A Review on Hadoop – HDFS Infrastructure Extensions

Kala Karun. A, Chitharanjan. K
Sree Chitra Thirunal College of Engineering
Thiruvananthapuram
kalavipin@gmail.com, chitharanjank@yahoo.com

Abstract— **Apache's Hadoop[1] as of now is pretty good but there are scopes of extensions and enhancements. A large number of improvements are proposed to Hadoop which is an open source implementation of Google's Map/Reduce framework. It enables distributed, data intensive and parallel applications by decomposing a massive job into smaller tasks and a massive data set into smaller partitions such that each task processes a different partition in parallel. Hadoop uses Hadoop distributed File System (HDFS) which is an open source implementation of the Google File System (GFS) for storing data. Map/Reduce application mainly uses HDFS for storing data. HDFS is a very large distributed file system that uses commodity hardware and provides high throughput as well as fault tolerance. Many big enterprises believe that within a few years more than half of the world's data will be stored in Hadoop. HDFS stores files as a series of blocks and are replicated for fault tolerance. Strategic data partitioning, processing, layouts, replication and placement of data blocks will increase the performance of Hadoop and a lot of research is going on in this area. This paper reviews some of the major enhancements suggested to Hadoop especially in data storage, processing and placement.**

Index Terms — **Distributed Computing, Hadoop, HDFS, Data Layout**

I. Introduction

Apache's Hadoop is an open source implementation of Google's Map/Reduce framework. It enables distributed, data intensive and parallel applications by decomposing a massive job into smaller tasks and a massive data set into smaller partitions such that each task processes a different partition in parallel. The main abstractions are 1. MAP tasks that process the partitions of a data set using key/value pairs to generate a set of intermediate results and 2. REDUCE tasks that merge all intermediate values associated with the same intermediate key. Hadoop uses Hadoop Distributed File System (HDFS) which is an implementation of the Google File System (GFS) for storing data.

The Hadoop Distributed File System (HDFS) [1] is a distributed file system designed to run on commodity hardware. Even though there are many similarities with existing distributed file systems, they are very much different. HDFS has a high degree of fault-tolerance and is developed to be deployed on low-cost hardware. HDFS provides efficient access to application data and is suitable for applications having big data sets.

HDFS cluster has master/slave architecture with a single Name Node as the master server which manages the file system namespace and regulates access to files by clients. The slaves are a number of Data Nodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS provides a file system namespace and allows user data to be stored in files. A file is partitioned into one or more blocks and these blocks are stored in a set of Data Nodes. The file system namespace operations like opening, closing, and renaming of files and directories are done by the Name Node. It also decides the mapping of blocks to Data Nodes. The Data Nodes are responsible for serving read and write requests from the file system's clients, block creation, replication and deletion upon instructions from the Name Node.

A large number of extensions to the existing Hadoop infrastructure have been proposed. Here some of the major extensions mainly focusing different data layouts, replication etc. are reviewed.

II. Hadoop-DB

Big enterprises are moving away from the traditional way of setting up the analytical databases in high end proprietary machines and moving towards cheaper commodity hardware and they need to analyze terabytes and peta bytes of data every day. There are arguments favoring two technologies for data analysis in these cases. The proponents of parallel databases believe that the performance and efficiency of parallel databases make them well suited for analytical processing. The other argument is that Map/Reduce based systems are more suitable because of their scalability, fault tolerance and flexibility to handle unstructured data. Both approaches have advantages as well as limitations. Parallel databases are not so good in fault tolerance and working with heterogeneous environment properties. In case of trade-off between fault tolerance and performance, parallel database selects performance extreme. The limitation with Map/Reduce is performance. In Map/Reduce, the user has to first model and load data before processing and this makes limited utilization of performance enhancing tools. Moreover, the traditional analytical data processing queries and reports are not well suited for one time query processing of Map/Reduce.

