Horme

Random Access Big Data Analytics

Guangchen Ruan, Beth Plale, Milinda Paithrage

RESEARCH CENTER

School of Informatics and Computing
INDIANA UNIVERSITY

Thanks to

HathiTrust analytics team at Indiana Uninversity:
Beth Plale
Yu (Marie) Ma
Milinda Paithrage
Guangchen Ruan
Zong Peng
Samitha Liyanage
Leena Unnikrishnan

Resource thanks for this work: Alfred P. Sloan Foundation, HathiTrust consortium, and NSF funded Extreme Science and Engineering Discovery Environment (XSEDE) project

Introduction

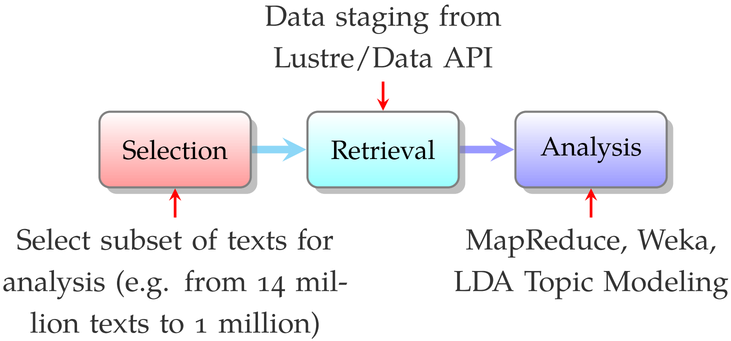
Motivation

HathiTrust: consortium providing stewardship of over **14 million** digitized books from research libraries in the US and beyond, 60% of which are in copyright.

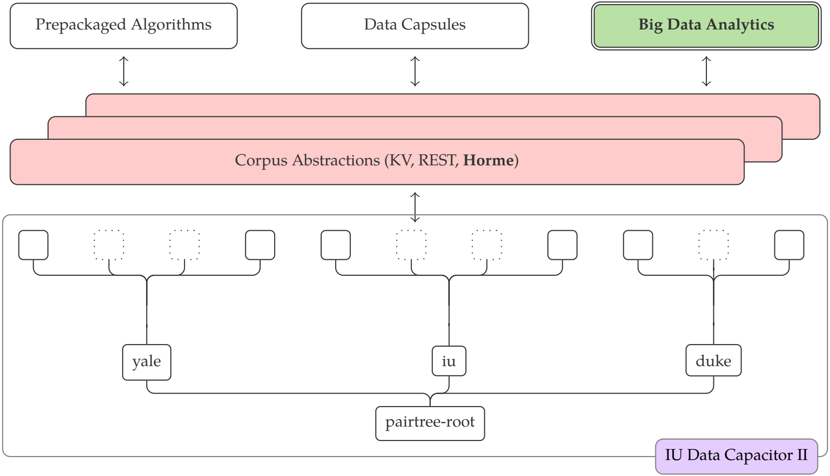
HathiTrust Research Center (HTRC) enables secure analytical access to the corpus.

Motivation

HTRC Analytics Workflow



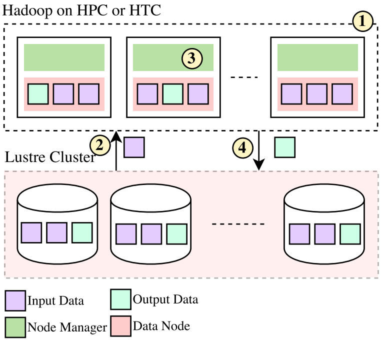
HTRC Analytics Infrastructure



Text Data Mining At HTRC

- General pattern:
  - Reduce 14 million texts to 1 million to analyze
  - Text storage is organized according to library that owns digitized book where access is by genre, subject, etc.
- HDFS performs poorly in random access use cases
- HBase good for random access, but needs to deploy external to HPC or HTC compute nodes due to transient nature of batch jobs
- HBase over Lustre is not optimal

Hadoop on HPC and HTC Environments



General workflow:

1. Setup Hadoop cluster
2. Partition and stage in blocks of digitized texts
3. Execute analysis algorithm
4. Stage out analytical results

