Replication at the Speed of Change – a Fast, Scalable Replication Solution for Near Real-Time HTAP Processing

Dennis Butterstein Daniel Martin
Knut Stolze Felix Beier
IBM Research & Development GmbH
Schönaicher Strasse 220
71032 Böblingen, Germany
dennis.butterstein@ibm.com
{danmartin,stolze,febe}@de.ibm.com

Jia Zhong Lingyun Wang
IBM Silicon Valley Lab
555 Bailey Ave
San Jose, CA 95141, United States
jia.zhong@ibm.com
wlingyun@us.ibm.com

ABSTRACT

The IBM Db2 Analytics Accelerator (IDAA) is a state-of-the art hybrid database system that seamlessly extends the strong transactional capabilities of *Db2 for z/OS* with the very fast column-store processing in *Db2 Warehouse*. The Accelerator maintains a copy of the data from *Db2 for z/OS* in its backend database. Data can be synchronized at a single point in time with a granularity of a table, one or more of its partitions, or incrementally as rows changed using replication technology.

IBM Change Data Capture (CDC) has been employed as replication technology since IDAA version 3. Version 7.5.0 introduces a superior replication technology as a replacement for IDAA's use of CDC – *Integrated Synchronization*. In this paper, we present how *Integrated Synchronization* improves the performance by orders of magnitudes, paving the way for near real-time Hybrid Transactional and Analytical (HTAP) processing.

PVLDB Reference Format:
Dennis Butterstein, Daniel Martin, Knut Stolze, Felix Beier, Jia Zhong, Lingyun Wang. Replication at the Speed of Change – a Fast, Scalable Replication Solution for Near Real-Time HTAP Processing. *PVLDB*, 13(12): 3245-3257, 2020.
DOI: https://doi.org/10.14778/3415478.3415548

1. INTRODUCTION

IBM Db2 Analytics Accelerator (IDAA)[1] is an enhancement of *Db2 for z/OS* (Db2z) for analytical workloads. The current version of IDAA is an evolution of IBM Smart Analytics Optimizer (ISAO)[17] which was using the BLINK in-memory query engine[14, 3]. In order to quickly broaden the scope and functionality of SQL statements, the query engine was first replaced with Netezza (IBM PureData System for Analytics[2]). Netezza's design is to always use table scans on all of its disks in parallel, leveraging FPGAs to apply decompression, projection and filtering operations before the data hits the main processors of the cluster nodes. The row-major organized tables were hash-distributed across all nodes (and disks) based on selected columns (to facilitate co-located joins) or in a random fashion. The engine itself is optimized for table scans; besides Zone Maps there are no other structures (e. g., indices) that optimize the processing of predicates. With the advent of IDAA Version 6, Netezza was replaced with *Db2 Warehouse* (Db2wh) and its column-store engine relying solely on general purpose processors. Various minor deviations between Db2z and Netezza were removed that way, which resolved many SQL syntax and datatype incompatibilities faced by our customers.

Figure 1 gives an overview of the system: the Accelerator is an appliance add-on to Db2z running on an IBM zEnterprise mainframe. It comes in form of a purpose-built software and hardware combination that is attached to the mainframe via (redundant) network connections to allow Db2z to dynamically offload scan-intensive queries. The Accelerator enhances the Db2z database with the capability to efficiently process all types of analytical workloads, typical for data warehousing and standardized reports as well as ad-hoc analytical queries. Furthermore, data transformations inside the Accelerator are supported to simplify or even avoid separate ETL pipelines. At the same time, the combined hybrid database retains the superior transactional query performance of Db2z. Query access as well as administration use a common interface – from the outside, the hybrid system appears mostly like a single database.

Table maintenance operations (e. g., reorganization and statistics collection) are fully automated and scheduled autonomically in the background. An IDAA installation inherits all System Z attributes known from Db2z itself: the data is *owned* by Db2z, i. e., security and access control, backup, data governance etc. are all managed by Db2z itself. The Accelerator does not change any of the existing procedures or violate any of the existing concepts. As a result, the combination of Db2z and the Accelerator is a

[1] http://www.ibm.com/software/data/db2/zos/analytics-accelerator/

[2] http://www.ibm.com/software/data/puredata/analytics/system/

Proceedings of the VLDB Endowment, Vol. 13, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3415478.3415548

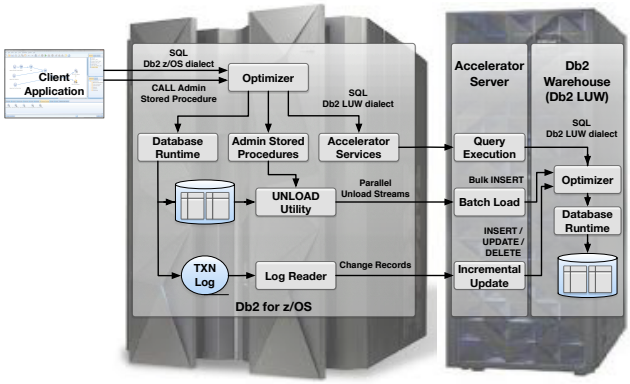

Figure 1: IDAA Architecture

hybrid system that consists of two architecturally very different databases systems: a traditional RDBMS that has its strengths at processing highly concurrent OLTP workloads, using well-established serialization and indexing concepts – and the Accelerator engine available for processing complex queries.

Both systems not only complement each other, but can also shield each other from workloads that may have a negative impact like exhaustive use of resources if dispatched to the less suitable system. For these query routing decisions, the Db2z optimizer implements a heuristic that decides if a given query should be processed by Db2z itself or if it should be offloaded to the Accelerator.