HadoopDB [2] is a hybrid system that has the best features of both parallel databases and Map/Reduce. HadoopDB approaches parallel database in performance and efficiency.

[1] http://hadoop.apache.org

At the same time it provides scalability, performance and flexibility of Map/Reduce.

In HadoopDB, multiple single node database systems are connected using Hadoop as the network communication layer and task co-ordinator. It achieves fault tolerance and the ability to operate in heterogeneous environments by taking over the scheduling and job tracking features of Hadoop and achieves the parallel database performance by executing most of the query processing inside the database engine.

A. Advantages

- ➢ HadoopDB can perform like parallel databases along with high fault tolerance, ability to run in heterogeneous environments and software license cost as Hadoop.
- ➢ It can cut down the data processing time especially on tasks that require complex query processing.
- ➢ It can yield scalability, fault tolerance and flexibility of Map/Reduce systems.

B. Limitations

- ➢ It forces users to use DBMS. Installing and configuring parallel DBMS is quite difficult.
- ➢ HadoopDB changes the interface to SQL. So the programming model is not as simple as Map/Reduce.
- ➢ HadoopDB locally uses ACID conforming DBMS engines which will affect the dynamic scheduling and fault tolerance of Hadoop.
- ➢ Intensive changes are needed to glue together Hadoop and Hive frameworks.

III. HADOOP ++

After identifying the limitations of HadoopDB, the research challenge that is tackled in [3] is to create a system that keeps the interface of Hadoop Map/Reduce, approaches parallel databases in performance and doesn't change the underlying Hadoop framework. To meet all these requirements Hadoop++ was developed. Hadoop++ is Hadoop with Trojan Index and Trojan Join. In terms of query processing Hadoop++ matches and improves the query run times of HadoopDB. A non intrusive approach is used in Hadoop++ without changing the underlying Hadoop framework. It changes the internal layout of a split, which is a horizontal partition of the data and writes appropriate user defined functions.

A. Trojan Index

Indexing capabilities can be achieved in Hadoop using Trojan index. The basic notion behind using Trojan index is that the schema and query workload is known already. The main features of Trojan index are non- invasiveness, no need to create a distributed SQL engine on top of Hadoop, optional index access path, possibility to create multiple index in the same split and seamless splitting. Trojan indexes are created during the data load time. The map function reads the input records of the relation and outputs key-value pairs. The key is a composite one which has split id and index attribute. The output is shuffled on split id. Reduce has an integral IndexBuilder function which creates the index structure. By this process, the internal representation of splits is changed i.e. a small index overhead and header/footer fields are added. The splits appended with the Trojan index are emitted and stored in the distributed file system.

Query processing algorithm loops over each split and obtains the key range covered by the index. If the search key matches with the index key, the index is loaded and the records having the search key are read. Otherwise the split will be passed over.

B. Trojan Joins

Trojan joins are also created during the data load time. Trojan join creation is similar to Trojan index creation. 'Map' iterates over splits of each relation and produces a composite key having split id, relation id and the join attribute value as the key. They are re-grouped and re-shuffled as in index creation. Then they are sorted on join key which give a set of records with the same join attribute value and will be passed to reduce. 'Reduce' creates one co-group per split.

The goal behind Trojan join is to avoid reduce phase since the data were already pre-partitioned. The algorithm access each input split and collects records of the same co-group.

C. Advantages

- ➢ Trojan index increases performance and parallelism since the map tasks processes the index independently. It also keeps the outside view of the block intact.
- ➢ Trojan join gathers the ability to process joins at the map side and thereby allowing Map/Reduce jobs to run faster even for large volumes of data.
- ➢ For selection queries and join operations, Hadoop++ runs faster than HadoopDB and Hadoop.
- ➢ No modification of Map/Reduce interface is required.

D. Limitations

- ➢ Arrival of new data or the modification of existing data necessitates the reorganization of Trojan index as well as Trojan join.
- ➢ Trojan index is heavily dependent on the block size. Increased block size leads to augmented index coverage.
- ➢ Similar to the DBMS, it is assumed that schema and query work load is known in advance.
- ➢ Static solution. Requires users to reorganize the input data.

