicated for performance, availability and reliability reasons.

An ideal storage system should deliver the same read (or write) access performance to all applications. The read and write performance may differ due to their perceived implications on the application performance. However, realizing same read (or write) access performance for all applications can be difficult to achieve because the data access time depends on a variety of factors including physical device characteristics, data locations, current utilization of devices, available I/O bandwidth, location of storage device, network topology, and delays, etc.

The current data management techniques fail to capture the syntax and semantics of jobs and the associations of data in various stages of jobs. Moreover, the goals of current efforts have been to make read operations faster as they are believed to be the biggest bottleneck. However, inconsiderate placement of intermediate results (writes) for reuse may affect the read performance adversely. Under this scenario, the gains derived by deploying multiple tiers in storage can be nullified easily by improper replica allocations to tiers, handling of memory resources, and avoidable data movement [4] [10] [11].

Here, *we address the issue of how to deliver ideal system performance for all read and write accesses.*

We understand that it can be and is hard to manage data storage and movement for any arbitrary set of applications contributing to data center workloads. Therefore, to begin with, we limit our scope to a set of applications that depict the lineage property, where intermediate data during computation must be written and read back later on. Such a scenario occurs commonly in data centers. For example, in one application scenario [11], data may be extracted using MapReduce [12] which are queried using Pig [13], then machine learning algorithms are used on the queried results [11], and are finally combined with other similar results to produce the final answers [11] [14]. The issues associated with data management gets amplified for applications with such chained jobs, which exhibit lineage. Therefore, dealing with large amounts of current "writes", which are future "reads" is equally important to achieve good performance [11] [15]. Multi-tier storage offers multiple dimensions, such as device type, network

[1] Storage refers to the overall data plane, whereas a storage node refers to a single physical device.

المنارة للاستشارات
www.manaraa.com

connectivity, and replication management, allowing exploration of data access solution space differently.

1.1 Motivation

The following trends are clear:
1. The size of data is ever increasing, and more and more data are being stored on remote storage.
2. The complexity and structure of data being processed for analytics varies dramatically.
3. Enterprise data centers includes thousands of processing (and heterogeneous) and storage nodes.
4. Data is organized in multiple heterogeneous tiers with a wide variety of storage devices in both local and remote storage to extract best performance.
5. Increasing amount of data are being used by multiple applications and/or series of jobs of the same application chained together.
6. Chained MapReduce ETL pipelines and Oozie workflows are most popular lineage based applications.
7. Current writes are future reads. Thus, writes dominate the over-all performance of these applications. Therefore, more emphasis needs to be given to initial data placement and replica management.
8. Data management needs to be both ecosystem as well as network-storage architecture-aware.

We develop and design a novel framework, called **LDM**, to address the challenges in lineage-aware data management to effectively utilize multi-tier storage hierarchy. LDM captures the inherent lineage information (using **block-graphs**) and reduce the data movement via network by placing them appropriately to enable maximal processing nearer to the storage locations as well as in appropriate storage tiers. Moreover, LDM utilizes all tiers[2] of storage to reduce data access delays in conjunction with workload aware tiering[3] by orchestrating multiple data management features. These include data placement, data replication management and data migration. In this paper, we limit our scope to data placement and data replication management, and in future would develop data migration capabilities. We believe LDM will have a huge impact on the performance and resource management of data processing platforms.

The paper is organized as follows. First, we discuss *lineage*, and the problems associated due to over-looking of these class of applications in Section 1. The related research is described in Section 2. In Section 3, we discuss the design of our data management framework, **LDM**, and the development of *Block-graphs* required to capture data dependability in workflows. This is followed by designing techniques for life-cycle management of data blocks utilizing various tiers of

[2] Storage media across all nodes with similar I/O characteristics form a tier.
[3] Tiering refers to orchestrating data between heterogeneous tiers of storage by leveraging individual strengths of each to maintain balance between Cost, Performance and Capacity.

المنارة للاستشارات
www.manaraa.com

storage. We evaluate the performance of LDM for two types of Data Center lineage-based applications along-with discussions of the results in Section 4. We conclude the paper in Section 5 with a discussion on future work.

2 Related Work

The concepts of lineage have been extensively used in databases mostly for recovery, space-optimizations, uncertainty, dependency and fault-tolerance, [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27]. There have been efforts [11][10] to understand lineage for in-memory computation for improving job recovery time in-case of fail-overs and performance in Data Centers with nodes having large memory. Zaharia et. al [10] forms distributed data-sets for in-memory computations (production and computation in-memory), which inherently improves performance. Tachyon[11] proposes an in-memory fault tolerant mechanism which leverages lineage to recover lost outputs by re-executing the steps which formed the data-sets. In-memory computations and storing of results in memory are infeasible for Big Data workloads as the working sets are huge to fit in RAM, along-with the time-varying nature of applications for production and consumption of data blocks [28]. Issues such as ensuring reliability and cost-effectiveness are other major challenges in such frameworks. Therefore, cost simulations in FBMsgs [28] that adding small SCM tier and efficient orchestrating data between tiers can lead to enhanced performance than equivalent spending on RAM or disks. Multi-tier storage offers multiple dimensions, such as device type, network connectivity, and replication management, which allows to explore to explore the issues associated with data access differently.

Most studies have focused on studying data center operations to consolidate the computing needs and organize and optimize computing for multiple applications. Computing resources are believed to be abundant, but without appropriate attention, they are mostly waiting for data and wasting cycles [3] [8] [9] [29] [14] [30]. Moreover, for lineage based applications, the impact is more severe due to data-dependency between tasks. Keeping all the data in memory (as done in Spark) may not be a wise choice either. We believe that the focus needs to shift from computing to data. What makes this shift relevant is the availability of oracle-like deterministic workload and data center storage topology aware data management. Datum access from storage and copying in memory is expensive. Therefore, we believe that studying data utilization patterns and developing strategies to optimize computing paths are the greatest needs at the current time [31].

Triple-H [32] designs a heterogeneous storage engine for HPC including RAM-disks, SSDs and HDDs, and Lustre FS to benefit HDFS. The data placement engine in [32] deals with tri-replication of blocks to ensure fault tolerance and the decisions of placement of replicas in a tier is based on storage space available with a usage-priority based tier migration model. hatS [33] proposes a model with an intent to remove performance bottlenecks by placing every block belonging to file in all tiers of storage. Multiple solutions [34] [35] [36] [33] [37] [38] [39]

المنارة للاستشارات
www.manaraa.com

[40] [41] [15] [4] [32] [35] have been proposed in literature to exploit multi-tier storage, but none addresses the issues associated with lineage or chained jobs. The storage layer is agnostic to the semantics of tasks on data and its execution characteristics.

To address these challenges, LDM utilizes the inherent lineage information (data-task associations) coupled with multi-tier storage for improving over-all application execution time. In the next section, we discuss the working of LDM and how it manages data-dependency (or *lineage*).

3 LDM

In large scale distributed systems, data management plays a vital role in processing and storing primary and backups of data across storage devices. Despite advanced optimizations being applied across various layers along the odyssey of data access, the data management layer still remains volatile [9] [8]. Goals of current efforts are to make read operations faster as they are believed as the biggest bottleneck. However, inconsiderate placement of intermediate results for reuse may affect performance adversely. This problem gets amplified for chained applications which exhibit lineage. It is now becoming clearer that dealing with large amounts of current "writes", which are future "reads" is equally important to achieve good performance. Therefore, in such data processing pipelines, its imperative to capture lineage or relationship across tasks and their dependency with data, i.e. data-task associations.

