Project SnappyData™ - Community Edition

Release Notes

Software Release 1.1.0
May 2019

TIBCO®

Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND

CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

ANY SOFTWARE ITEM IDENTIFIED AS THIRD PARTY LIBRARY IS AVAILABLE UNDER

SEPARATE SOFTWARE LICENSE TERMS AND IS NOT PART OF A TIBCO PRODUCT. AS SUCH, THESE SOFTWARE ITEMS ARE NOT COVERED BY THE TERMS OF YOUR AGREEMENT WITH TIBCO, INCLUDING ANY TERMS CONCERNING SUPPORT, MAINTENANCE, WARRANTIES, AND INDEMNITIES. DOWNLOAD AND USE OF THESE ITEMS IS SOLELY AT YOUR OWN DISCRETION AND SUBJECT TO THE LICENSE TERMS APPLICABLE TO THEM. BY PROCEEDING TO DOWNLOAD, INSTALL OR USE ANY OF THESE ITEMS, YOU ACKNOWLEDGE THE FOREGOING DISTINCTIONS BETWEEN THESE ITEMS AND TIBCO PRODUCTS.

TIBCO, the TIBCO logo, the TIBCO O logo, TIBCO ComputeDB, SnappyData, and Snappy are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. Please see the readme.txt file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS.

CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

This and other products of TIBCO Software Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (https://www.tibco.com/patents) for details.

Contents

Overview

SnappyData™ is a memory-optimized database based on Apache Spark. It delivers very high throughput, low latency, and high concurrency for unified analytic workloads that may combine streaming, interactive analytics, and artificial intelligence in a single, easy to manage distributed cluster.

SnappyData offers a fully functional core OSS distribution, which is the **Community Edition**, that is Apache 2.0 licensed. The **Enterprise Edition** of the product, which is sold by TIBCO Software under the name **TIBCO ComputeDB™**, includes everything that is offered in the OSS version along with additional capabilities that are closed source and only available as part of a licensed subscription.

New Features

The focus of the 1.1.0 release was mainly stability at scale. However, the following feature is also included:

- **Support for Structured Streaming**
  The SnappyData structured streaming programming model is the same as Spark structured streaming. Additionally, TIBCO provides an in-built Sink to ingest data into SnappyData tables. The Sink supports idempotent writes, ensuring consistency of data when failures occur, as well as support for all mutation operations such as inserts, appends, updates, puts, and deletes. This feature is experimental in this release.

Performance/Scalability Enhancements

The following performance enhancements are included in SnappyData 1.1.0 release:

- **Scalability improvements for snapshot isolation** (c12fa66)
  Lower memory overhead and faster cleanup of old versioned data.

- **glibc malloc settings** (1942811)
  Default glibc malloc settings to reduce off-heap memory overhead.

- **Broadcast and exchange reuse** [SNAP-2789]
  Enabled broadcast and exchange reuse for column and row tables.

- **Partition pruning support for row table scan** [SNAP-2474]
  Added partition pruning for row tables. This is already available for column tables.

- **Performance improvement for limit query on external tables** (dd590a2)
  Limit query on external table used to scan all partitions earlier. Now it will not scan all partitions if results can be returned using only a few partitions.

- **Common-subexpression elimination for tokenized constants**
  Enabled common-subexpression elimination for tokenized constants. This improves performance of TPC-H Q19 among others. [SNAP-2462]

- **Enabled plan caching for hive thrift server sessions** [SNAP-2457]
  For better performance query plans are cached by default for hive thrift sessions.

Resolved Issues

The following issues are resolved in SnappyData 1.1.0 release:

- Start Hive Thrift server by default in the product. (f7ecc73)

- Implemented failover in ODBC. ODBC clients now have the failover functionality in case the server it is connected to goes down. [SNAP-2591]

- Fix connection to secure cluster via Spark's Thrift server. [SNAP-2751]

- Include SparkR library as part of distribution. (ce1874e)

- Set default value for **spark.sql.files.maxPartitionBytes** to 32 MB. This significantly reduces temporary heap usage when ingesting from external file sources. [SNAP-2962]

- Streaming micro batch thread keeps running forever when snappy job fails due to failure from outside streaming query. [SNAP-2909]