It is very important to understand that nothing has to be changed on existing applications that are already using Db2z in order to take advantage of the acceleration: they are still connecting only to Db2z and use its specific SQL dialect. Deep integration into existing Db2z components ensures that only little specific training is required to operate the Accelerator. Db2z remains the owner of the data; data maintenance, backup and monitoring procedures do not change.

Because IDAA operates on copies of the tables in Db2z, any changes to these tables must also be reflected on the Accelerator. This is normally done by running the IDAA `ACCEL_LOAD_TABLES` stored procedure that refreshes either an entire table or the changed (horizontal) partitions of that table on the Accelerator. This refresh mechanism matches many use-cases of IDAA, as analytical systems are often updated by ETL jobs that run in batch mode so that changes are applied in bulk on a scheduled basis.

Another use case, however, is to allow Accelerator-based reporting over "online" data, providing a solution that combines the features of traditionally separated OLTP vs. reporting systems that are connected by aforementioned ETL jobs. For such scenarios, the table or partition granularity of the refresh mechanisms is too coarse: if the total size of the changes is rather low but the changes themselves are spread over multiple partitions (or the tables themselves are not partitioned), then re-loading the majority of partitions or an entire table implies a lot of unnecessary work. Also, the practical minimum latency that can be achieved by running the aforementioned refresh procedures is too long; it is impractical to run the stored procedures every few minutes. Therefore, IDAA offers its "Incremental Update" feature which refreshes the copy tables on the Accelerator by asynchronously monitoring the Db2z transaction log for changes. Completed and committed transactions are replicated to the Accelerator. For efficiency reasons, those changes are applied in "micro batches" that typically contain data from all transactions that committed during a certain time interval (e. g., the last 60 seconds). This technology was implemented by the existing replication product called *IBM Infosphere Change Data Capture (CDC)*[3].

Integrated Synchroniziation (InSync) is the successor for CDC in the context of IDAA. Since it is not a general purpose replication product, it is much more light-weight and specifically tailored to the Accelerator. Installation and administration of InSync is significantly reduced, while replication performance exceeds CDC.

2. RELATED WORK

Relational database system vendors have been focusing on analytical DBMS appliances for a while; popular examples are Oracle Exadata[4], the Teradata appliance[5], EXASOL's EXASolution Appliance[6], and SAP's HANA[7] [7, 13]. Similarly, appliances running MapReduce implementations together with an analytical DBMS have become increasingly popular, sometimes under the name of "Big Data Analytics Platform". They use MapReduce programs for analytics on unstructured data and add additional ETL flows into an analytical DBMS that runs on the same appliance for reporting and prediction on cleansed, structured data. Essentially these systems promise an all-in-one solution for data transformation and analytics of high-volumes of structured and unstructured data. Examples are the EMC Greenplum DCA[8] and the "Aster Big Analytics Appliance"[9].

Common to all of these products is a shared nothing architecture that hash-partitions the data of each table and distributes it across the available compute nodes of the cluster. There is a strong focus on optimizing long-running, scan-heavy queries and analytical functions. Data access is mostly read-only and data modifications are done in large batches through bulk load interfaces. The distributed, shared nothing architecture makes these systems unsuitable for OLTP with a mixed read/write workload and the requirement for very low response times. Clearly, it is impossible to achieve the short access path and thus the low response times of a traditional OLTP architecture because these systems require a query to go through all nodes of a cluster for scanning a table and a coordinator node to combine intermediate results before synchronizing and possibly re-distribute work at runtime. By design, these systems have a minimum response time of several tens or hundreds milliseconds, as opposed to OLTP systems where the lowest response times typically are in the single digit milliseconds range.

[3] http://www.ibm.com/software/data/infosphere/change-data-capture/
[4] http://www.oracle.com/us/products/database/exadata/overview/index.html
[5] http://www.teradata.com/data-appliance/
[6] http://www.exasol.com/en/exasolution/data-warehouse-appliance.html
[7] http://www.sap.com/solutions/technology/in-memory-computing-platform/hana/overview/index.epx
[8] http://www.greenplum.com/products/greenplum-dca
[9] http://www.asterdata.com/product/big-analytics-appliance.php

This is the reason why OLTP and analytics require separate software and hardware architectures. IDAA is no exception to this – it also comes in form of two fundamentally different systems. However, IDAA is based on the notion of an *Analytics Accelerator*, a transparent attachment to an OLTP system that automatically classifies analytical workloads and chooses the "best" platform for the work to be processed. In contrast to other systems, Db2z and the Accelerator are tightly integrated so that users experience a single coherent system in terms of usage, monitoring, and administration.

There are several proposals of architectures in the research community that resemble the hybrid design of IDAA: [1, 2] propose an RDBMS that uses the MapReduce paradigm for the communication and coordination layer and a modified PostgreSQL database for the individual nodes. In [4], a review of the feasibility of architectures and implementation methods to achieve real-time data analysis is presented, [5] proposes a theoretical architecture for a hybrid OLTP / OLAP system. [15] proposes a hybrid row-column store to avoid having to operate on duplicate data and deal with consistency and data coherence problems. HYRISE [9, 8] is an in-memory column store that automatically partitions tables based on data access patterns to optimize for cache locality. OLTP-style access yields to partitions with a higher number of columns vs. analytical access that yields to smaller partitions.