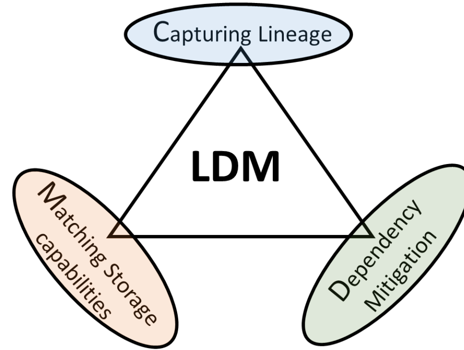


Fig. 1: Components of LDM.

In LDM, we provide a uniform execution environment across the storage server and compute server. We address the specific needs of a cluster of applications with data-dependencies. LDM resides in the Master (or Head) node of clusters where jobs are submitted by applications and data management decisions are made. Consider Figure 1, LDM includes the following three components.

المنارة للاستشارات
www.manaraa.com

1) Capturing Lineage Information. This component is accomplished by a set of APIs that generate and analyze the metadata associated with application code and extract semantic knowledge of the computational workflow and logic from the code to build task and block graphs (described later).

2) Matching Storage capabilities. This component uses the information about the storage devices (type, capacity, performance, etc.) as well as location (local or remote) to categorize and classify them. These APIs reside on DataNodes and storage servers and transmit storage devices information as part of the status updates regularly to the NameNode for its own use in making data location decisions.

3) Dependency Mitigation. This component use the above two modules assisted by the information stored in the Distributed file system to make data management decision and policies. These include APIs for initial Data Placement and Replica Management to decide where to place the date and its copies in terms of tier and device. Data Migration API will use the lineage information to evict already placed blocks as well as determines the utility of the blocks for both capacity and efficient utilization of storage tiers.

Before describing these components in detail, we first describe Hadoop and MapReduce ecosystem in brief as this will be used as a running example to demonstrate the need and our approach to solution.

Hadoop Ecosystem and MapReduce

Hadoop and its data processing framework - MapReduce is the de-facto large data processing framework for Big Data [13]. Hadoop is a multi-tasking system which can process multiple data sets for multiple jobs in a multi-user environment across multiple machines at the same time [42] [43]. Each MapReduce job consists of multiple processes submitting I/Os concurrently for Map, Shuffle and Reduce stages, each having skewed I/O requirements [44] [45] [46]. Hadoop Distributed File System (HDFS) uses a block-structured file system to deliver reliable storage [13] [43].YARN (Yet Another Resource Negotiator) is used for per-application based resource negotiating agent and is a centralized platform to ensure consistency and data manageability. YARN has enabled Hadoop with the flexibility to encompass multiple data processing engines such as Spark, Storm, etc. to process and manage the data concurrently. It has also enabled Hadoop with multi-tenant processing capabilities such as different applications/ processing engines working concurrently by using application based containers.

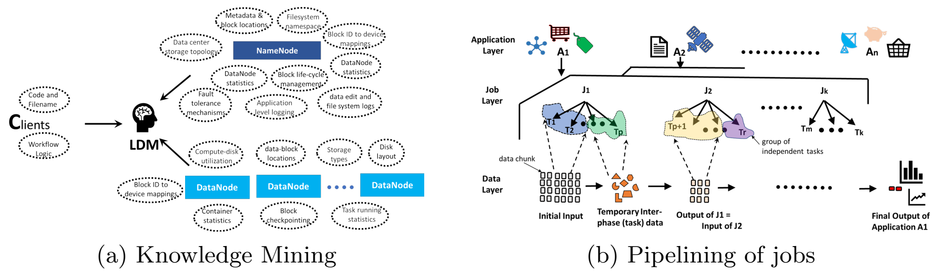
HDFS splits the files into fixed size file system blocks (64/128 MB), known as chunks, which is typically tri-replicated for achieving the fault-tolerance, availability and performance parameters. HDFS follows a leader-follower architecture, with a NameNode, which manages storage and several DataNodes hosting the data [43] [13]. The NameNode manages the file system namespace and associated metadata (file-to-chunk maps) as well as contracts the access to files by clients (once brokered, the clients interact directly with DataNodes). The NameNode operates entirely in memory, persisting its state to disk. All such information are persisted in two major files in the NameNode: 1) *fsimage*, which stores the

المنارة للاستشارات
www.manaraa.com

complete snapshot of the file system metadata at a particular instant; and, 2) *edit log* for the incremental changes to the file system namespace [43]. Thus, the NameNode is central to data management and can be exploited in developing necessary policies.

3.1 LDM framework

The key idea behind LDM is to build a data management system which has an oracle-like capability to know the future usage of data and therefore take deterministic actions based on its knowledge. The information of the data processing framework and ecosystem is already present and needs to be properly harnessed. All such information and statistics are already being produced or consumed by different components of the system. Therefore, this knowledge when amalgamated with our tier-and-network aware storage policies should yield better performance for lineage class of applications.



(a) Knowledge Mining (b) Pipelining of jobs

Fig. 2: (a) LDM Knowledge mining to aid data management policies; (b) Job pipelining and data-task associations.

In Hadoop like environments, LDM would work with YARN resource manager and HDFS data management to transform information into intelligence and use the intelligence acquired to execute policies for to mitigate the impact of data dependency for lineage class of applications. Consider Figure 2a, which briefly describe the information produced by different system components. LDM can use this information to create data management policies to achieve better performance for applications exhibiting lineage.

In the following three sections, we describe the details of the working components of LDM.

Capturing Lineage Information

It is imperative to understand and extract information about the workflow and the dependencies within and among the applications. The semantics and syntax of every application can be extracted by mining the code and amalgamated with the data processing steps to understand the ecosystem and achieve

المنارة للاستشارات
www.manaraa.com

efficiency in the computation. We achieve this functionality by developing a set of APIs. The Client API will understand the computation flow and build task graphs based by mining the application code. A task graph will be represented by a DAG (directed acyclic graph) representing the tasks as nodes and associated dependencies between tasks as directed edges.A task graph alone does not exhibit the location based data dependency. LDM, therefore, will use the file-system information about the block to device mappings (for example: fsimage, and editlog for HDFS) to associate blocks of data with tasks (using task blocks) to develop ***Block graphs***. Block graphs capture all the data block-task associations, which would deterministically capture data lineage across tasks. This knowledge would aid in mitigating the impact of delays associated to writing and then subsequently reading intermediate results. The interaction between the Client API for task graphs and filesystem namespace is achieved by the Data block-Task Associativity API working on the NameNode.

Understanding Ecosystem: Job Pipelines

Chained Jobs are a popular class of applications that are executed on clusters. Essentially, the jobs are pipelined and the output of a job forms the input (or a part of the input) of the next job. Such jobs are common in several business and scientific applications. For example, Job pipelines are produced by Hadoop workflow managers like Oozie to perform ETL (Extract, Transform and Load) applications [11] [34]. Data is extracted using MapReduce, then queried using Pig, followed by machine learning algorithms delivering the query results, that are combined with other results [11]. Clearly, there is data-dependency between jobs.

A typical data center workload scenario consists of multiple applications having highly skewed I/O characteristics that exists concurrently as concurrently as shown in Figure 2b. The following hierarchy commonly is maintained for data processing.

1. Application Layer: Data Centers spawn multiple instances of many applications using compute and storage servers. As depicted in Figure 2b, n applications A = $\{A_1, A_2, \ldots, A_n\}$ are waiting to be executed on the cluster at some time instance 't'. These applications vary in nature (i.e. analyzing customer behavior, weather patterns, genomics, financial data, etc.) and are independent.

2. Job Layer: An application A_i, is a conglomeration of several jobs denoted by set J = $\{J_1, \ldots, J_k\}$ with data-dependency. In Figure 3, application A1 is shown in detail as consisting of jobs J_1 to J_k.