- **UI Enhancements and fixes**:
    - Provided a control on SnappyData Monitoring Console, to expand or collapse the whole row per member node entry in a single click. [SNAP-2900]
    - Improved GUI display of put into. [SNAP-2773]
    - On the UI, a column named **Overflown Size**/ **Disk Size** in tables. [SNAP-2602]
    - Display sparklines for CPU and memory usage for individual members. [SNAP-2908]
    - Changing default page size for all tabular lists from 10 to 50, sorting members list in tabular view on member type for ordering all nodes such that all locators first, then all leads and then all servers. [SNAP-2926]
    - Provide the user of SnappyData Monitoring console, a control over auto update. [SNAP-2661]
- A **--config <directory>** option can be passed to the **snappy-start-all** script now to take config files from that folder instead of default conf folder. **log4j.properties** file is still taken from the default conf folder. (5911532)

- Property is provided to set whether Hive meta-store client should use isolated ClassLoader. (b825fd6)

- Facility provided to store the temporary join result in off-heap for put into operation. (80f2d30)

- SQL expression alias in projection cannot be used in GroupBy. [SNAP-2237]

- Support fully qualified column names in projection. [SNAP-2440]

- Use Spark convention to return Catalog listDatabases/listTables output in lower-case. (41594f5)

- Quote table and schema name in some commands before executing on GemFireXD connection. This allows support for reserved keywords in GemFireXD parser such as, **default** as schema name. (5fade0b)

- Fixing some meta-data query inconsistencies: (5601184)
    - Add support for SHOW VIEWS.
    - Use **schemaName** for the column instead of Spark's **database** in SHOW TABLES.
    - Show CHAR/VARCHAR types instead of STRING for those types of columns in meta-data queries.

- Error in select if table(replicated) is created with backticks as delimiter. (91a9a63)

- Always route statements like **show tables/views...**, **describe** to lead from Snappy Shell to give consistent results. (1d5e0fc4)

- Fix for SHOW TABLES command. When the standard table type TABLE is given, then show all of row, column, external, sample, stream, and topK tables. (d0a1002)

- Fix for JDBC driver Jar running with Spark 2.3 + versions (java.lang.NoSuchFieldError: MAX_ROUNDED_ARRAY_LENGTH). (568b7e1)

- Mismatch in the expected and actual inserted rows, after inserting data from a column table to another column table. [SNAP-2902]

- Fix crash due to SEGV in putInto operations. [SNAP-2975]

- Avoid double free of page that caused server crash due to SIGABORT/SIGSEGV. [SNAP-2934]

- GemFireCacheImpl.oldEntryMap causes memory leak. [SNAP-2654]

- Fix for exception while startup - **MetaException: Metastore contains multiple versions**. [SNAP-2982]

- Exception occurs when the maximum column projection corresponds to 128th column position. [SNAP-2890]

- Fix for filter push down to scan level when **IN** list has constants but in cast node. (1630148)

- Key_columns option does not check for the validity of column names at time of table creation. [SNAP-2790]

- Some in built functions returning wrong results due to tokenization and plan caching. [SNAP-2712]

- Global lock to serialize concurrent puts. [SNAP-2381]

- NPE during lead failure/restart. [SNAP-2389]

- Fixes for multiple inconsistency issue in snapshot isolation. [SNAP-2985]

- Change the default auto reconnect setting to false. For locator it is still set to true (72052fc)

- Disable load-balance by default on servers. (be351df)

- Increase the weightage of lead to avoid it being thrown out of distributed system in case of network partitioning. [SNAP-2959]

- Use off-heap cache (if configured) for streaming-sink. [SNAP-2761]

- Wrap non-fatal OOME from Spark layer in LowMemoryException. [SNAP-2956]

- Handling the case when SnappyDataBaseDialect is used to determine table schema with the table name not containing schema name. [SNAP-2368]

- Fix backward compatibility issues with sample tables. (7a7e902)

- Trim the JOB_DESCRIPTION property in Spark jobs. [SNAP-2818]

- Planner is not used in IncrementalExecution. [SNAP 2634]

Known Issues

The following known issues are included in SnappyData 1.1.0 release:

Key	Item	Description	Workaround
SNAP-3014	PreparedStatement support is not complete.	• Some of the JDBC PreparedStatement API is not supported. Such as the batch prepared statement. • PreparedStatement on views is not supported yet.	Usually, prepared statement is used for performance reasons. SnappyData by default uses execution plan reuse and caching with constants in the query ignored for lookup of cached plan. So repeated statement execution already uses this and is very fast.
SNAP-1375	JVM crash reported	This was reported on: • RHEL kernel version: 3.10.0-327.13.1.el7.x86_64 • Java version: 1.8.0_121	To resolve this, use: • RHEL kernel version: 3.10.0-693.2.2.el7.x86_64 • Java version: 1.8.0_144
SNAP-1422	Catalog in smart connector inconsistent with servers	Catalog in smart connector inconsistent with servers when a table is queried from spark-shell (or from an application that uses smart connector mode) the table metadata is cached on the smart connector side. If this table is dropped from SnappyData embedded cluster (by using snappy-shell, or JDBC application, or a Snappy job), the metadata on the smart connector side stays cached even though catalog has changed (table is dropped). In such cases, the user may see unexpected errors like "org.apache.spark.sql.AnalysisException: Table `SNAPPYTABLE` already exists" in the smart connector app side for example for `DataFrameWriter.saveAsTable()` API if the same table name that was dropped is used in `saveAsTable()`	• User may either create a new Snappy Session in such scenarios. **Or** • Invalidate the cache on the Smart Connector mode. For example, by calling `snappy.sessionCatalog.invalidateAll(.`

SNAP-1153	Creating a temporary table with the same name as an existing table in any schema should not be allowed	When creating a temporary table, the SnappyData catalog is not referred, which means, a temporary table with the same name as that of an existing SnappyData table can be created. Two tables with the same name lead to ambiguity during query execution and can either cause the query to fail or return wrong results.	Ensure that you create temporary tables with a unique name.

SNAP-1753	TPCH Q19 execution performance degraded in 0.9	A disjunctive query (that is, query with two or more predicates joined by an OR clause) with common filter predicates may report performance issues.	To resolve this, the query should be rewritten in the following manner to achieve better performance:
			`select sum(l_extendedprice) from LINEITEM, PART where ( p_partkey = l_partkey and p_size between 1 and 5 and l_shipinstruct = 'DELIVER IN PERSON' ) or ( p_partkey = l_partkey and p_brand = 'Brand#?' and l_shipinstruct = 'DELIVER IN PERSON' )`
			
			`select sum(l_extendedprice) from LINEITEM, PART where ( p_partkey = l_partkey and l_shipinstruct = 'DELIVER IN PERSON') and ( p_size between 1 and 5 or p_brand = 'Brand#3')`

SNAP-1911	JVM crash reported	This was reported on: • RHEL kernel version: 3.10.0-327.13.1.el7.x86_64 • Java version: 1.8.0_131	To resolve this, use: • RHEL kernel version: 3.10.0-693.2.2.el7.x86_64 • Java version: 1.8.0_144
SNAP-1999	JVM crash reported	This was reported on: • RHEL kernel version: 3.10.0-327.13.1.el7.x86_64 • Java version: 1.8.0_131	To resolve this, use: • RHEL kernel version: 3.10.0-693.2.2.el7.x86_64 • Java version: 1.8.0_144
SNAP-2017	JVM crash reported	This was reported on: • RHEL kernel version: 3.10.0-514.10.2.el7.x86_64 • Java version: 1.8.0_144	To resolve this, use: • RHEL kernel version: 3.10.0-693.2.2.el7.x86_64 • Java version: 1.8.0_144
SNAP-2436	Data mismatch in queries running on servers coming up after a failure	Data mismatch is observed in queries which are running when some servers are coming up after a failure. Also, the tables on which the queries are running must have set their redundancy to one or more for the issue to be observed.	This issue happens due to Spark retry mechanism with SnappyData tables. To avoid this issue, you can stop all the queries when one or more servers are coming up. If that is not feasible, you should configure the lead node with `spark.task.maxFailures = 0`;
SNAP-2381	Data inconsistency due to concurrent putInto/update operations	Concurrent putInto/update operations and inserts in column tables with overlapping keys may cause data inconsistency.	This problem is not seen when all the concurrent operations deal with different sets of rows. You can either ensure serialized mutable operations on column tables or these should be working on a distinct set of key columns.

SNAP-2457	Inconsistent results during further transformation when using snappySession.sql() from jobs, Zeppelin etc.	When using snappySession.sql() from jobs, Zeppelin etc., if a further transformation is applied on the DataFrame, it may give incorrect results due to plan caching.	If you are using Snappy Jobs and using snappySession.sql("sql string") you must ensure that further transformation is not done. For example:
			`val df1 = snappySession.sql("sql string")`
			`val df2 = df1.repartition(12) // modifying df1`
			`df2.collect()`
			The above operation gives inconsistent results, if you are using df2 further in your code. To avoid this problem, you can use snappySession.sqlUncached("sql string"). For example:

			val df1 = snappySession.sqlUncached("sql string") val df2 = df1.repartition(12) // modifying df1 df2.collect()