In contrast to these approaches that all propose a single, unified system architecture for a hybrid DBMS, we believe that the required properties for OLTP and analytics are fundamentally different and in some cases even mutually exclusive as noted in Section 1. The transactions costs per query (network I/O, scheduling, etc.) as exposed by the different designs are so different that we believe that a single system can never fit analytical and OLTP access patterns equally well. As a result, our approach is to implement separate systems, using the OLTP system at the front (so that the performance characteristic of OLTP workloads are not negatively impacted) and to use query routing (somewhat comparable to what has been proposed in [6]) and data synchronization mechanisms to integrate the analytics system. Of course, this means that the data may not be perfectly in sync, but IDAA provides several mechanisms to achieve coherency at the application level, i. e., although there are temporary inconsistencies, applications will not notice this because the refresh mechanisms have been chosen and integrated into the application's data loading mechanisms (see Section 3.3 for details). Besides IDAA, some systems following an approach with separated OLTP and OLAP systems have been proposed [11, 10] with BatchDB being closely related to the presented approach. InSync additionally includes inter- and intra-transaction optimizations speeding up replication and does not execute OLAP queries on one data snapshot only. Instead, issued HTAP queries will individually be handled to fulfill per-query data freshness requirements.

3. DESIGN AND OPERATION OF INCREMENTAL UPDATE

The following section describes the architecture of the IDAA incremental update based on InSync. Several significant aspects are discussed, which lead to the replacement of CDC.

3.1 High-Level Replication Architecture

All replication products have two major components: (1) a *Capture Agent* that processes the transaction logs of the source database system, and (2) an *Apply Agent* that processes the changes from (1) and applies them to the target database system.[10] Figure 2 denotes these as *Db2z Log Reader* and *Integrated Synchronization*, respectively.

Contrary to CDC, the Log Reader does not actively process the transaction log. Instead, it is the Integration Synchronization component that requests a range of log records actively. Another noticeable difference of the Log Reader is that it does not try to filter log records for committed transactions. All the transaction-related processing as well as forming the micro batches and applying them to the target tables is handled by the Integrated Synchroniziation component on the target side.

Similar to CDC, InSync has to deal with heterogenous database systems. Db2z and Db2wh use different page formats and different structures in the transaction log. Therefore, commonly known techniques for log shipping in homogenous environments cannot be adopted, at least not without a significant amount of effort.

3.2 Management Interface

Administration of InSync is fully handled through the IDAA Server, providing a seamless user interface. It receives requests whether to enable/disable replication for a specific table, or to start/stop the replication process itself. Such user-driver requests as well as other internal requests are passed on to Integrated Synchronization.

Internal requests are needed because InSync operates directly on the underlying database. Concurrent operations of the IDAA Server – like schema manipulations or bulk loads – have to be synchronized. Many of the IDAA Server's actions that manipulate data are only available when replication is suspended for the affected tables for the duration of the operation.

3.3 Integration with Bulk Load

An important aspect is the integration of InSync with IDAA's bulk-LOAD mechanism. InSync itself does not handle the transfer of an entire table, e. g., after it has been first defined – bulk load is specialized and highly tuned for that. Therefore, IDAA allows to combine both mechanisms in the following fashion:

1. IDAA LOAD is used for the initial transfer of the full table data from Db2z to the Accelerator. During the processing of the LOAD, InSync remembers the current write position in the Db2z transaction log, i. e., it sets a *capture point*.

2. Subsequently, InSync uses this capture point as start position for requesting transaction log records from the

[10] Additional components may be used, for example, to manage meta data about the table mappings, access control, etc.

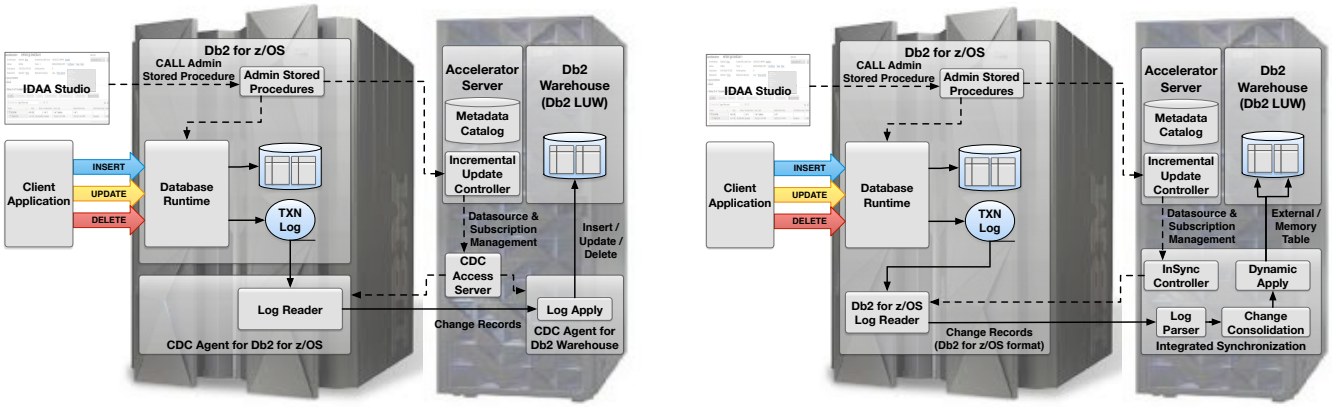

Figure 2: Architecture of the *Incremental Update* Feature

Log Reader. Thus, it obtains all changes on the Db2z tables resulting from inserts, updates and deletes.

3. When large scale changes for the table occur or the table is modified by Db2z utilities that bypass the transaction log, the administrator can reload the table via the bulk load, which temporarily suspends replication. Replication must remain suspended while the table is loaded to avoid the risk of applying an update twice (once as part of the IDAA load and again via InSync).

4. Once the table has been synchronized, replication is reactivated so that all changes occurring after the new capture point can again be replicated by CDC.

Note that a table is still fully enabled for writes in Db2z during bulk-load. Such changes are spilled, and applied by replication in a special apply mode when the load phase finished. This special apply mode merges the spill queue for the table with the changes that came from the bulk load, and makes sure to apply row changes not captured by load, and to ignore them otherwise.