3. Task Layer: Task is the smallest granularity of computation which access and process data. Each job J_i in an application is further sub-divided into a set of tasks T = $\{T_1, \ldots, T_p\}$. In a typical scenario, multiples such tasks are run concurrently. Depending on the application, the intermediate results may be required immediately as well as later on.

Data Layer: A task consumes or produces data which is organized across multiple heterogeneous tiers of storage. Due to storage virtualizations, multiple processes contend for the same physical resource for read and write access that

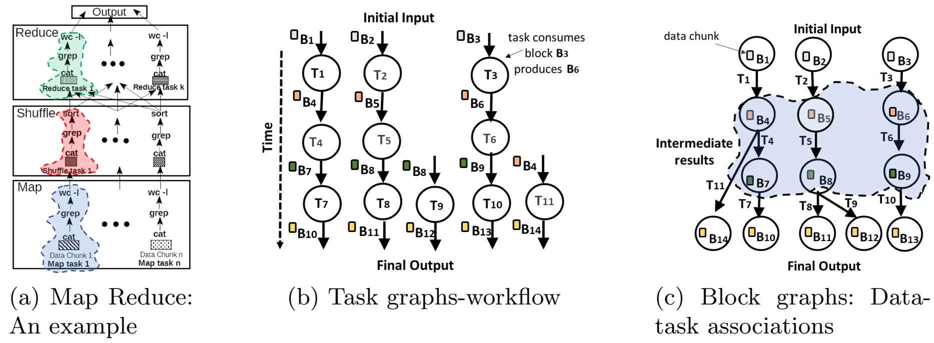
المنارة للاستشارات

www.manaraa.com

may be stored in local (DAS) or in remote storage (NAS/SAN). I/O across the network exposes data transfer to network congestion, delay and losses.

Task graphs

A task graph is a sequence of all the tasks associated with each other and is represented by a DAG (directed acyclic graph). Task graphs can be formed by mining of the application code coupled with logic extraction of the ecosystem. Based on the framework like MapReduce, Spark or Pig, and the logical data flow, the NameNode can create the task graph when an application is submitted. The task graph provides an overview of job and dependencies among tasks currently running on the system as well as maps the application requirements. Figure 3b shows the task graph for a set of inter-linked or chained jobs.



(a) Map Reduce: An example

(b) Task graphs-workflow

(c) Block graphs: Data-task associations

Fig. 3: (a) MapReduce Job: An example of DAGs of tasks (direction of arrows represents event time). Dataflow representations (b) Task graphs- workflow; (c) Block graphs: Data-task associations.

Block graphs: Data-Task associations

Despite capturing the workflow, a task graph fails to understand the lineage of data and associativity of different tasks and blocks of data as conjoint pairs. This is extremely fatal, as a data block might be consumed/produced by multiple tasks and affect the over-all performance of the application. There is a clear need to extract these relationships and utility of each data block to develop proper data management policies. LDM uses the file-system information about the blocks to device mappings (similar to fsimage and editlog for HDFS) stored in the NameNode to associate blocks with tasks to construct a Block graph. In LDM, the Data-Task Associativity API working on the NameNode captures these interactions. The knowledge of these interactions aid in mitigating the impact of delays associated to writing and then subsequently reading intermediate results.

Block graphs are essentially representation of data-task associations, where the blocks of data (as a single entity or replicas) form the vertices and the tasks producing/consuming them as edges as shown in Figure 3c. The utility

المنارة للاستشارات
www.manaraa.com

and reusability of blocks can be determined using this representation. During run-time, only initial input data (initial filename) is available, the initial vertices (block-IDs) are formed using the filesystem namespace and the logical tasks. The graph for later stage is constructed assuming every task produces a data block (not necessarily a full data chunk). Application code mining can determine tasks consuming/producing a data block, thus providing a complete overview of data usage.

An Example: A MapReduce job- DAG of tasks and Block Usage

We use a MapReduce application source code analysis to describe how the Client API can extract the entire computation logic (task graphs) and the Data-Task Association API integrates the filesystem namespace to build block graphs. Figure 3a depicts a simple MapReduce function to count the number of instances of unique words in a file of multiple TBs in size. The code is broken into simple Unix commands, such as, cat, grep, and wc -l (subtasks), and these logical partitions may be executed in different machines.

In the Map phase, 128 MB data chunks are read from storage nodes and are processed to form collection of $< key, value >$ pairs. The Shuffle stage partitions the output of Maps based on "keys" to be processed and transfers data to reducers by the Reduce tasks. The DAG structure and data needs data for computation to be persisted in storage. *Figure 3a clearly depicts the lineage, task graph and how a block graph can be generated from it.*

3.2 Matching Storage capabilities

Hadoop like large distributed systems gained popularity due to their design of bringing computation closer to data which were primarily for DAS setups of acomprising large number of inexpensive machines. However, as we move forward and data being scattered, it is necessary to deploy remote storage servers (NAS/SAN) across thousands and millions of storage devices with varied I/O and physical characteristic. The storage hierarchy may include Main Memory (RAM), Solid State Drives SSDs, and Hard Disk Drives (HDDs), etc. Storage media across all nodes with similar I/O characteristics form a **tier** [4] [11] [34]. The question of when, where and how-to organize data over multiple tiers in the hierarchy becomes important to reap the maximum benefits.

Software defined storage (SDS) is part of the solution to deliver storage services. Automated tiering is one of the major focus to deliver SDS [47]. Recall that *Tiering* refers to orchestrating data between heterogeneous tiers of storage by leveraging individual strengths of each to maintain balance between Cost, Performance and Capacity. Our goal in LDM is to use the information about the storage devices (type, capacity, performance, etc.) as well as location (local/remote) to categorize and classify them. The next step is to implement appropriate APIs to reside both at the DataNodes and storage servers to send their usage status regularly to the NameNode. The NameNode will use the information in making data location decisions.

المنارة للاستشارات
www.manaraa.com

Need for multiple tiers and automated tiering

Disk-based storage devices are the backbone of data centre storage. HDDs provide the perfect blend of cost and capacity to satisfy the volume requirement of Big Data. But due to their physical limitations, non-volatile devices, also known as storage class memories (SCMs), such as SSDs are also being used in large data centres. SCMs offer superior access time due to non-moving parts. SCMs used with legacy disk based interface such as SATA/SAS were incapable of harnessing the throughput and inherent parallelism. However, recent advances such as faster PCIe bus technology (also known as NVMe Express) [48] [49], PCIe switches, Linux block layer redesign, etc. are enabling SCMs to provide higher performance.

Despite superior random performance of SCMs (or SSDs) over HDDs, their higher costs, need for write amplification, and lower lifespan remain concerns for long-term economically feasibility. A hybrid approach with heterogeneous tiers of storage such as those having HDDs and SCMs coupled with workload aware tiering to balance cost, performance and capacity have become increasingly popular [50].

Existing definitions of tiers considers only device characteristics. They do not take into consideration proximity to compute resources and effects of network transfers. Data in a local HDD might be more valuable than in a SSD in a remote location. However, it is not possible to store all data in local storage due to the working set sizes, large spectrum of concurrent applications running on a node and their varying I/O characteristics. Remote storage offers an alternative to satisfy the volume requirement. However, performing I/O across the network makes data transfer prone to network issues like congestion, delay and losses. Such data movements are thus expensive and affect application performance. Therefore, resource manager typically monitors network congestions and tries to utilize replication to maintain performance.

An intelligent and deterministic data management technique would orchestrate the application needs apriori to minimize data movements leveraging ecosystem tools (replication, tiers, etc.) efficiently while ensuring performance. An ideal data management scheme should envision local-remote storage conjoint-pairs analogous to in a similar manner as the cache-main memory model, but with a different interface.

