MixApart: Decoupled Analytics for Shared Storage Systems

Madalin Mihailescu, Gokul Soundararajan[†], Cristiana Amza
University of Toronto, NetApp[†]

Abstract

Data analytics and enterprise applications have very different storage functionality requirements. For this reason, enterprise deployments of data analytics are on a separate storage *silo*. This may generate additional costs and inefficiencies in data management, e.g., whenever data needs to be archived, copied, or migrated across silos. We introduce MixApart, a scalable data processing framework for shared enterprise storage systems. With MixApart, a single consolidated storage back-end manages enterprise data and services all types of workloads, thereby lowering hardware costs and simplifying data management. In addition, MixApart enables the local storage performance required by analytics through an integrated data caching and scheduling solution. Our preliminary evaluation shows that MixApart can be 45% faster than the traditional *ingest-then-compute* workflow used in enterprise IT analytics, while requiring one third of storage capacity when compared to HDFS.

1 Introduction

Data analytics frameworks, such as MapReduce [5, 9] and Dryad [12], are utilized by many organizations to process increasing volumes of information; activities range from analyzing e-mails, transaction records, to executing clustering algorithms for customer segmentation. Using such data flow frameworks, data partitions are loaded from an underlying distributed file system, passed through a series of operators executed in parallel on the compute nodes, and the results are written back to the file system. The file system component [5, 11] is a standalone storage system by itself; data is stored on local server disks, with redundancy (e.g., 3-way replication) to provide data protection.

The benefits provided by data analytics frameworks are well understood. Yet, it is unclear how to integrate them with existing enterprise storage systems.

Enterprise storage manages *high-value* data, i.e., emails and transaction records, that need to be secured from tampering, protected from failures, and archived; high-value data demand enterprise-level data management features. Traditional analytics, however, have been designed for *low-value* data, i.e., web logs and click streams, that can be regenerated on a daily basis; low-value data do not need enterprise-level data management. Consequently, file systems built for data analytics lack many of the data management features essential for the enterprise environment, e.g., support for standard protocols, storage efficiency features, and data protection mechanisms. Being purpose-built, these file systems fail to take into account the usual data lifecycle.

We classify the data lifecycle into three phases: (i) data acquisition, (ii) data use, and (iii) data retirement and study the limitations of the purpose-built file systems in each phase. First, enterprises cannot migrate their existing applications to the newer file systems, due to their use of standard SCSI/NFS protocols, as opposed to the specialized APIs shipped with the new frameworks. Second, the design points of these newer file systems are not suited for enterprise workloads. Enterprise workloads have small files, have overwrites, and require strong consistency [8], whereas file systems built for analytics expect large files, file appends, and provide relaxed consistency [11]. Third, as its value decreases, the data must be retired from active use, yet it must be accessible to recover from failures as well as to meet regulatory compliance. File systems for analytics [5, 11] utilize data replication for both performance and data reliability. While replication helps for active data, data protection techniques such as erasure coding [13] are better suited to protect inactive data.

Due to these disparities, enterprise IT deploys analytics for *high-value* data in a separate storage system (*silo*). This creates an environment where an *enterprise silo* contains data for user-facing enterprise applications and an *analytics silo* contains analytics data, usually a subset of the total enterprise data. This deployment leads to a substantial upfront investment as well as complex workflows to manage data across different file systems. Enterprise analytics typically works as follows: (i) an application records data into the enterprise silo, (ii) an extraction workflow reads the data and ingests the data into the analytics silo, (iii) an analytics program is executed and the results of the analytics are written into the analytics file system, (iv) the results of the analytics are loaded back into the enterprise silo. As new data arrives, the entire workflow is repeated.

MixApart *replaces* the distributed file system component in current analytics frameworks with an end-to-end distributed storage solution, XDFS, composed of:

- a stateless *caching layer*, built out of local disks, colocated with compute for data analytics *performance*,
- a *shared storage system* for data reliability and data management, and
- a *data scheduler* that coordinates data transfers from shared storage into the disk cache, just in time.

MixApart targets comparable performance to a dedicated data analytics framework for *high value* enterprise data, at similar compute scales, while improving storage efficiency and simplifying data management. By relying on existing enterprise storage for data management, XDFS removes the redundancy requirements in current distributed file systems, thereby lowering hardware costs. Moreover, the XDFS disk cache stores only relevant data as requested by analytics, further increasing efficiency.

The decoupled design allows deployment of most processing required by data analytics as *apart* from its data storage system, within the same data center, or even on cloud infrastructures; at the same time, a single consolidated storage back-end manages and services data for all types of workloads.