This way, IDAA offers a best-of-both-worlds approach where log-based replication is used for continuous propagation of small, dispersed changes while efficient full table or partition-level transfer is used after bulk changes or operations that bypass the log. The bulk load can be applied to reload the *full table*, or only a selected set of partitions. In the second case, it is enforced that change detection is used to identify at least all those partitions that must be reloaded. If no clear decision can be made whether a partition was changed or not, it is reloaded in order to guarantee data consistency. Db2z collects run-time statistics on partition-level already, and that is exploited to make those decisions.

To actually enable this level of integration, InSync is aware of the concept of table partitions in Db2z, which is the granularity on which bulk load operates. In order to map the partition granularity from the Db2z table to the table in Db2wh, the table's schema is augmented by the Accelerator with a hidden column used to record the partition ID for each row. The partition ID is not using a 1:1 mapping; instead, the rows of one partition from the Db2z side may occur multiple times (e. g., due to multiple subsequent bulk loads) in the corresponding Db2wh table using different partition ID values. IDAA controls the data access such that only those rows from the most recent bulk load are visible for new queries. Rows from previous bulk loads are kept until all queries accessing them are finished, and then those rows are purged physically. Thus, IDAA implements a variant of multi-version concurrency control to enable concurrent query execution and bulk loads.

As a consequence, InSync needs to maintain the hidden partition ID column while updates are being propagated from Db2z to Db2wh. The partitioning information for each modified row can be deduced from Db2z transaction log information already. However, the mapping applied by IDAA's bulk load is *not* considered by InSync as the synchronization of this mapping incurs a too high performance penalty. InSync does apply a 1:1 mapping, and IDAA is aware of that and applies special treatment for replicated rows in subsequent bulk loads. [16]

3.4 Integration with Query Processing

For the query processing, the introduction of incremental update means literally no change. The decision to execute queries in the Accelerator is based only on the availability of data and not on its currency, so the query offload heuristics are not affected. As transactions are continuously propagated from the source to the target database, they are isolated from concurrently running queries by the normal isolation mechanisms in the target database. Of course, the fact that data continuously changes shows up at the application level, e.g. running the same analytic report twice on the Accelerator may yield different results – the same way as if the report query had been executed twice on the source system.

3.5 Where is the problem then?

Why invest effort to develop a high speed, low latency replication system despite having a proven solution in the field? Why would any customer take the step to migrate from CDC to InSync? Although IDAA presents a transparent solution for applications, stale data is not acceptable for a lot of use cases. For example, a credit card fraud detection system does not protect from fraudulent transactions. For such kind of applications, being able to run queries on guaranteed current data is key.

The guarantee IDAA gives is that any change committed in the source database before a query started will be seen when the query is offloaded to the Accelerator. To implement this guarantee, a delay protocol is employed. It works in combination with micro-batching of captured changes. As a result, the query may see changes from later commits, but it will never miss any commit that happened before the query started:

1. Db2z optimizer offloads eligible queries to the IDAA-Server, along with the position in the transaction log for the commit of the last change that happened to any of the referenced tables.

2. IDAA-Server blocks the query and sends a request to InSync containing the position in the transaction log along with a maximum wait time.

3. Upon each batch commit, InSync checks whether a particular query request can be satisfied (request transaction log position <= current transaction log position) with the current data. If that is the case, an answer indicating success is send to IDAA-Server which in turn unblocks the query and executes it in the backend database. Otherwise, a check is made whether the maximum wait time has been exceeded already. If so, the query is either cancelled or failed back to Db2z for execution.

Consequently, an application issuing a query in above mode does not have to cope with stale data. In the worst case it might have to "buy" current data by waiting for the specified amount of time – but when the query is executed, it's guaranteed to get at least the database state valid when the query was submitted to Db2z by the client application. This delay protocol implements what is commonly referred to as *Hybrid Transactional and Analytical Processing* – OLTP and OLAP processing united in only one database system.

How expensive is it to employ such a system? How large has the timeout to be set depends on the latency between source and target databases. This is a key differentiator between the CDC and InSync. Significant throughput increases and decreasing latency (compare 9) with InSync prepares the ground for true HTAP capabilities. Whereas with CDC high latencies of hours were not uncommon, InSync targets latencies in a range of minutes or even seconds. Most applications are able to cope with queries taking a couple of more seconds or minutes to execute – queries running for hours are inappropriate. Please note that the described protocol does not affect replication throughput either. In contrast to CDCs approach, InSync does not split batches to check if a query request can be satisfied as early as possible at the cost of replication throughput. Instead, InSync only checks for satisfiable queries after a complete batch has been applied to the target database. This way the delay protocol has (almost) no effect on the speed of change on the target database. The only additional work after a batch has been committed is to check if the transaction log position of a query is <= the current transaction log position which is updated when a batch is completely applied to the target database. There may be 100s of queries queries in the system, and said comparisons have to be conducted for each of those queries in the worst case.

4. THE CAPTURE AGENT ON A DIET

The Change Data Capture (CDC) Capture Agent responsible for reading the Db2z transaction log exhibits drawbacks in its daily use. It requires a significant amount of time and knowledge to set it up and maintain the separate process. Reading, extracting and preparing the log records for transmission to the target system is an expensive process and consumes significant CPU resources. InSync addresses these drawbacks. The following section describes in more detail how we achieve that.

4.1 Change Data Capture: Capture Agent

Using CDC imposes the following requirements on the user. Complex log record processing: Today CDC's architecture requires the capture agent to do 6 processing steps before the data is ready to be transmitted to the target (cf. Figure 3):

1. Read log records, decrypt: To begin with, we need to read the data from the storage. In the case of IDAA, we operate in a secure environment – data is stored encrypted. As the capture agent needs to read the decrypted contents of a log record for subsequent steps, the read data has to be decrypted.