IV. CO- HADOOP

CoHadoop [4] provides a solution for co-locating related files in Hadoop Distributed File System (HDFS). HDFS is extended to provide co-location at the system level. A new file property is used to identify related files and modify the placement policy of HDFS. This modification retains the benefits of Hadoop including load balancing and fault tolerance.

The default block placement policy provides load balancing through even data distribution across the Data Nodes. This policy works well for applications access and use single file.

Improvements in performance can be obtained for applications using data from multiple files using custom-made approaches. Co-Hadoop is one such approach which helps the applications to control data placement at the file-system level.

To achieve co-location, HDFS is extended with a new file level property called locator. There is N:1 relationship between files and locators. Each file is assigned to at most one locator and many files can be assigned to the same locator. Files with the same locator will be placed in the same set of Data Nodes. For files with no locator, default strategy is used. To manage locator information and to keep track of co-located files, a new data structure called locator table is introduced into the Name Node of HDFS. Locator table stores the mapping of locator to the set of files. It is maintained dynamically in main memory of Name Node and will be re-created when the Name Node is restarted. For the reconstruction of the locator table, the locator value of each file will be kept in its INode, which is a disk resident data structure that stores all file properties.

When a file f with locator value l is created, then the locator table will be searched to find an entry for l. If an entry for l is not present, a new entry (f, l) is created and the default data placement policy is used to select the Data Nodes to store the replicas. If an entry for l is present, the list of files with locator l will be accessed from the table. To replicate f, select ‘r’ Data Nodes from this list where r is the replication factor. CoHadoop guarantees perfect co-location if sufficient space is present in each Data Node and failures are nil.

A. Advantages

- Flexible compared to Hadoop++ and HadoopDB.
- Improves performance without affecting the dynamicity and fault tolerance of Hadoop and well suited for applications that continually consume new data.
- Pre-processing and loading cost is small compared to the savings of query time.

B. Limitations

- Slightly slow due to high network utilization.
- Indexing aspects are not improved and detailed knowledge of input data is required.

V. HAIL

HAIL (Hadoop Aggressive Indexing Library) [5] is an enhancement of HDFS and Hadoop Map/Reduce which significantly improve the performance of Map/Reduce tasks. It modifies the upload pipeline of HDFS to create different clustered indexes on each block replica. HAIL maintains the block replicas in different sort orders and different clustered indexes. So the chance of finding an appropriate index increases and improves the runtime performance. Indexes are created while uploading data to HDFS and hence additional data read is not required.

The basic idea behind HAIL is to create indexes on attributes of interest at load time with minimal changes to Hadoop. HAIL achieves this by modifying the data loading pipeline so that the index is created on each replica as it is loaded into HDFS. While Hadoop uses a pipeline that takes a block of data from the client and passes it without any change to the three replicas (default replication factor is 3) sequentially, HAIL buffers the block at each replica, sorts it based on the attribute which is indexed, create index for that block and flushes it to disk. During this process, HAIL also changes the data format of Hadoop’s standard plain text to PAX [6] binary. PAX is used here for effective column oriented grouping within each block. PAX takes up less space and hence the load time is improved. It can also improve query times by reducing the amount of data needed for projection when the operation is on a few attributes.

A. Advantages

- HAIL improves both upload and query times.
- Failover properties of Hadoop are not changed.
- HAIL works with existing Map/Reduce jobs incurring only minimal changes to these jobs.

B. Limitations

- HAIL can be very useful if selection is done on an attribute that has been indexed.
- Depending on the use case, PAX is suboptimal and PAX is not related to data compression.
- HAIL is not suited for processing of full text files since to create indexes some sort of schema is required.
- Jobs can use only one index at a time and ,memory requirements are more compared to standard Hadoop.