2 MapReduce Workload Analysis

Data analytics frameworks process data by splitting a user-submitted *job* into several *tasks* that run in parallel. In the input data processing phase, e.g., *Map* phase, tasks read data partitions from the underlying distributed file system, compute the intermediate results by running the computation specified in the task, and shuffle the output to the next set of operators, e.g., *Reduce* phase. The bulk of the data processed is read in this initial phase.

Recent studies [3, 7] of production workloads deployed at Facebook, Microsoft Bing, and Yahoo! make three key observations: (i) there is high *data reuse* across jobs, (ii) the input phases are, on average, *CPU-intensive*, and (iii) the I/O demands of jobs are *predictable*. Based on the above workload characteristics, we motivate our MixApart design insights. We then show that, at typical I/O demands and data reuse rates, MixApart can sustain large compute clusters, equivalent to those supported by current dedicated deployments.

2.1 High Data Reuse across Jobs

Production analytics workloads exhibit high data reuse across jobs with only 11%, 6%, and 7% of jobs from Facebook, Bing, and Yahoo!, respectively, reading a file once [3]. Using the job traces collected, Ananthanarayanan et al. estimate in-memory cache hit rates in the region of 60% with an optimal cache replacement policy [3], by allocating 32GB of memory on each machine.

MixApart's distributed storage employs a cache layer built from local server drives; this disk cache is two orders of magnitude larger than an in-memory cache. Hence, even a simple LRU policy suffices to achieve near-optimal data reuse. Furthermore, we expect reuse to increase as computations begin to rely on iterative processing – e.g., Mahout [6]. Interconnected jobs, such as job pipelines, will naturally exhibit data reuse in MixApart, as the current job input would be the output of the previous job. These trends and observations indicate that, by caching data after the first access, MixApart can significantly reduce the number of I/Os issued to shared storage for subsequent accesses.

2.2 CPU-intensive Input Phases

In addition to high data reuse, operations such as compression/decompression, data serialization, task setup/cleanup, and output sort increase the average time spent on the CPU [2]. Indeed, Zaharia et al. show that the median map task duration is 19s for Facebook's workloads, and 26s for Yahoo!'s [15], while the typical partition size is 64MB. Higher processing times indicate that a task's effective I/O rate is low, thereby there is ample time to move data from the shared storage to the distributed cache.

For instance, a task running for 20s to process a 64 MB input partition implies a task I/O rate of 25 Mbps. A storage server with a 1 Gbps link sustains 40 such map tasks concurrently, even when all data is read from shared storage. The distributed cache layer further improves scalability. For example, with a 70% cache hit rate and a task I/O rate of 25 Mbps, more than 130 map tasks can process data in parallel from cache and shared storage.

2.3 Predictable I/O Demands

Average high data reuse and low task I/O rates confirm the feasibility of MixApart. Individual job patterns that deviate from the average, however, could potentially impact the scalability, by congesting the shared storage when data reuse is low and aggregate job I/O demands are high. Hence, *coordination* is required to smooth out shared storage traffic and ensure efficient storage bandwidth utilization at all times.

Previous studies observe that production workloads have very predictable task times [3, 7]. In fact, the analysis of Facebook traces [7] shows that processing jobs can be classified into 6 bins. In addition, tasks of a job have similar durations [3]. MixApart uses task durations to derive an *effective map task I/O rate*, i.e., the rate at which a task reads data from storage, at job submission time. Given the available storage bandwidth, MixApart can use the respective I/O rates to proactively schedule data transfers from shared storage to its on-disk caches *just in time* and *in parallel* for job tasks. By having a *global* view of future job I/O demands, MixApart can schedule storage traffic intelligently in order to smooth out traffic to shared storage across all compute jobs.

2.4 Estimating Cluster Sizes Supported

We expand our analysis to estimate the average compute cluster sizes that can be supported by MixApart based on the workload characteristics introduced above, i.e., the typical data reuse across jobs, the computation to I/O ratio, and the storage bandwidth utilization.

Figure 1 plots the cluster size (in number of map tasks) for various workload characteristics. We vary the data reuse ratios from 0 to 0.99, task durations from 0.5s to 40s, and vary the storage bandwidth between 1 Gbps and 10 Gbps. The analysis shows that large analytics clusters can be sustained; for example, with a data reuse rate of 80% and average map task duration of 20s, MixApart can support 2000 parallel tasks.