Replication

The data chunks are usually replicated and stored in different nodes across the storage system. The purpose of replication is tri-fold, i.e. achieving fault-tolerance, availability and performance (consider Figure 4). The current schemes [42] [32] [51] [50] [33] of replica management leverage the number of copies or replicas and their locations. [32] designs a heterogeneous storage engine for HPC including RAMdisks, SSDs and HDDs, and Lustre FS to benefit HDFS. The data placement engine in [32] deals with tri-replication of blocks to ensure fault tolerance and the decisions of placement of replicas in a tier is based on storage space available with a usage-priority based tier migration model. [33] proposes a

المنارة للاستشارات

www.manaraa.com

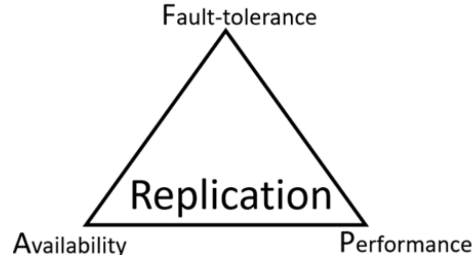


Fig. 4: Advantages of replication.

model with an intent to remove performance bottlenecks by placing every block belonging to file in all tiers of storage.

An interesting approach to replica management would be to i) take into consideration the tier-device characteristics and device utilization and ii) move replicas dynamically based on time-varying application I/O requirement. This adds another dimension which would be highly beneficial to all applications, especially, those which have data-dependency, i.e. lineage based applications. Currently, tiering is mostly concerned with defining hotness or randomness and not on replica management as a tool for multi-tier environments. Well-managed multiple replicas of data will lead to better performance.

Storage Device Classification and Categorization

The classification and categorization of storage devices refers to associating them with a *performance-score or rating, PR. PR* includes performance governing parameters like speed, remaining capacity, number of channels for I/O access (i.e., one for SATA/SAS HDD or SSDs, 8/16 for NVMe SSDs), current utilization, location in storage architecture (DAS or NAS/SAN) and cost/GB. The key goal behind PR is to overcome the deficiencies of current practices and integrate them into the data management policies and tools.

PR is a dynamic parameter for each device as the factor defining them vary over time. The appropriate APIs on DataNodes and the storage servers can collect all such relevant parameters for the devices attached to them, and periodically transmit them to the NameNode via status updates. The Storage Classification API residing on the NameNode utilizes all these parameters to profile every storage device at the run time. The value of PR for a storage device and the location of the client requesting write/read for data placement allows a correct decision to be made. For example, the value of an HDD attached locally can be higher than a SSD in a remote location (location w.r.t. computation locality), or vice-versa. Such decisions are complex and tiering decisions cannot be made solely on device characteristics. Therefore, LDM provides a dynamic and unified method with the use of performance ratings to profile the available resources to aid data management decisions.

المنارة للاستشارات
www.manaraa.com

3.3 Dependency Mitigation

Our goal is to focus on mitigating the impact of associated delays due to incorrect management of data dependency for applications exhibiting lineage. The knowledge gained by using the task and block graphs and the awareness of the storage systems capabilities allow us to develop sound data management policies. Primarily, we will perform *Initial Data Placement, Replication Placement,* and *Data Migration* tasks to decide the storage device(s) to place the data and if, when, and where to move data blocks dynamically.

The Dependency Mitigation component uses the lineage information and the PR values to deliver data placement and replica placement decisions. It includes two APIs, one for initial Data Placement and one for Replica Placement. Data Migration API uses the lineage information to evict already placed blocks as well as determines the utility of the blocks for both capacity and efficient utilization of storage tiers. *Please note that LDM is concerned only with the data management of intermediate results. For some tasks, initial data can be treated as intermediate results if the data is being migrated for the computation.*

HDFS Data Placement: An Example

Currently in HDFS, during the write phase, as soon as a worker writes 80% of a data chunk (64 MB) in memory, it tries to persist the data chunk in storage [43]. The worker contacts the NameNode through RPC calls for a list of DataNodes which can host the chunks and its copies. The NameNode follows rack-aware fault-tolerant algorithms to protect against network failures for the placement of data, generates a list of available DataNodes to the client. The client directly communicates with all DataNodes and pipelines the data chunks one-by-one in 4KB data packets following an ACK based protocol received [43]. This pipelining slows the write process, as to maintain fault tolerance and consistency the slowest DataNode governs the over-all performance. The current schemes do not leverage the tiers of storage available for placement of initial data and its replicas. We propose to use a different strategy in LDM as outlined below.

Initial Data Placement

With the knowledge of all tiers in storage media (PR), the current write as well as retrieval needs, and the knowledge of data-task associations of all the tasks currently running or to be spawned in near futures with the help of the block graphs, LDM is better equipped to dictate policies for placing the initial data across the storage to benefit future reads. When a worker contacts the NameNode for writing a data chunk B, the Initial Data Placement API residing on the NameNode uses the block graphs to compute the data-dependency factor, known as "*Lineage Quotient* LQ_B" for the data block B. LQ_B determines the utility value of block B and based on it, a storage media is selected where the data should be initially placed. To unify the placement algorithm, a global storage table (**GST**) for placement is maintained. GST suggests the storage medias

www.manaraa.com

(based on recently computed PRs) that are suitable for a range of LQ_B as described below.

Algorithm 1 LDM Initial Data Placement

1: **for** every application $a \in A$ **do**
2: Compute block-graphs $BG_a \in BG$
3: **end for**
4: **for** every block $B \in BG$ **do**
5: Compute LQ_B using reusability R
6: Refer global storage table GST to compute refactored $PRs-$
7: (based on locality L of task producing block B)
8: Select storage media m based on LQ_B and refactored PR
9: Send location of storage media m to Worker
10: Pipeline replicas as per Algorithm 2.
11: **end for**

Algorithm 1 describes the working of the Data Placement API. LQ_B is determined by investigating the *reusability* R, i.e. the outdegree of block B from the block graph. Lineage factor LQ_B increases with the number of tasks using it next, i.e. (*reusability* R). The closer the data is placed to the task generating it, lesser is the network footprint usage. Therefore, *locality* L plays a vital role in determining the appropriate rack and storage media in it for the initial placement of data. For all the storage devices, the performance rating PR is refactored based on network distance from *locality* L, higher the distance lower the performance rating. For example, for a task generating block B, a HDD with lower network distance might have a higher refactored performance rating than a SSD which is far away. Based on the *global table* GST, storage media and its location is sent to the worker to write the data.

Replica Placement

Once the first data block (4KB) of the 64MB data chunk is written to the storage media, the replicas needs to be placed in the pipeline. The principle is similar to the current fault tolerance mechanisms of the Hadoop ecosystem. The placement of replicas is managed by the Replica Management API residing in the NameNode. The current schemes, take into consideration fault tolerance for placement of replicas to guard against network failures. To mitigate dependencies, the write throughput of all the pipelined replicas is important which is governed by network bandwidth and storage media. LDM manages this as follows.