VI. ELASTIC REPLICA MANAGEMENT SYSTEM FOR HDFS (ERMS)

The data access patterns in HDFS vary heavily and the current replication strategy of creating a fixed number of replicas may be inadequate. Based on the data access patterns, data in Hadoop can be classified into different types. *Hot data* – data having a large number of concurrent access and high intensity of access, *cold data* - unpopular and rarely accessed data, *normal data* – rest of the data other than hot and cold. In a large and busy HDFS network, hot data will have concurrent and intense access. Replicating hot data only on three different nodes is not adequate to avoid contention. Also for cold data, three replicas may produce unnecessary overhead. To tackle these issues ERMS [7] is proposed which introduces an active/standby storage model which takes advantage of a high performance complex event processing engine to distinguish the real time data types and brings an elastic replication policy for the different types of data. It uses Condor [8] to increase the replication factor of hot data in standby nodes and to remove extra replicas of cold data. Erasure codes can be used to save storage space and network bandwidth when hot data changes to cold data.

A. Advantages

- Improves data locality by keeping more replicas of hot data and less replicas of cold data.
- Dynamically adapts to changes in data access patterns and data popularity and network overhead is less.

- Enhances the reliability and availability of data.

B. Limitations

- Efficiency depends on a number of threshold values and hence the careful selection of threshold values is needed.
- Memory requirement is high.

VII. TROJAN HDFS

In Trojan HDFS [9], a new data layout called Trojan Layout is used which internally organizes data blocks into different attribute groups according to the workload to improve the access performance. It creates the data layout without affecting the fault tolerance properties of Hadoop. Hadoop stores multiple replicas in different Data Nodes without changing the layout. Trojan HDFS creates different Trojan layouts for replicas and so the incoming requests can be forwarded to the replica with the most suitable layout.

Current Map/Reduce processes data in row oriented fashion which is having the limitation of reading the entire attributes even if only some of them are needed. Column stores also have some limitation like data blocks of different column may be in different nodes. So, a hybrid layout called PAX [6] can be used. The idea is to store same data in blocks as in row layout and column layout is used inside the blocks. Trojan layout is similar to PAX in that the same data is stored in blocks but internally any layout can be used. The goals of Trojan HDFS are (a) for an incoming query workload, find the right Trojan layout for the blocks which will improve the query performance, (b) keeps the interface unchanged and (c) zero administrative burden.

A. Advantages

- Trojan layouts are created and accessed in a way that is invisible to the users.
- Outside HDFS unit i.e. the data block is not changed. Only the internal representation is changed.
- No change in Map/Reduce processing pipeline is required and seamless query processing is achieved and can enrich Trojan layouts with standard DBMS optimizations.

B. Limitations

- Not adapted to changes in workload and index creation is expensive.
- Query performance improves only for the indexed attributes and can't improve I/O performance.

VIII. CHEETAH

Cheetah [10] is a scalable and flexible custom data warehouse built on top of Hadoop. It combines the benefits of both data warehouse and Map/Reduce technologies. It provides a simple query language which is easily understood by people having little SQL knowledge. It utilizes Hadoop's optimization techniques for data compression, access methods and materialized views and provides high performance by processing 1 GB of raw data per second. Virtual views are created on top of a star or snowflake schema which is the commonly used schema in data warehouses [11]. Since these virtual views are provided to the users for querying, there is no need to understand the underlying schema design and hence the query language is made simple.

HDFS has no provision to store data blocks having the same partition key in the same set of nodes. Cheetah assumes that big dimension tables permit insertion operation only or they have slowly changing dimensions and the queries require the snapshots of the dimensions [11]. So the big dimension tables are de-normalized and the dimension attributes are stored directly in the fact tables.

To support schema changes on big fact tables, schema versioned tables are provided. Each fact table row has a schema version id. Schema version table stores attribute information for each version. This makes schema changes very easy. Nested tables are also supported by Cheetah.