Hadoop on HPC and HTC Environments

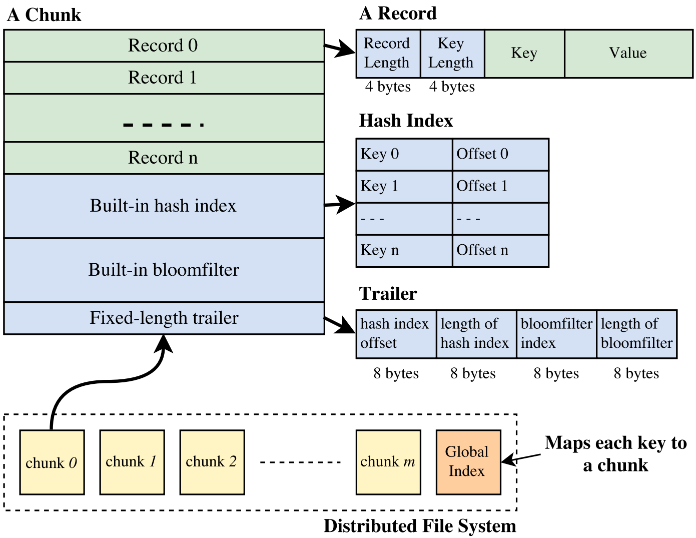
- Data is stored in *parallel file system, e.g., Lustre* connected to compute nodes via network
- Often data gets staged to scratch space in compute nodes (e.g. HDFS data nodes) from Lustre before actual computation
- Results get copied back to Lustre after job completion
- **High data staging overhead**
- HDFS on HPC and HTC environments limited by scratch space capacity of compute nodes

Horme

Horme

- Indexed binary file format for packing key/value pairs where size of value range from couple of kilobytes to couple of megabytes
- Set of tools for packing and managing key/value pairs
- Library for reading and writing indexed binary files
- Storage extensions for popular Big Data processing frameworks to read and write Horme binary files
- Not tied to any processing framework
- Works on top of any file system
- Delegates replication, file striping and fault-tolerance to underlying file system

Horme - BloomHashIndexFile



Horme - Programming Abstraction

- Reader

Scan	`for(Record r : BloomHashIndexFile)`
Random Access	`Record get(Key key)`
Membership Test	`boolean probablyHasKey(Key key)`
Accessors	`HashIndex getHashIndex()` `BloomFilter getBloomFilter()`

- Writer

Write	`void append(Key key, Value val)`
Misc.	`int getLength()`

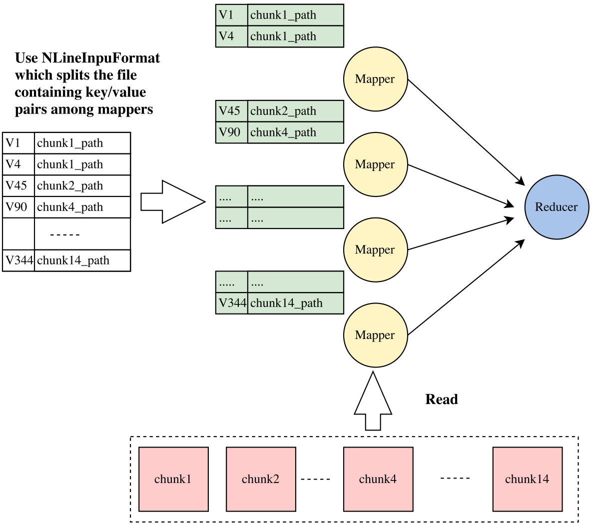
Horme - Distributed Packing Utility

- Pack key-value raw data into binary chunks
- Pleasingly parallel packing process
- For sake of load balancing each chunk should carry roughly same payload
  - Same # of records (e.g., simulation analysis task)
  - Same chunk file size (e.g., text mining task)

Horme - Parallel Processing Layer

- Hadoop input and output format extension on top of BloomHashIndexFile
- Retains MapReduce key-value semantics
- Supports both scan and random access
- Binary chunks can be served from network file system or HDFS cluster coupled with Hadoop deployment

Horme - Processing from Network Storage

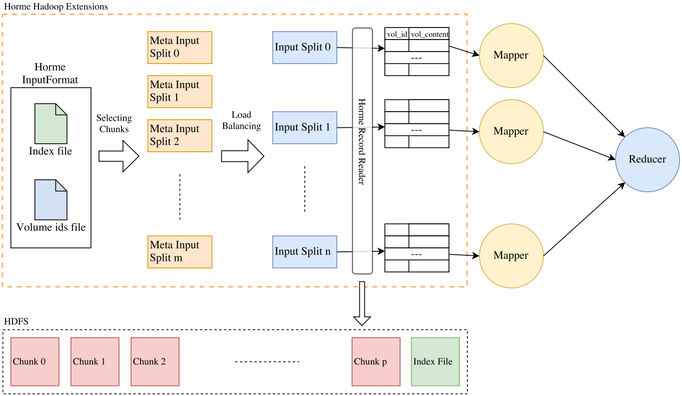


Horme - Processing from Network Storage

- With NLineInputFormat, a single input split consists of N consecutive lines
- `BloomHashIndexFile` `Reader` reads value corresponding to each key in input split at mapper
- Input sorted by binary chunk path so that `RecordReader` need only open a new file when next line is a new chunk
- Workload balanced on number of records processed
- Workload approximately balanced on size of records