Algorithm 2 describes the working of the Replica Placement API for pipelining replicas. Similar to initial data placement the Lineage Quotient for (two additional copies for tri-replication) blocks LQ'_B and LQ''_B, respectively, is calculated with additional parameters. The reusability factor R for every replica is reduced by 1 (with minimum 0) from the previous replica value. Here, the locality L of the previous replica is used only to determine a separate rack for

المنارة للاستشارات
www.manaraa.com

Algorithm 2 LDM Replica Placement

1: **for** every replica r of block B **do**
2: Compute Reusability factor of replica r
3: $R^r = (R-1), R^r \geq 0$
4: Refer global storage table GST to compute refactored PRs
5: (based on locality L of previous replica and storage rack-aware policy)
6: Select storage media m based on LQ_r and refactored PR
7: Send location of storage media m to Worker for pipelining replica
8: $R = R^r$, previous replicas reusability factor.
9: **end for**

storage than the initial block to respect fault tolerance. The assumption here is to use replicas for satisfying the performance for parallel tasks trying to consume the same data set as well as effective capacity utilization of tiers. Therefore, the probability of all replicas occupying the fastest tier is reduced, unless it is a highly dependent block. Based on the Lineage factor of the replicas, the locations are determined in a similar manner to the placement of initial data. The worker is informed of the storage media and the locations for both the blocks and it follows the data write pipeline ACK protocol.

4 Experiments and Performance Evaluation

Through trace-driven and log-based simulations, we evaluate the performance of LDM and compare it with the current implementation of YARN (and HDFS) using our in-house developed system simulators. We discuss the testbed setup followed by the performance evaluation in the section below.

4.1 Testbed setup

Our experimental testbed consist of our Hadoop cluster and trace (and log) collection remote nodes. The Hadoop cluster topology consists of 1 NameNode, 1 Secondary NameNode and 8 DataNodes. Each node has 16 cores (Two 2.0 GHz 8-Core Intel E5 2650), 128 GB of memory, GigE and QDR (40Gbit) Infiniband interconnects, and 2.5 TB Hitachi HDDs. We use CDH v5.11.1 (Cloudera Hadoop) with the latest implementation of YARN and HDFS. The heterogeneous capability is achieved by using specifications similar to 256 GB Samsung SSD 840 pro with the distribution of capacity in the ratio of 1:8 as compared to HDDs attached locally to the nodes.

We select industry and academia wide used Hadoop benchmarks considering a wide diaspora of I/O workload characteristics, as specified in HiBench [52] & TPC Express Benchmark (TPCx-HS)- Hadoop suite [53]. These benchmarks have been designed to recreate enterprise Big Data Hadoop cloud environments, stressing the hardware and software resources (storage, network and compute) as observed in production environment. We use benchmarks to form long-running

المنارة للاستشارات
www.manaraa.com

lineage applications which have inter and intra job data dependency. We also run non-lineage concurrent applications to emulate a realistic shared big data infrastructure. We discuss the details of the applications in the next section.

The NameNode and DataNode statistics, job (and container) running history along with File System details (block locations, chunk-to-device mappings) are collected in remote log collection nodes. We collect traces from the block layer of a disk of the DataNodes in such a stage where the applications have submitted block I/O structures to the block device using the *blktrace* [54] linux utility. The traces include details such as process id (**pid**), CPU core submitting I/O, logical block address (LBA), size (no. of 512 byte disk blocks), data direction (read/write) information for each I/O request[4]. In the next section, we discuss briefly an example of lineage based application, which we use for our experiments followed by the performance evaluation of LDM.

Lineage Based Application: An example

We use chained MapReduce inter-related jobs to emulate applications which exhibit lineage. Figure 5 shows a Lineage application using three MapReduce benchmarks (TeraGen, TeraSort and TeraValidate) which run along with long running non-lineage concurrent applications to form a shared Data Center workload. TeraGen, TeraSort and TeraValidate form logical data-dependent steps to produce the final result.

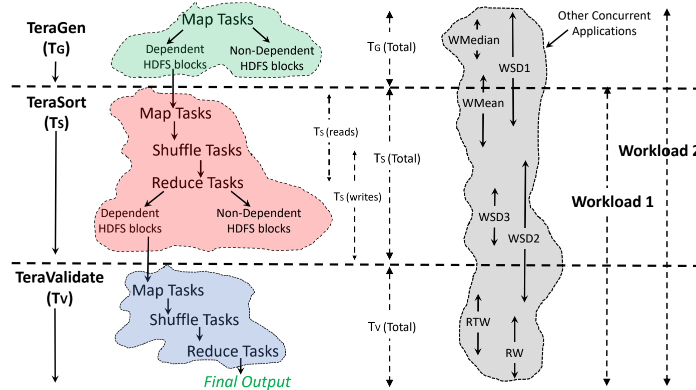


Fig. 5: Data Center Workload Emulation.

TeraGen is a data generating job, which produces 1 TB of random data. TeraGen consists of only Map tasks with no reduces. We use TeraGen as a phase which generates data and persTieists it in HDFS, which in-turn is consumed by the next phase, i.e. TeraSort. TeraSort consumes the data generated by TeraGen

[4] Please note, we collected (stored) the traces remotely on a different machine through the network and not stored in the same local HDFS disk for maintaining the purity of the traces & minimize the effects of the SCSI bus [54–56].

www.manaraa.com

to sort the data. TeraSort consists of Map phase to read the data, there is one map task per HDFS block. The intermediate data is shuffled and partitioned according to the number of reducers, which are sorted and persisted to HDFS in the reduce phase. The result of TeraSort is used by TeraValidate to validate the output of TeraSort. There are other applications which run concurrently. Consider Table 1, we have divided the jobs and applications running during TeraGen, TeraSort and TeraValidate into Phase 1, 2, and 3 respectively. We execute combination of these phases to form two types of workloads which exhibit lineage, the details are described as follows.

Table 1: Experimental Data Center Workloads.
Lineage Application for
Workload 1 "LDM_{W1}": (**TeraSort** (T_S) $\rightarrow$ **TeraValidate** (T_V)).
Workload 2 "LDM_{W2}": (**TeraGen** (T_G) $\rightarrow$ **TeraSort** (T_S) $\rightarrow$ **TeraValidate** (T_V)).

Phase	**Workload 1**	**Workload 2**
Phase 1	*Does Not Exist*	**TeraGen**(T_G), WSD_1, $WMedian$, $WMean$.
Phase 2	WSD_1, **TeraSort**(T_S), WSD_2, $WMean$, WSD_3.	WSD_1, **TeraSort**(T_S), WSD_2, $WMean$, WSD_3.
Phase 3	WSD_2, **TeraValidate**(T_V), RTW, RW.	WSD_2, **TeraValidate**(T_V), RTW, RW.

where, W_x denotes instance x of Workload W.
Abbreviations of Workloads:
WSD: Word Standard Deviation;
WMean: Word Mean;
WMedian: Word Median;
RTW: Random Text Writer;
RW: Random Writer.

Workload 1: For Workload 1, the lineage application LDM_{W1} is a scenario where the data is already residing in HDFS and stored in HDDs, i.e. we assume data generating phase, Phase 1 *Does Not Exists.* The initial data is retrieved from HDDs and **TeraSort** (T_S), sorts the data in Phase 2 and further the output is required by **TeraValidate** (T_V) in Phase 3 for producing the final result. Therefore, the lineage application LDM_{W1} forms a chain **TeraSort** (T_S) $\rightarrow$ **TeraValidate** (T_V). These applications are batch processing where large chunks of data is already present in storage and at run-time it needs to be acquired. There are a many applications, where data is generated or received on the fly

المنارة للاستشارات
www.manaraa.com

(eg: stream applications, joins, etc) and used subsequently for further processing, which are discussed in Workload 2.

Workload 2: For Workload 2, the lineage application LDM_{W2} are such examples of chained jobs, where data is generated on the fly and then subsequent results are produced and consumed. Therefore, the lineage application LDM_{W2} forms a chain **TeraGen** (T_G) $\rightarrow$ **TeraSort** (T_S) $\rightarrow$ **TeraValidate** (T_V).