2. Filter, merge, decompress: Now that we have decrypted data ready, the capture agent has to decide whether the log record data has to be transmitted or not (filtering). The read log record might not belong to a replicated table. In this case it is discarded and the capture agent continues reading through the transaction log. The capture agent might operate in a distributed environment (Db2z data sharing[11]) imposing the necessity to merge log records received from different data sharing members. Merging means the records have to be arranged in log sequence order to maintain correctness on the target. To make a log record data readable, it has to be to decompressed before further processing. Log records in the Db2z transaction log are written in the same way as they are written on the table pages. Nowadays, this means – at least in most cases – that data is written compressed to the transaction log.

Steps 1 and 2 are not explicitly done by the capture agent. In fact they are done by the facility providing access to transaction log. As a consequence, those steps cannot be eliminated. The following steps are necessary due to the platform independent design of CDC:

[11] https://www.ibm.com/support/knowledgecenter/en/SSEPEK_12.0.0/dshare/src/tpc/db2z_introdatasharing.html

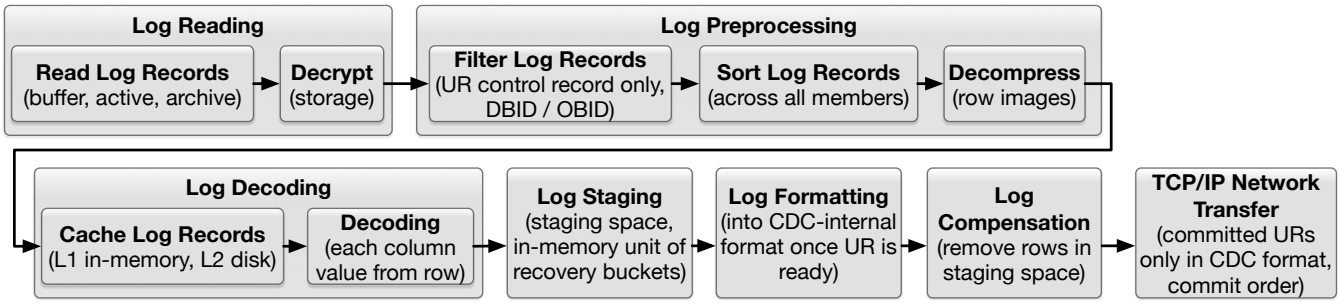

Figure 3: CDC Capture Agent pipeline

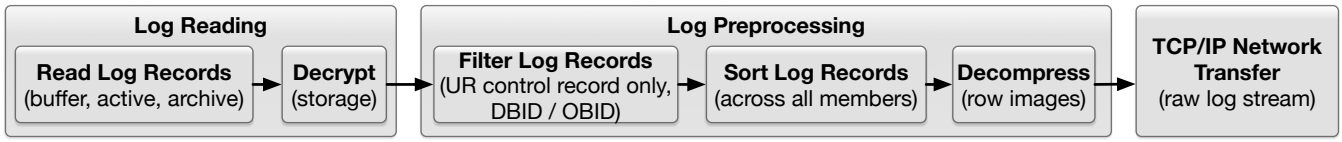

Figure 4: Log Reader pipeline

3. Read data must be decoded. This means, it has to extract every column from a log record in Db2z page record format (reordered row format, basic row format). More precisely, it has to convert the source's internal row format to actual column formats, e. g., text or decimal numbers. CDC employs a canonical, intermediate data format to transfer data from source to target. This has to be done for every single column of every single row. In fact, processing overhead is even worse. Update log records contain a before image as well as an after image and therefore more than the actual amount of columns in a table row have to be decoded. These operations are very expensive and sum up quickly.

4. Stage rows: The decoded rows are collected in UR (Unit of Recovery) buckets in the staging space. Usually, these buckets contain a whole set of transactions. UR buckets are kept in the staging space until the capture agent has read the commit record completing a given UR. Until then, data is accumulated in the staging space which might cause the staging space to overflow.

5. Format rows into CDC-internal format: As soon as an UR bucket is ready (commit was read), the log data is formatted into CDC-internal format to transfer it to the target. Again, the decoded data is converted to a database-independent intermediate representation.

6. Process compensated changes: Rollbacks write compensation records to the transaction logs. These special log records contain inverse operations to the specific action they revert. CDC is designed to save network traffic as well as target database load aiming at a low latency. Following this principle, compensated operations are removed from the staging space easing before they would be transmitted. This directly saves the network traffic for transmission along with the corresponding processing effort in the target database. These savings come at the expense of more complex staging space management. Every compensation log record needs a look up in the staging space main memory and on disk). The removal steps are done one by one for every compensation record causing a significant amount of staging space management overhead.

Summing it up, out of the 6 steps two (1 to 2) cannot be avoided. These steps are inherent to read the transaction log. In contrast, steps 3 to 6 are required due to CDCs platform independent design and can be removed or moved to the target system, which InSync does (cf. Figure 4). Support of multiple source and target database platforms imposes the need for intermediate representations along with the required conversions between source, intermediate and target formats. The effort for this design accumulates throughout the whole pipeline and becomes apparent in both:

1. generated CPU processing costs (MIPS: Millions Instructions per Second): System Z uses a consumption-based pricing model for executed instructions on a processor. Thus, customers are interested in keeping the amount of used MIPS as low as possible.

2. throughput: number of inserts, updates and deletes (IUDs) per second on the target. Throughput directly affects the LOAD-strategy of a customer. Higher IUD rates enables users to solely rely on replication technology. The lower the IUD rates the more likely it is that customers have to use periodic reloads because of consistently growing latency.