3 MixApart Architecture

MixApart uses the disk space available within each compute node in the analytics dedicated cluster as an on-disk caching tier for scalable and efficient processing of shared storage enterprise data. Users submit MapReduce *jobs* on datasets stored in shared storage; a job is split into multiple *tasks* at specific data granularities (e.g., 64MB) and executed in parallel on the cluster. MixApart has two goals: (i) preserve the scalability and performance benefit of compute-data co-location, and (ii) ensure efficient resource utilization. MixApart addresses these goals by utilizing two essential components:

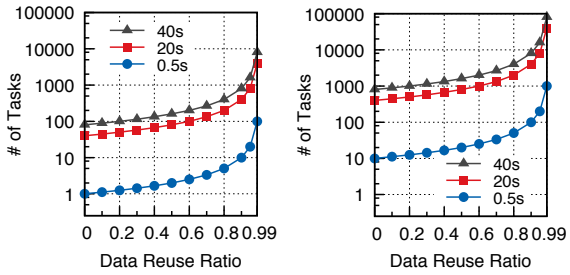


Figure 1: **Compute Cluster Sizes.** We show the number of parallel map tasks sustained by the cache + shared storage architecture for MapReduce analytics. Labels represent average map task durations – e.g., 0.5s for I/O intensive tasks. We plot values for cache-storage bandwidths of (a) 1Gbps, and (b) 10Gbps.

- *Data-aware Task Scheduler*: a modified task scheduler that schedules tasks using a scheduling policy (*FIFO* or *Fair*) and the on-disk cache contents,
- *Compute-aware Data Scheduler*: a module that transfers data from shared storage to caches, as needed by tasks, using the scheduling policy and task I/O rates.

Figure 2 shows the MixApart architecture, composed of two layers: a distributed compute layer and a distributed data storage layer (XDFS), along with the schedulers and metadata managers associated with the two layers. Job tasks are scheduled onto compute nodes by a *task scheduler*. The *data scheduler* schedules data transfers from shared storage into the cache to be performed by XDFS data nodes. The two schedulers exchange scheduling and location information about the tasks and data they respectively manage. The resulting schedules optimize data fetch *parallelism*, overlap computation and I/O latency, and co-locate computation and data.

Specifically, once a job is submitted to the compute layer, this layer passes job-level information, such as data requests, and I/O rates to the *data scheduler* located in the XDFS layer. Guided by this compute-layer information, the *data scheduler* schedules requests for data on behalf of job tasks; data blocks are transferred from shared storage into the XDFS on-disk cache. The data scheduler communicates back to the *task scheduler* data location information within the on-disk cache. This location information is used by our data-aware task scheduler to choose the appropriate compute node for each task.

3.1 Data-Aware Task Scheduler

The data-aware task scheduler prioritizes the execution of tasks according to the chosen scheduling policy as well as the cache contents. The task scheduler allows tasks to fully utilize the cache, while maintaining the underlying policy. For example, the *FIFO* policy schedules

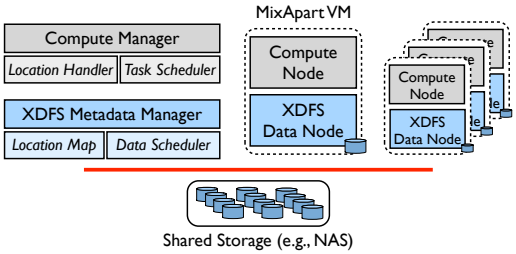


Figure 2: **MixApart Architecture.**

jobs based on arrival time and a user-defined job priority, while *Fair* schedules jobs in increasing order of number of running tasks [15]. Upon resource availability events from compute nodes, the task scheduler assigns a task from the job at the head of the scheduling queue, prioritizing tasks with a higher fraction of data already in the cache, or on their way to the cache.

The task scheduler is work conserving: if tasks on behalf of jobs currently expected to run do not saturate the compute capacity due to lack of input data in the cache, the scheduler selects tasks from lower priority jobs.

3.2 Compute-Aware Data Scheduler

MixApart uses per-job task I/O rates and the knowledge of the scheduling policy to efficiently transfer data from shared storage into cache nodes, given the available storage bandwidth. MixApart allows administrators to specify bounds on bandwidth consumed by analytics workloads, such that enterprise applications are not impacted.