Cheetah supports row level security [12] based on virtual views and methods to anonymize data [13] which help the external clients to access and secure the data. It has a JDBC interface through which user can submit queries. It also provides a non-SQL interface through which data can be accessed with fine granularity. The queries will be processed by a query driver into a Map/Reduce job having a map phase and reduce phase plans. If the output size is very large, the user can write a Map/Reduce program to analyze the output.

The data is normally stored in columnar format and hence suitable for columnar data compression. The type of compression is determined dynamically based on the data type. In Hadoop, the number of map tasks required for a job is determined by the framework and the number of reducers is given directly given by the job which has significant impact on the performance. In Cheetah, the number of reducers is determined automatically based on some heuristics like the number of reducers should be proportional to the number of groups of columns in the query and the reducer count should be more if the 'group by' columns includes a column having a very large number of elements. Cheetah exploits materialized views [14] on their virtual view abstractions. Cheetah improves the access overhead of small queries (small input file size) by reading directly from HDFS and processing it locally.

A. Advantages

- Simple query language and high performance.
- Seamless integration of Map/Reduce and data warehousing concepts and efficient data compression.

B. Limitations

- Query reconstruction overhead and cannot guarantee that all fields of the needed record are present in the same Data Node.
- Identical layout for all replicas and no column grouping.
- Unnecessary column reads and I/O throughput is limited.

IX. RCFILE

RCFile [15] is a data placement structure introduced for Map/Reduce based data warehouses like HDFS. It combines the advantages of both row and column stores. Similarity with row stores is that data in a row will be present in the same node and hence the tuple reconstruction cost is less. As a column store it can perform data compression and can avoid unnecessary column read operations.

In RCFile, the basic unit of data is row group. All the records are partitioned into row groups. A row group contains sync marker, metadata header and table data. Sync marker indicates the beginning of a group, the metadata header contains metadata information about the group and table data is a column store. Metadata header is compressed using run length encoding and the table data will be compressed in parts using Gzip compression algorithm.

A. *Advantages*

- It has comparable data loading speed with row stores and is adaptable to workload changes and is read optimized since unnecessary column reads during table scans are avoided.
- It provides column wise compression and has efficient storage utilization.
- Unnecessary column reads can be avoided through independent column compressions and unnecessary decompressions through lazy decompressions.

B. *Limitations*

- Since the columns are highly interleaved in a single block, efficient I/O elimination will become difficult.
- Tuning row group size and I/O transfer size correctly is difficult and additional space overhead because of the metadata information about row groups.

X. DARE

Placing data near to computation enhances performance in data intensive applications. DARE [16] is an adaptive data replication mechanism that helps in achieving a high degree of data locality. DARE adapts to the change in workload conditions. Each node executes the algorithm to create replicas independently of heavily accessed files in a short interval of time. Data with correlated access are distributed to various nodes as new replicas are created and old ones expire which also enhance data locality. No extra network overhead is incurred since it is making use of existing remote data retrieval.

The algorithm creates replicas of popular files and at the same time minimizes the number of replicas of unpopular files. A greedy approach is used which incurs no extra network traffic since it is making use of existing remote data retrieval. Usually when a map task is launched the required data may be present locally or in a remote node. The remote data will be fetched and used without any local storage. But in DARE this remote data will be stored locally thereby increasing the replication factor by one. It also uses a replication budget to limit the storage consumed by the dynamically created replica.

An ElephantTrap [17], a mechanism to find the largest flows in a network link structure is used to replicate popular files. A probabilistic approach is also used for the aging mechanism.

A. *Advantages*

- Improves data locality with no extra network overhead.
- Turn around and slow down time is improved.
- It is scheduler agnostic, so it can be used in parallel with other schemes.

B. *Limitations*

- Storage requirements are high. More data structures and synchronization mechanisms are needed.
- Intense modification of Hadoop is required.