As can be seen in Figure 5, for every phase we have marked the Dependent and Non-Dependent HDFS blocks. Dependent HDFS blocks are those blocks which are consumed by the set of tasks in future. While Non-Dependent HDFS blocks are those blocks which are either never consumed or those replicas of Dependent blocks which provide fault-tolerance. The discovery of data dependence and mitigating the impact of this dependency is critical for application performance. With trace-driven experiments and performance evaluation of results in the next section, for both data-center workload scenarios, i.e. Workload 1 and Workload 2, we compare the data-dependency management capability of LDM with the latest implementation of HDFS.

4.2 Performance Evaluation

We compare the effectiveness of our dependency mitigation technique scheme, LDM, with the latest implementation of HDFS used deployments for both data-center workload scenarios. For our experiments, we use the default parameters, which is based on the storage devices and driver specifications. Based on trace-driven simulations, in the next section we analyze the performance of both the schemes, i.e. HDFS and LDM for Data Center workloads 1 and 2.

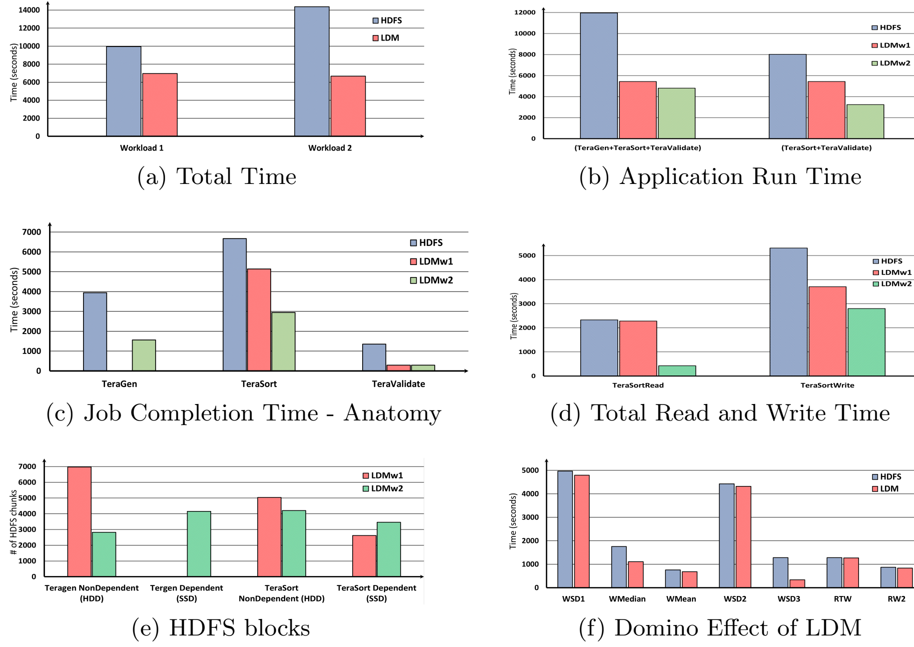
Total Workload Completion Time

Figure 6a represents the total time taken (y-axis) for finishing the data-center workloads. This represents the time to complete all applications in the workload, i.e. lineage as well as other non-lineage applications. It is observed that LDM reduces the time taken to finish the workloads significantly, thereby bridging the deficiencies of HDFS to manage data dependencies. The performance gain (in terms of completion time) is 29% and 52% for workloads 1 and 2, respectively. In LDM, the dependent blocks are identified and placed in SSDs. This serves multi-fold. First, the time to write data and the subsequent future read access is improved as now the dependent blocks do not have to undergo contentions at the disk interface. Secondly, the pipelining of replicas somehow constricts the performance during writes, as the replicas of dependent blocks are placed in HDDs, but placing one (or more) replica in SSD improves the future read time significantly.

It is also observed that though Workload 2 is a subset of Workload 1, and there is more opportunities of optimization via LDM in workload 2, but the gains derived are much more. We discuss in detail with fine grained analysis of each job for both the workloads to understand how LDM optimizes and manages data-dependency which leads to savings in I/O time.

المنارة للاستشارات

www.manaraa.com



(a) Total Time (b) Application Run Time

(c) Job Completion Time - Anatomy (d) Total Read and Write Time

(e) HDFS blocks (f) Domino Effect of LDM

Fig. 6: Performance Evaluation of LDM: (a) Total Time taken by HDFS and LDM; (b) Time taken by Chained Applications; (c) Anatomy of Job completion time of Chained MapReduce Applications; (d) Total Read and Write Time of TeraSort job to show the difference between LDM for lineage applications LDM_{w1} and LDM_{w2}; (e) Dependent and Non-Dependent blocks; (f) Impact of LDM on other concurrent applications running at the same time.

Time taken by chained applications

Figure 6b represents the time taken (y-axis) for completing (TeraGen T_G + TeraSort T_S + TeraValidate T_V) and (TeraSort T_S + TeraValidate T_V) by Hadoop (HDFS), and LDM optimizations during workloads 1 and 2, i.e. LDM_{w1} and LDM_{w2}, respectively. It is observed that LDM outperforms HDFS by 55%. The graphs for LDM_{w1} is same, as the data already resides in HDD for consumption for TeraSort, i.e. TeraGen does not exists. Moreover, we also observe that the time taken to complete (TeraSort T_S + TeraValidate T_V) for LDM_{w1} is lower than LDM_{w2} by 40%, though in both cases, we calculate the times from the start of TeraSort and to the finish of TeraValidate.

In-order to understand the difference between performance for applications LDM_{w1} and LDM_{w2} from the beginning of TeraSort T_S and end of TeraValidate T_V using LDM, we plot the individual job completion times of TeraGen, TeraSort and TeraValidate, as shown in Figure 6c.

From Figure 6c, we observe the following: 1) The time taken by LDM is significantly lower than for all jobs. 2) TeraGen does not exist for LDM_{w1}, as we

www.manaraa.com

assume that the data is already residing in HDFS. Hence for the analysis of results between LDM optimizations for lineage applications LDM_{w1} and LDM_{w2}, we do not consider the time taken to complete TeraGen. 3) Time taken by TeraValidate for both LDM_{w1} and LDM_{w2} is same, suggesting that most of the difference occurs during TeraSort phase. Therefore, we further investigate the TeraSort job.

Figure 6d shows the total time taken to read or write, right from the first block of read to the last block (similarly for writes) belonging to TeraSort job, i.e. ($T_s(reads)$) and ($T_s(writes)$) respectively.

For reads, there is marginal gain between HDFS and LDM_{w1}, while there is significant improvements for LDM_{w2}. This is attributed to the characteristic of the job, as most of the reads occur in the Map phase, for LDM_{w1}, as the data is fetched from HDDs which is nearly same as with no optimization, i.e. HDFS. While for LDM_{w2}, the TeraGen phase is optimized and LDM places (writes) dependent blocks in SSDs, therefore, leading to large reduction in time for reads during TeraSort. We classify the data chunks (128 MB) into dependent and non-dependent, i.e. those HDFS blocks decided by LDM to be placed in SSD and HDD, respectively.

Figure 6e shows the number of HDFS chunks (y-axis) versus the dependent or non-dependent blocks decided by LDM to be placed during that phase. The dependent blocks are written during that phase and consumed in the next. While non-dependent are those data chunks which are those blocks which are not required for computation in near future. We observe that for TeraGen, all blocks are classified non-dependent for LDM_{w1}, as those data sets are already residing in HDD, while none are dependent. For LDM_{w2} application, the dependent blocks are placed in SSD, which are used during the TeraSort run. This justifies the higher read performance for LDM_{w2} than LDM_{w1} for the same LDM optimization observed at the same start and end point (in terms of job completion).