The data scheduler mimics the job scheduling logic to ensure that the next expected job will have its input data available in the cache. For example, when the *FIFO* policy is used at the compute tier, *job arrival times* and *priorities* determine the data transfer order. Similarly, with *Fair*, transfer requests are sorted in increasing order of number of *per-job not-yet-analyzed cached data blocks* (including blocks under analysis). Transfer requests are further sorted per-job, by global block demand, i.e., the number of tasks interested in a data block, across all jobs.

4 Evaluation

We implement a MixApart prototype within Hadoop by adding the XDFS caching logic and the compute-aware data scheduler; the task scheduler uses the *FIFO* policy. We evaluate MixApart on a 100-core cluster connected to a NFS server, running on Amazon EC2 [1]; the XDFS caches access the storage using local mount points. We compare MixApart to a standalone Hadoop framework.

Experimental Setup: We use 50 EC2 "standard-large" instances to deploy the compute and the XDFS data nodes; each instance hosts a pair of compute-data, and is configured with 2 virtual cores, 7.5 GB RAM, and 850 GB of local *ephemeral* storage. Each compute node provides 2 map slots and 1 reduce slot. The compute and the XDFS metadata managers use one "standard-extra-large" instance (4 virtual cores, and 15 GB of RAM). The shared storage is provided by a "cluster-compute-extra-large" instance configured with 23 GB RAM, 2 Intel Xeon X5570 quad-core CPUs, and 10 Gbps network. We configure the shared storage server to store data on 4 EBS (*elastic block storage*) volumes assembled in a RAID-0 setting; the storage is exposed using NFS.

We use local instance storage for the XDFS cache; the same disks are also used for HDFS. HDFS is configured with 3-way replication. We limit the NFS server bandwidth to 1 Gbps. This setting also mimics a production environment where enterprise workloads would still be able to use the remaining 9 Gbps.

Dataset: We use 12 days of Wikipedia statistics[1] as the dataset; the data is 82 GB uncompressed (average file size 295MB) and 23 GB compressed (85MB). HDFS uses 246 GB of storage capacity for uncompressed data, and 69 GB for compressed. MixApart uses 82 GB, and 23 GB for the cache, respectively, when the entire dataset is analyzed. As expected, MixApart's storage needs are one third of HDFS capacity needed. In general, with only a subset of the total data being actively analyzed, MixApart is able to capture the relevant data for analytics in the cache, further reducing overall storage requirements.

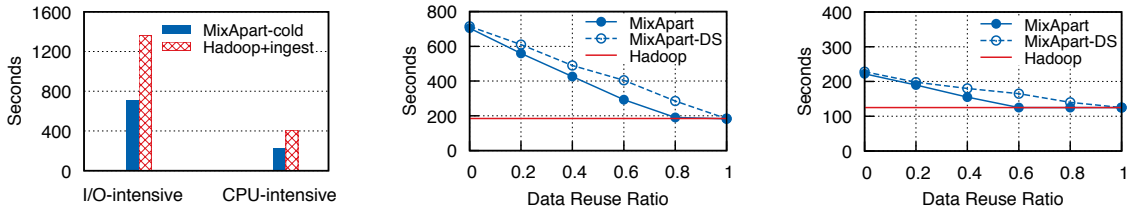
Workloads: We run a simple MapReduce job to aggregate page views for a regular expression; we run the job on uncompressed and compressed input. Tasks processing uncompressed data are more I/O intensive than when input is compressed. Namely, for *uncompressed*, the effective map task I/O rates are roughly 50 Mbps. For *compressed*, I/O rates are 20 Mbps (due to higher CPU use). We denote runs on uncompressed data as *I/O-intensive*, and runs on compressed data as *CPU-intensive*.

4.1 Preliminary Results

Figure 3 shows that: (a) overlapping the computation with data ingest improves performance, (b) data reuse allows MixApart to match the performance of Hadoop+HDFS, and (c) scheduling data transfers using compute-awareness enables efficient bandwidth use.

Data Ingest: We compare the performance of MixApart with no data in the cache (denoted MixApart-cold) to Hadoop (Hadoop+ingest) – Figure 3(a). With Hadoop+ingest, data is ingested into HDFS before a job starts. By overlapping computation and data transfers, MixApart reduces job durations by approximately 45%.

[1] http://dumps.wikimedia.org/other/pagecounts-raw



(a) Ingest Time (b) Pageview (I/O intensive) (c) Pageview (CPU intensive)

Figure 3: **Analytics on a 100-core cluster with MixApart and Hadoop.** We show *job durations* for a MapReduce job running on a Wikipedia dataset. The *CPU-intensive* experiment processes the *compressed* data; the *I/O-intensive* processes the *uncompressed* data. In (a) we run MixApart with a cold cache (MixApart-cold). In (b), (c) we run MixApart with a warm cache, with the data scheduler logic, and without (MixApart-DS).