XI. CLYDESDALE

Clydesdale [18] is a research prototype built on top of Hadoop for structured data processing. It improves performance without making changes to the underlying platform. It inherits the scalability, fault tolerance and elasticity of Map/Reduce. For storing data, Clydesdale uses HDFS. Column oriented layout is used to store data and tries to co-locate the columns of a given row to the specified node. Map tasks are carefully designed so that the data structures used for query processing can be shared by multiple threads and allows multiple tasks to execute successively on any node.

Clydesdale can deploy multi core servers with huge memory capacity and employs n-way joins instead of generic 2-way joins. It can be used for workloads satisfying any generic schema but targeted especially for star schema. In star schema datasets the fact tables are larger than the dimension tables. Fact table and the master copy of all dimension tables are available in HDFS. Fact table is stored in Column Input Format (CIF) [19]. Each column of the table will be stored in separate HDFS file so that only files corresponding to the columns needed in the query can be read. This avoids un-necessary column reads. CIF also allows Hadoop to use location aware scheduling for map tasks. CIF uses Record Interface provided by the Avro Serialization Framework for Map/Reduce tasks.

A. *Advantages*

- Improves the performance of structured data.
- No change in the underlying framework is required.
- Inherits all the attractive features of Hadoop including scalability, elasticity and fault tolerance.

B. *Limitations*

- As of now, no SQL parser is present and the SQL queries are submitted as Map/Reduce programs in Java.
- Updates to dimension tables are not permitted.
- Scheduling Clydesdale workloads along with Map/Reduce loads pose network load management problems and parallelism may be limited for small data sets.

XII. Comparison of extensions

Comparison of Hadoop enhancements reviewed in the previous sections is given in Table I. Several factors like fault tolerance, scalability, data locality, load balancing, performance, load time interface change to Map/Reduce, changes to Hadoop framework, indexing and layout etc. are compared.

References

[1] K. Shvachko, H. Kuang, S. Radia, R. Chansler, "The Hadoop Distributed File System," 26th IEEE Symposium on Mass Storage Systems and technologies, Yahoo!, Sunnyvale, pp. 1-10, May 2010.

[2] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz and A. Rasin, "HadoopDB: An architectural Hybrid of Map/Reduce and DBMS Technologies for Analytical Workloads," Proceedings of VLDB Endowment, 2(1), pp.922-933, August 2009.

[3] J. Dittrich, J.-A. Quian´e-Ruiz, A. Jindal, Y. Kargin, V. Setty and J. Schad, "Hadoop++: Making a yellow elephant run like a cheetah (without it even noticing)," Proceedings of VLDB Endowment,3 (2), pp. 515-529, September 2010.

[4] M. Y. Eltabakh, Y. Tian, F. Özcan, R. Gemulla, A. Krettek, J. McPherson, "CoHadoop: flexible data placement and its exploitation in Hadoop," Proceedings of the VLDB Endowment, 4 (9), pp. 575-585, June 2011.

[5] J. Dittrich, J.-A Quian´e-Ruiz, S. Richter, S. Schuh, A. Jindal, J. Schad, "Only aggressive elephants are fast elephants," Proceedings of the VLDB Endowment, 5(11), pp 1591-1602, July 2012.

[6] A. Ailamaki, et al., "Weaving Relations for Cache Performance," Proceedings of VLDB'01, pp. 169–180, 2001.

[7] Z. Cheng et al., "An Elastic Replication Management System for HDFS," Workshop on Interfaces and Abstractions for Scientific Data Storage, IEEE Cluster 2012, 2012.

[8] D. Thain, T. Tannenbaum and M. Livny, "Distributed computing in practice: the Condor experience: Research Articles", Concurrency and Computation: Practice and Experience, 17(2), pp. 323-356, February 2005.

[9] A. Jindal, J.-A Quian´e-Ruiz, and J. Dittrich, "Trojan Data Layouts: Right Shoes for a Running Elephant", Proceedings of SOCC '11, Article No. 21, 2011.