During the writes of TeraSort (refer to Figure 6d), in both scenarios the time is reduced significantly. It would be expected that the write time for LDM_{w1} and LDM_{w2} should be same, as the difference is only for the reads from SSD (in case of LDM_{w2}) and writes should be the same but LDM_{w2} performs 17% better. This is because of the nature of the dataflow framework (MapReduce), which does a lot of inter-mediate local writes, which in the case of LDM_{w2} is in SSD, which is faster.

Therefore, we observe that LDM is able to manage the data-dependency while reducing the time taken by over-all workload and chained applications. We further study the impact of LDM on other concurrent non-lineage data center applications.

Impact of LDM on other concurrent applications

Figure 6f represents the time taken by different concurrent non-lineage applications by Hadoop (HDFS) and LDM. It clearly shows that by employing LDM, the I/O time for other concurrent applications also reduces. It forms a *Domino effect*, in which optimizing on type of application also effects the performance

المنارة للاستشارات
www.manaraa.com

of others. This is so because some of the blocks (dependent) which belong to Chained applications are offloaded from the HDD request queue and placed in a different media (SSD). Therefore, there are fewer interferences and the multiplexing of I/O effect is reduced, which leads to over-all savings in execution time for non-lineage concurrent applications.

LDM is designed and developed to mitigate the impact of delays associated to dependency of data in lineage class of applications. Through trace-driven based experiments, LDM shows to successfully orchestrate the application needs apriori and match storage capabilities to deliver performance. There is also evidence that by deploying LDM, the execution time of other applications also reduces. We conclude in the next section with discussions on future work.

5 Conclusion and Future Work

LDM provides a uniform execution environment across storage and compute, which addresses specific needs of applications with data-dependency (or *lineage*). LDM is a lineage-aware data management system which has an oracle-like deterministic capability to know the future usage of data based on knowledge already present in the data processing framework and ecosystem. These informations and statistics are being produced and consumed by different components of the system. LDM amalgamates these informations from the entire data center ecosystem to dictate tier-aware storage policies for lineage class of applications to mitigate the impact of data dependency for lineage class of applications. Through the development of **block graphs**, LDM is able to capture the complete time-based data-task associations and use it to perform life-cycle management through tiering of data blocks belonging to applications exhibiting lineage. With trace-driven experiments, LDM is able to achieve 29% to 52% reduction in over-all data center workload execution time. Moreover, by deploying LDM with extensive pre-processing creates efficient data consumption pipelines, which also reduces write and read delays significantly.

In future, we plan to develop and design data migration capabilities in LDM as well implement it for PCIe based NVMe SSDs to leverage parallelism provided by them. We also plan to implement LDM for hybrid set-ups comprising of DAS, NAS and SAN setups with a wide variety of HDDs, and SCMs (with different combinations) and study the data movement impact across networks. LDM opens an avenue for a large diaspora of application and data processing frameworks and we have implemented it for MapReduce environments. It would be interesting to extend LDMs capability to understand various other frameworks like Spark, Parquet, etc. and work together in a unified environment to cater to different syntax and semantics of applications. Broader impact of LDM is that it would aid Data Centers to effectively utilize multiple tiers of storage while keeping the Total Cost of Ownership (TCO) low as well as ensuring lower memory and resource footprint leading to energy savings.

المنارة للاستشارات
www.manaraa.com

References

1. R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson, "Near-data processing: Insights from a MICRO-46 workshop," *IEEE Micro*, vol. 34, no. 4, pp. 36–42, 2014.
2. D. Tiwari, S. S. Vazhkudai, Y. Kim, X. Ma, S. Boboila, and P. J. Desnoyers, "Reducing Data Movement Costs Using Energy-Efficient, Active Computation on SSD," in *Presented as part of the 2012 Workshop on Power-Aware Computing and Systems*, 2012.
3. P. Mishra and A. K. Somani, "Host managed contention avoidance storage solutions for big data," *Journal of Big Data*, vol. 4, no. 18, 2017.
4. I. Iliadis, J. Jelitto, Y. Kim, S. Sarafijanovic, and V. Venkatesan, "Exaplan: Queueing-based data placement and provisioning for large tiered storage systems," in *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2015 IEEE 23rd International Symposium on*, Oct 2015, pp. 218–227.
5. Y. Kim, A. Gupta, B. Urgaonkar, P. Berman, and A. Sivasubramaniam, "Hybridstore: A cost-efficient, high-performance storage system combining ssds and hdds," in *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, July 2011, pp. 227–236.
6. P. Mishra, M. Mishra, and A. K. Somani, "Bulk i/o storage management for big data applications," in *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2016 IEEE 24th International Symposium on.* IEEE, 2016, pp. 412–417.
7. B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center." in *NSDI*, vol. 11, 2011, pp. 22–22.
8. M. Bjørling, J. Axboe, D. Nellans, and P. Bonnet, "Linux block io: Introducing multi-queue ssd access on multi-core systems," in *Proceedings of the 6th International Systems and Storage Conference*, ser. SYSTOR '13. New York, NY, USA: ACM, 2013, pp. 22:1–22:10. [Online]. Available: http://doi.acm.org/10.1145/2485732.2485740
9. M. Bhadkamkar, J. Guerra, L. Useche, S. Burnett, J. Liptak, R. Rangaswami, and V. Hristidis, "Borg: Block-reorganization for self-optimizing storage systems," in *Proccedings of the 7th Conference on File and Storage Technologies*, ser. FAST '09. Berkeley, CA, USA: USENIX Association, 2009, pp. 183–196. [Online]. Available: http://dl.acm.org/citation.cfm?id=1525908.1525922
10. M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation.* USENIX Association, 2012, pp. 2–2.
11. H. Li, A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Tachyon: Reliable, memory speed storage for cluster computing frameworks," in *Proceedings of the ACM Symposium on Cloud Computing.* ACM, 2014, pp. 1–15.
12. J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, pp. 107–113, 2008.
13. A. K. Somani, M. Mishra, and P. Mishra, "Applications of hadoop ecosystems tools," in *NoSQL.* Chapman and Hall/CRC, 2017, pp. 173–190.
14. Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "Haloop: Efficient iterative data processing on large clusters," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 285–296, Sep. 2010. [Online]. Available: http://dx.doi.org/10.14778/1920841.1920881