Above metrics are key differentiators for customers. Consequently, every reduction in costs and every increase in throughput will directly affect customer success.

4.2 Log Reader Enhancements

The key design points in log reader development were to tightly integrate the capture agent into Db2z and remove processing steps from the capture agent component. Tight integration of the log reader component simplifies installation and maintainability lowers the hurdle to start and operate IDAA. Moving processing steps from source to target demands for technical finesses presented in the following.

4.2.1 Move processing steps to target

Pushing required processing steps towards the Accelerator moves computation intensive tasks away from the source system towards the appliance. The cost model on the appliance does not include any computational costs and therefore directly lowers total cost of ownership for an IDAA system.

1. Decoding of the log records is no longer done on the source. In fact, encoded log record are given to the target database (Db2wh). There, the external table interface has been extended with the ability to handle native Db2z log records. No intermediate representation for data transmission is needed for this replication solution.

2. A staging space is no longer used. All read transaction log records – regardless of whether a transaction will be rolled back or committed in the future – is transmitted to target system. This helps to keep latency low as no time is wasted waiting for end of a transaction while the throughput is kept high.

3. With the removal of staging space, net-effect and compensation processing have moved to a different place in the processing pipeline. It is now done when data is prepared to be applied to the target database. At this point in time, it is already know if a transaction has been committed. Net-effect / compensation processing are done on whole batches of transactions rather than on single transactions which is much more efficient.

The combination of these modifications leads to a highly efficient appliance-centric processing model. It saves a significant amount of computation resources on the source side, which is the OLTP system.

5. DATA PROCESSING

The Log Reader on the source database system (Db2z) provides an RACF [12] or Passticket [13] protected API for interaction with InSync. The InSync component employs a single thread continuously interacting with the provided API to download chunks of variable size from the transaction log and store them in a cache. A single log parser thread retrieves data from the cache, reads through the log, classifies the log records according to their particular type (such as BEGIN, INSERT, DELETE, UPDATE, COMPENSATION, COMMIT or ROLLBACK) and triggers actions in the Transaction Manager defined by the type of the log record read. The Transaction Manager collects data manipulating log records classified by transaction in in-memory representations of transactions – the TransactionWorkItem. TransactionWorkItems can either be processed stand-alone or as a batch (5.3) together with other transactions. Batch processing exhibits optimization possibilities between the transactions. Single TransactionWorkItems or batches of TransactionWorkItems are finally applied to the target database by constructing delete and insert statements using Dynamic Apply Path Selection (7). In the following sections, the data processing depicted in figure 5 will be explained in more detail.

5.1 Log Parser

The Log Download thread continuously downloads chunks of the transaction log to the first in first out (FIFO) organized cache. Cache contents are processed single-threaded by the Log Parser. FIFO processing is essential, as it guarantees replay of the transmitted changes in log sequence order. Violations of log sequence order would lead to inconsistencies of the target database state. The Log Parser thread, reads through the cache contents on a per record basis. Records are structured as figure 7 illustrates (only showing data relevant for parsing). Each record[14] contains – besides the raw row information – a header that is required for Db2z processing. Within these headers, information about the log records' context are provided. This information is typical for write-ahead database systems employing Algorithms for Recovery and Isolation Exploiting Semantics (ARIES [12]) and makes our approach generally applicable for such databases.

Some of the information important for parsing the log are:

Type: Denotes the type of a log record. All of the replication relevant types for InSync trigger corresponding methods in Transaction Manager. Relevant types are:

- Unit of Recovery Control Records: Mark transaction start (BEGIN), rollback (ABORT) or commit (COMMIT). The Log Record Identifier of a BEGIN record is used as URID field for all records wrapped in the opened transaction.

- Undo / Redo log records: Redo log records denote modifications to database state (inserts / updates / deletes). These records contain information to uniquely identify the table they operate on. In Db2z this is implemented in the form of the database identifier (DBID) and the object identifier (OBID) along with the actual row data in database internal format. In case a transaction is rolled back, compensation log records are written to the transaction log. Compensation log records contain the inverse operation of the action in the log record they undo.

- Utility Log Records: Not only data modifications write log records to the transaction log, but also utilities for maintenance of similar. Depending on the utility log record, InSync is able to process the log record or it removes the affected table from replication.

- Diagnostic Log Records: On top of above required and well known log records, a new class of records is added:

[12]Resource Access Control Facility managing access to resources on z/OS operated mainframe systems (`https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zsecurity/zsecc_042.htm`).

[13]Temporarily valid one-time password, generated from an application dependent name, a secret token and current time information (`https://www.ibm.com/support/knowledgecenter/SSLTBW_2.4.0/com.ibm.zos.v2r4.icha700/pass.htm`).

[14]`https://www.ibm.com/support/knowledgecenter/en/SSEPEK_12.0.0/admin/src/tpc/db2z_logrecord.html`