[10] S. Chen, "Cheetah: A high performance, custom data warehouse on top of mapreduce," Proceedings of VLDB, 3(2), pp. 1459–1468, 2010.

[11] R. Kimball, The Data Warehousing Toolkit, John Wiley & Sons, New York, 1996.

[12] Q. Wang, T. Yu and N. Li, et al., "On the Correctness Criteria of Fine-Grained Access Control in Relational Databases", In Proceedings of VLDB, pp.555–566, 2007.

[13] L. Sweeney, "K-Anonymity: A Model for Protecting Privacy", International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 10(5), pp. 557–570, 2002.

[14] A. Y. Halevy, "Answering queries using views: A survey," The VLDB Journal — The International Journal on Very Large Data Bases, 10(4), pp.270-294, December 2001.

[15] H. Yongqiang, L. Rubao, H. Yin, N. Jain, Z. Xiaodong, X. Zhiwei, "RCFile: A fast and space-efficient data placement structure in Map/Reduce-based warehouse systems", Proceedings of 27th International IEEE Conference on Data Engineering (ICDE), pp.1199-1208, April 2011.

[16] L. Abad, Y. Lu and R. H. Campbell, "DARE: Adaptive Data Replication for Efficient Cluster Scheduling," Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER 11), pp. 159-168, 2011.

[17] Y. Lu, B. Prabhakar and F. Bonomi, "ElephantTrap: A low cost device for identifying large flows," Proceedings of IEEE Symposium on High-Performance Interconnects, HOTI '07, pp. 99-108, 2007.

[18] T. Kaldewey, E. J. Shekita, S. Tata, "Clydesdale: Structured data processing on MapReduce", Proceedings of the 15th International Conference on Extending Database Technology, pp. 15-25, 2012.

[19] A. Floratou, J. M. Patel, E. J. Shekita and S. Tata, "Column-Oriented Storage Techniques for MapReduce ", Proceedings of VLDB, 4(7), pp. 419-429, 2011.

Table I Comparison of Hadoop Infrastructure Extensions

	HadoopDB	Hadoop++	HAIL	Co-Hadoop	ERMS	Trojan HDFS	RCFile	DARE	Cheetah	Clydesdale
Fault tolerance	Almost same as Hadoop	less than Hadoop	Almost same as Hadoop	Same as Hadoop	Same as Hadoop	Same as Hadoop	Same as Hadoop	Same as Hadoop	Same as Hadoop	Same as Hadoop
Scalability	Better than parallel data bases	Highly scalable	Highly scalable	Highly scalable	Highly scalable	Highly scalable	Highly scalable	Highly scalable	Highly scalable	Same as Hadoop
Data locality	-	-	considered	considered	-	considered	-	considered	-	considered
Load time	More than Hadoop	More than HadoopDB	Less than Hadoop++	Less than Hadoop++	More than Hadoop	More than Hadoop	Less than column stores	More than Hadoop	More than Hadoop	Same as Hadoop
Performance and speed	Approaches to parallel DBMS	More than Hadoop	More than Hadoop++	More than Hadoop++	More than Hadoop++	More than Hadoop++	More than Hadoop	More than Hadoop	High	High
Map/Reduce Interface Change	Change to SQL interface	No change	No change	No change	No change	No change	No change	No change	JDBC as well as No- SQL interface	Record interface provided by Avro
Changes in Hadoop	Deep changes	No change	Minimum	Minimum	Minimum	Minimum	Minimum	Intense changes	-	No change
Indexing and layout	No indexing, Tabular as in RDBMS	Indexing, Trojan Index	Indexing, PAX	No indexing, default layout	No indexing, default layout	Indexing, Trojan Layout	No indexing, hybrid layout	No indexing, default layout	No indexing, default layout	No indexing, Multi –CIF format
Data Compression	PostgreSQL's data compression method	-	-	-	-	-	column compression	-	Column compression	Compressed using distributed cache mechanism