المنارة للاستشارات
www.manaraa.com

15. G. Ananthanarayanan, A. Ghodsi, A. Wang, D. Borthakur, S. Kandula, S. Shenker, and I. Stoica, "Pacman: Coordinated memory caching for parallel jobs," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 20–20.
16. E. L. Sun and S. Das, "Data-dependency-driven flow execution," Mar. 1 2018, uS Patent App. 15/249,841.
17. A. Aggarwal, R. Arasan, S. Bose, D. Das, R. K. Kaushik, M. K. Meyer, G. Ramasamy, and J. D. Seideman, "System and method for automatically capturing and recording lineage data for big data records," May 18 2017, uS Patent App. 14/944,849.
18. S. P. MacLeod, C. L. Kiernan, and V. Rajarajan, "Data lineage data type," Aug. 13 2002, uS Patent 6,434,558.
19. A. J. Peacock, C. Couris, C. Storm, A. Netz, C. Y. Cheung, M. J. Flasko, K. Grealish, G. M. Della-Libera, S. P. Carlson, M. W. Heninger *et al.*, "Job scheduling and monitoring," Oct. 5 2017, uS Patent App. 15/596,709.
20. O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom, "Uldbs: Databases with uncertainty and lineage," in *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 2006, pp. 953–964.
21. R. S. Krajec and A. G. Gounares, "Highlighting of time series data on force directed graph," Apr. 26 2016, uS Patent 9,323,863.
22. A. S. Mohammad, M. A. Boston, and I. Lapsker, "Data lineage transformation analysis," May 24 2016, uS Patent 9,348,879.
23. A. Woodruff and M. Stonebraker, "Supporting fine-grained data lineage in a database visualization environment," in *Data Engineering, 1997. Proceedings. 13th International Conference on*. IEEE, 1997, pp. 91–102.
24. R. Bose, "A conceptual framework for composing and managing scientific data lineage," in *Scientific and Statistical Database Management, 2002. Proceedings. 14th International Conference on*. IEEE, 2002, pp. 15–19.
25. R. Bose and J. Frew, "Lineage retrieval for scientific data processing: a survey," *ACM Computing Surveys (CSUR)*, vol. 37, no. 1, pp. 1–28, 2005.
26. J. Widom, "Trio: A system for integrated management of data, accuracy, and lineage," Stanford InfoLab, Tech. Rep., 2004.
27. C. Ré and D. Suciu, "Approximate lineage for probabilistic databases," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 797–808, 2008.
28. T. Harter, D. Borthakur, S. Dong, A. Aiyer, L. Tang, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Analysis of hdfs under hbase: A facebook messages case study," in *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 14)*. Santa Clara, CA: USENIX, 2014, pp. 199–212. [Online]. Available: https://www.usenix.org/conference/fast14/technical-sessions/presentation/harter
29. D. Choi, M. Jeon, N. Kim, and B.-D. Lee, "An enhanced data-locality-aware task scheduling algorithm for hadoop applications," *IEEE Systems Journal*, 2017.
30. F. N. Afrati and J. D. Ullman, "Optimizing joins in a map-reduce environment," in *Proceedings of the 13th International Conference on Extending Database Technology*. ACM, 2010, pp. 99–110.
31. M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets." *HotCloud*, vol. 10, pp. 10–10, 2010.
32. N. S. Islam, X. Lu, M. W. ur Rahman, D. Shankar, and D. K. Panda, "Triple-h: A hybrid approach to accelerate hdfs on hpc clusters with heterogeneous storage architecture," in *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, May 2015, pp. 101–110.

المنارة للاستشارات
www.manaraa.com

33. K. R. Krish, A. Anwar, and A. R. Butt, "hats: A heterogeneity-aware tiered storage for hadoop," in *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, May 2014, pp. 502–511.
34. E. Kakoulli and H. Herodotou, "Octopusfs: A distributed file system with tiered storage management," in *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 2017, pp. 65–78.
35. M. Grund, J. Krüger, H. Plattner, A. Zeier, P. Cudre-Mauroux, and S. Madden, "Hyrise: a main memory hybrid storage engine," *Proceedings of the VLDB Endowment*, vol. 4, no. 2, pp. 105–116, 2010.
36. N. S. Islam, M. Wasi-ur Rahman, X. Lu, and D. K. D. Panda, "Efficient data access strategies for hadoop and spark on hpc cluster with heterogeneous storage," in *Big Data (Big Data), 2016 IEEE International Conference on*. IEEE, 2016, pp. 223–232.
37. S. Lee, J.-Y. Jo, and Y. Kim, "Performance improvement of mapreduce process by promoting deep data locality," in *Data Science and Advanced Analytics (DSAA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 292–301.
38. P. K. Gunda, L. Ravindranath, C. A. Thekkath, Y. Yu, and L. Zhuang, "Nectar: Automatic management of data and computation in datacenters." in *OSDI*, vol. 10, 2010, pp. 1–8.
39. J. T. Olson, S. R. Patil, R. M. Shiraguppi, and G. A. Spear, "Handling data block migration to efficiently utilize higher performance tiers in a multi-tier storage environment," Apr. 27 2017, uS Patent App. 15/499,274.
40. G. Zhang, L. Chiu, and L. Liu, "Adaptive data migration in multi-tiered storage based cloud environment," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. IEEE, 2010, pp. 148–155.
41. M. Mihailescu, G. Soundararajan, and C. Amza, "Mixapart: decoupled analytics for shared storage systems," in *In USENIX FAST*, 2012.
42. K. Krish, B. Wadhwa, M. S. Iqbal, M. M. Rafique, and A. R. Butt, "On efficient hierarchical storage for big data processing," in *Cluster, Cloud and Grid Computing (CCGrid), 2016 16th IEEE/ACM International Symposium on*. IEEE, 2016, pp. 403–408.
43. V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. New York, NY, USA: ACM, 2013, pp. 5:1–5:16. [Online]. Available: http://doi.acm.org/10.1145/2523616.2523633
44. W. Lu, Y. Wang, J. Jiang, J. Liu, Y. Shen, and B. Wei, "Hybrid storage architecture and efficient mapreduce processing for unstructured data," *Parallel Computing*, vol. 69, pp. 63–77, 2017.
45. G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris, "Scarlett: coping with skewed content popularity in mapreduce clusters," in *Proceedings of the sixth conference on Computer systems*. ACM, 2011, pp. 287–300.
46. S. Moon, J. Lee, X. Sun, and Y.-S. Kee, "Optimizing the hadoop mapreduce framework with high-performance storage devices," *J. Supercomput.*, vol. 71, no. 9, pp. 3525–3548, Sep. 2015. [Online]. Available: http://dx.doi.org/10.1007/s11227-015-1447-3
47. M. Di Mauro, M. Longo, F. Postiglione, G. Carullo, and M. Tambasco, "Software defined storage: Availability modeling and sensitivity analysis," in *Perfor-*

المنارة للاستشارات
www.manaraa.com

mance Evaluation of Computer and Telecommunication Systems (SPECTS), 2017 International Symposium on. IEEE, 2017, pp. 1–7.

48. M. Lee, D. H. Kang, M. Lee, and Y. I. Eom, “Improving read performance by isolating multiple queues in nvme ssds,” in *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication.* ACM, 2017, p. 36.
49. K. T. Malladi, M. Awasthi, and H. Zheng, “Flexdrive: A framework to explore nvme storage solutions,” in *High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016 IEEE 18th International Conference on.* IEEE, 2016, pp. 1115–1122.
50. J. N. Khasnabish, M. F. Mithani, and S. Rao, “Tier-centric resource allocation in multi-tier cloud systems,” *IEEE Transactions on Cloud Computing*, vol. 5, no. 3, pp. 576–589, 2017.
51. A. Spivak, A. Razumovskiy, D. Nasonov, A. Boukhanovsky, and A. Redice, “Storage tier-aware replicative data reorganization with prioritization for efficient workload processing,” *Future Generation Computer Systems*, 2017.
52. S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, “The hibench benchmark suite: Characterization of the mapreduce-based data analysis,” in *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, March 2010, pp. 41–51.
53. TPC(tm). (2016, Aug) Tpc express benchmark(tm) hs (tpcx-hs)-hadoop suite overview. [Online]. Available: http://www.tpc.org/tpcx-hs/
54. A. D. Brunelle, “blktrace user guide,” February 2007.
55. A. Riska, J. Larkby-Lahet, and E. Riedel, “Evaluating block-level optimization through the io path,” in *2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, ser. ATC’07. Berkeley, CA, USA: USENIX Association, 2007, pp. 19:1–19:14. [Online]. Available: http://dl.acm.org/citation.cfm?id=1364385.1364404
56. F. Chen, D. A. Koufaty, and X. Zhang, “Hystor: Making the best use of solid state drives in high performance storage systems,” in *Proceedings of the International Conference on Supercomputing.* New York, NY, USA: ACM, 2011, pp. 22–32. [Online]. Available: http://doi.acm.org/10.1145/1995896.1995902

المنارة للاستشارات
www.manaraa.com