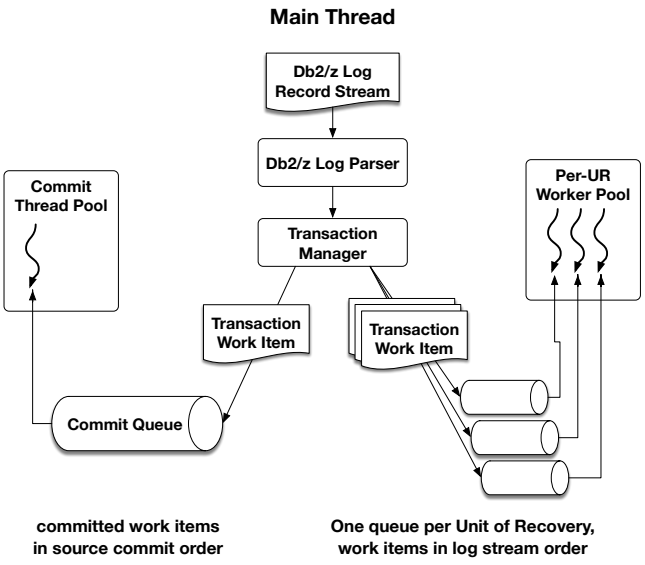

Figure 5: InSync Technical Overview

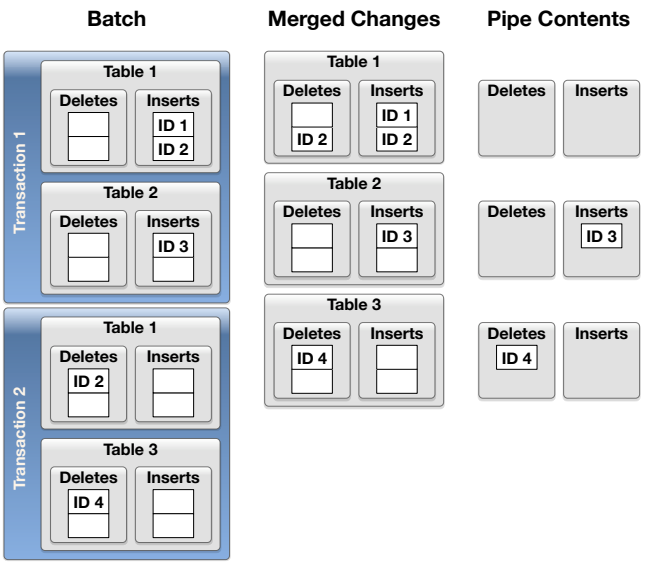

Figure 6: Batch Processing

Figure 7: Log Record Layout

Diagnostic log records. These records are written e. g., when the database schema is modified. This enables InSync to detect schema modifications using their respective log records. If a compatible schema change is encountered, it is processed accordingly. Unsupported schema changes cause affected tables to be removed from replication.

Log record identifier: Used to identify a particular log record within the transaction log. In the case of Db2z the used identifier varies depending on whether the system operates in data sharing mode or not. In non-data sharing mode the unique Relative Byte Address (RBA) whereas in data sharing environments the (potentially) non-unique Log Record Sequence Number (LRSN) is used.

Unit of Recovery ID: Undo / Redo Log Records carry a URID set to the log record identifier of the BEGIN record the undo / redo record belongs to.

The parsing process itself is designed to be as efficient as possible. To achieve optimal performance, only necessary information is read by using numeric offsets. Extraction and conversion of any row data is strictly avoided. Offset usage, allows for direct access to the relevant information without the need to touch any unnecessary log record data. The only data fields extracted from the (opaque) log records, are key columns required for further processing. The log record row data is not converted or processed any further by the parser. Instead, it is passed unmodified to the Transaction Manager for further processing.

5.2 Transaction Manager

The Transaction Manager collects all information about transactions stored in the transaction log. As we transmit all data – no matter if a transaction has been committed or will ever be committed, the Transaction Manager maintains an associative data structure which maps a URID to its corresponding in-memory transaction representation. This data structure helps the Transaction Manager to assign undo / redo log records to the wrapping transaction and keep track of the order of the transactions (ascending order with respect to the URID). The Log Parser calls a method according to the log record type read from the transaction log and triggers one of the following actions in the Transaction Manager:

- BEGIN: Creates an in-memory representation (TransactionWorkItem) associated with its URID that depicts a transaction in the transaction log. All subsequent insert, update, delete or compensation records associated with the above URID along with their respective LRSNs are collected in a transaction.
- INSERT / DELETE: Adds row image and LRSN to the transaction the records belong to. The DBID and OBID fields are used to store change records in the transactions separated by table.
- COMPENSATION: Adds the LRSN of the log record to be undone to the compensation list.
- UPDATE: Update records, in contrast to delete or insert records, contain two row images: a before image which contains the full row image before the update and an after image – a full row image of the row after the update. To add update information to the Transaction Manager, it is decomposed in a delete constructed from the before image and an insert constructed from the after image. The constructed images are added similar to native insert / delete records. This update decomposition allows for optimizations like net-effect processing 5.3.

- COMMIT: Signals successful completion of a transaction. When encountering this record, the corresponding transaction identified by its URID is removed from the open transaction heap and either added to the currently open batch (5.3) or send to a thread for processing right away.
- ABORT: Signals that the transaction has been rolled back. The URID is extracted and the transaction including all its occupied resources are released. The transaction is removed from the transaction heap.

5.3 Batch Processing

A batch is a set of committed transactions. Its size is determined by two configurable parameters: maximum number of transactions and batch timeout. Either the maximum size of a batch or the batch timeout is reached. Both of the criterions lead to closing a batch (called *sealing*). Sealed batches are added to the commit queue (FIFO) which is processed single-threaded.

The first processing step for a batch is to merge the collected changes across all transactions. This is done by iterating through the transactions (in commit order) and append the changes of every transaction to table objects identified by a combination of DBID and OBID. The result is a collection of table objects accumulating the changes of all transactions in a batch in commit order. Keeping the commit order is essential to guarantee a consistent target database state.

In the second step, unnecessary database operations are removed – a process called net-effect processing. But what are unnecessary database operations? To answer this question, recall that updates are decomposed in a delete and an insert row (see 5.2). When executing queries, for each table in a batch two queries will be issued: a delete query and a insert query. Delete queries are executed before the insert queries for a table. This processing scheme has two immediate consequences:

1. Deletion of a row appearing after the same rows insert in the same transaction will lead to incorrect target database state: The delete will be applied to the target database before the insert of the row.
2. Deletion of a row with an ID that has been inserted in a preceding transaction can be safely removed from the batch without affecting correctness.