Horme - Processing from HDFS

Horme Hadoop extensions determine input splits and convert input splits to key-value pairs for consumption by mappers

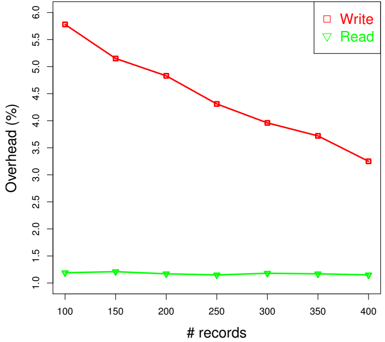


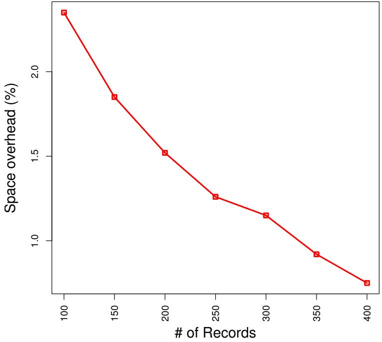
Experimental Evaluation

Experimental Evaluation Environment

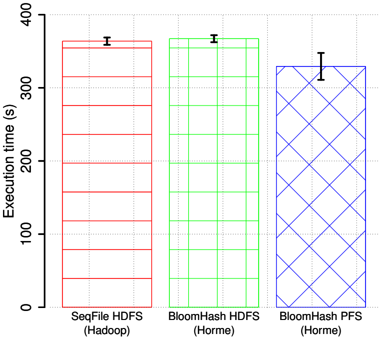
- Single node setting: 4-core 2.4 GHz Intel W3503 Xeon CPU, 8 GB RAM, 7200 RPM SATA drive
- Distributed node setting: each node is two Intel Xeon E5-2650 v2 8-core processors, 32 GB RAM, 180 GB local disk at 7200 RPM
- Each record has fixed sized key (100 byte) and value (2 MB)
- Hadoop v2.6.1
- 128 MB HDFS blocks with default replication factor of 3
- Lustre parallel file system
- Dataset size: 425,276 texts (digitized books)

Overhead of BloomHashIndexFile

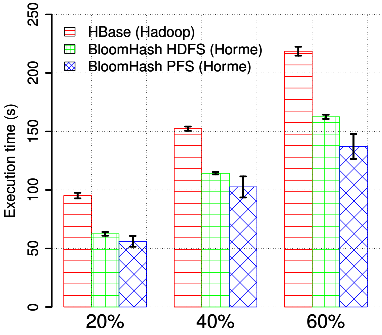




Horme vs Vanilla Hadoop

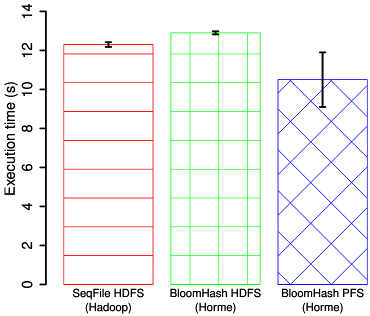


Scan/Sequential Access



Query/Random Access

Horme vs Vanilla Hadoop



Write

Observations

- Horme's "Random read from PFS" outperforms Hadoop's "Random read from HBase" by 41.1%, 32.7%, and 37.3% for 20%, 40% and 60% query setting, respectively.
- Horme superior to "Random read from HDFS" by 10.8%, 10.2% and 15.6% respectively in three settings

Related Work

- Fadika, Zacharia, et al. **"Mariane: Mapreduce implementation adapted for hpc environments."** 2011 IEEE/ACM 12th International Conference on Grid Computing. IEEE, 2011.
- Sehrish, Saba, et al. **"Mrap: A novel mapreduce-based framework to support hpc analytics applications with access patterns."** Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing. ACM, 2010.
- Dong, Bo, et al. **"A novel approach to improving the efficiency of storing and accessing small files on hadoop: a case study by powerpoint files."** Services Computing (SCC), 2010 IEEE International Conference on. IEEE, 2010.

Future Work

- Improve read performance through delayed record fetching when reading BloomHashIndexFile
- Evaluation is over 500,000 texts (books); future work is to evaluate in realistic setting of HTRC
- Horme's use in HTRC is for large-scale parallel pattern matching of n-gram (multi-word) terms on a large subset of HT corpus. Open question around packing strategy that reduces random access time for 80% of anticipated workload