Caching and Scheduling: Figures 3(b) and 3(c) show job durations for various data reuse ratios. Job durations decrease with higher reuse ratios as a bulk of the data is fetched from local disks, avoiding the NFS bottleneck. The compute-aware data scheduler improves performance by scheduling data transfers just-in-time. Specifically, MixApart with 0.8 data reuse ratio matches the performance of Hadoop with all data in HDFS.

Feasibility Analysis: Furthermore, the results are consistent with the analysis presented in Section 2. For *I/O-intensive*, the NFS server with 1 Gbps of bandwidth can sustain 20 parallel tasks (map task I/O rates are 50 Mbps). With 0.8 data reuse ratio, 80 parallel tasks can use local disks and 20 parallel tasks can use the NFS server to achieve the same performance as Hadoop with all data in HDFS. For *CPU-intensive*, lower I/O rates allow MixApart to match stand-alone Hadoop performance starting with only 0.5 data reuse ratio.

5 Related Work

As the MapReduce paradigm becomes widely adopted within enterprise environments, the drawbacks of a purpose-built filesystem become glaring. Recent work has looked at techniques to interface Hadoop with enterprise filesystems [4, 14] as well as studied methods to provide enterprise storage features on top of existing MapReduce frameworks [10]. Others have studied the benefits of in-memory caching for MapReduce workloads [3, 16]. We leverage the insights of Ananthanarayanan et al. [3] to argue for a disk caching layer. In general, works that optimize performance for specific job classes [3, 16] can be layered on top of MixApart, just as they would be with frameworks such as Hadoop. We enable analytics on enterprise filesystems by leveraging a caching layer for acceleration without any changes to existing enterprise storage, while others [4, 14] modify enterprise filesystems (i.e., PVFS/GPFS) to support analytics workloads. In this sense, MixApart allows customers to leverage the infrastructure already deployed without additional hardware costs or system downtime.

6 Conclusions and Future Work

MixApart enables scale-out of data analytics, while allowing enterprise IT to meet its data management needs. MixApart achieves these goals by using a cache layer at the compute nodes and intelligent schedulers to utilize the shared storage efficiently. MixApart reduces job durations by 45% compared to the traditional *ingest-then-compute* method, while using one third of storage capacity. We are extending MixApart to support analytics across data centers, thus allowing customers to maintain the data on-premise and leverage clouds for processing.

References

[1] Amazon EC2. `http://aws.amazon.com/ec2`.

[2] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica. Disk-Locality in Datacenter Computing Considered Irrelevant. In *HotOS'11*.

[3] G. Ananthanarayanan, A. Ghodsi, A. Wang, D. Borthakur, S. Kandula, S. Shenker, and I. Stoica. PACMan: Coordinated Memory Caching for Parallel Jobs. In *NSDI'12*.

[4] R. Ananthanarayanan, K. Gupta, P. Pandey, H. Pucha, P. Sarkar, M. Shah, and R. Tewari. Cloud analytics: Do we really need to reinvent the storage stack? In *HotCloud'09*.

[5] Apache. Hadoop Project. `http://hadoop.apache.org`.

[6] Apache. Mahout. `http://mahout.apache.org`.

[7] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz. Energy Efficiency for Large-Scale MapReduce Workloads with Significant Interactive Analysis. In *EuroSys'12*.

[8] Y. Chen, K. Srinivasan, G. Goodson, and R. Katz. Design Implications for Enterprise Storage Systems via Multi-Dimensional Trace Analysis. In *SOSP'11*.

[9] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI'04*.

[10] B. Fan, W. Tantisiriroj, L. Xiao, and G. Gibson. DiskReduce: RAID for Data-Intensive Scalable Computing. In *PDSW'09*.

[11] S. Ghemawat, H. Gobiof, and S.-T. Leung. The Google File System. In *SOSP'03*.

[12] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. In *EuroSys'07*.

[13] D. A. Patterson, G. Gibson, and R. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *SIGMOD'88*.

[14] W. Tantisiriroj, S. Patil, G. Gibson, S. W. Son, S. J. Lang, and R. B. Ross. On the Duality of Data-intensive File System Design: Reconciling HDFS and PVFS. In *SC'11*.

[15] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay Scheduling: a Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. In *EuroSys'10*.

[16] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster Computing with Working Sets. In *HotCloud'10*.