To ensure correct target database state when situation 1 appears and to optimize the amount of database operations conducted on the target in situation 2 InSync employs the following techniques:

1. Intra transaction net-effect: A delete issued after an insert of the same row in the same transaction is not collected in the transaction in-memory representation. Instead, the collected insert is marked for removal during net-effect processing.
2. Inter transaction net-effect: When a delete on a row with an ID that has been inserted in a preceding transaction occurs, both records are marked for removal during net-effect processing.

Identification of rows net-effect can be applied to, is done by comparing the records using a unique key. Before being able to enable a table for replication a unique key (real unique key or informational unique key) has to be defined on the table. This key is the only data which is extracted from the row data images.

Assume InSync collects the transmitted transaction log in a single batch and the log sequence received was (transaction ID, table number, operation unique key): BEGIN 1 (1, 1, INSERT 1) (1, 1, DELETE 1) BEGIN 2 (2, 3, DELETE 4) (2, 1, DELETE 2) (1, 2, INSERT 3) COMMIT 1 COMMIT 2

The constructed in-memory representation of the transactions is depicted in figure 6. First BEGIN 1 causes creation of a in-memory representation of transaction 1. Log record (1, 1, INSERT 1) causes the row image for INSERT ID to be added to the inserted rows for table 1. The following log record triggers intra transaction net-effect processing: the row identified by ID 1 has been added before in the same transaction (TX). Consequently, the delete for that row is not added to the in-memory representation. Instead, the delete causes the preceding insert of ID 1 to be marked for removal during net-effect processing. BEGIN 2 creates a in-memory representation for transaction 2. The next log records add an insert of ID 3 to table 2 of TX 1 and deletes of ID 2 and ID 4 for tables 1 and 3 in TX 2. The commits for TX 1 and TX 2 cause the transactions to be completed and added to the currently open batch. After the batch has been sealed, the collected changes are merged. Table 1 now combines all changes of table 1 of TX 1 and TX 2. Merging of table 2 and 3 does not have any effect as these tables are contained in one of the transactions only. After merging, intra transaction net-effect processing is applied. In table 1 ID 2 has been inserted in TX 2 and subsequently deleted in TX 2. Hence, without applying both operations to the target database, the database state will still be correct. When writing the changes to the data structure (pipes / memory areas) as input for query processing, both the insertion and the deletion of ID 1 will be ignored. For tables 2 and 3 the collected changes will be written to the data structure as expected.

While intra transaction net-effect processing is required to keep database state correct (a delete will always be executed before an insert, even if it was after the insert in the log stream), inter transaction processing is a optimization only. Log sequences like continued updates to a row with the same ID exhibit the optimizations' full potential. In this case, we only need to conduct the first delete (if the row was present in the table before) and the last insert. All updates in between are unnecessary work and hence can be removed by net-effect processing.

Be aware that this processing scheme modifies the transaction pattern between source and target. On the source, each transaction collected in the batch is recorded as single transaction. After they have been collected in a batch, all of the transactions are consolidated in a single transaction on the target. This implies that either a whole batch is applied to the target or none of the changes of a batch is applied at all.

Besides batch processing, InSync employs another handling strategy for transactions exhibiting a particular behaviour. If a transaction has not been committed before a configured threshold of collected rows is reached, no compensation records and no deletes have been read, it can be processed in preapply mode.Instead of waiting for a commit for the transaction and adding it to the current batch, preap-

ply immediately starts to feed changes into an open transaction but does not commit the executed SQL statement in the target database. The changes remain uncommitted until the commit log record for the transaction has been processed. This is comparable to the concept of speculative execution – to speed up overall processing, we assume that the transaction will eventually commit since rollbacks are much less likely than commits in production transactional database systems.

The benefits of preapply is improved latency: if a large transaction is only processed when the commit is seen, a large number of changes have to be applied as part of processing the commit. However, the time can be used already for streaming the changes in smaller chunks into the target table. At commit time, only a small tail of remaining changes has to be processed. In addition, preapply helps to reduce memory pressure. Instead of channeling the transactions' changes through the whole batch processing and potentially keeping memory resources reserved by large transactions, preapply bypasses the full blown processing and makes sure to free resources as fast as possible and prioritize large transaction that might potentially increase latency.

However, this comes at a cost: If the preapplied transaction does not commit as expected, a delete or compensation record is read, there is some overhead involved for restoring a safe system state. First of all, the changes applied to the open connection need to be rolled back. Afterwards, the preapplied transaction is completed as any other transaction. Upon reading its' commit log record there is a special handling however. The currently open batch will be sealed and processed. This is what we call a batch breaking change which has impact on the performance of InSync.

The field of application for preapply also is very narrow. Transactions can only be preapplied as long as there haven't been any deletes and compensation records. As you may remember from section 5.3, deletes are always processed before inserts. As a consequence, if a delete is read in a transaction after a insert has already been applied, this order would be violated and can lead to results in the target database state. The same is true for compensation records – log records already applied to the connection cannot be easily removed when a compensation record is read.

6. MEMORY MANAGEMENT

Memory Management is an important aspect for InSync. All row images extracted from the transaction log along with the data structures required for their management, are channelled through InSync's limited amount of main memory. Long running transactions might accumulate a large amount of row images while millions of small transactions are processed in parallel. Figure 6 shows the data flow of the extracted row images.

When the Transaction Manager appends row images to the corresponding in-memory transaction representation, it keeps track of the data size of the row images stored in main memory. The collected row images keep accumulating until a configured threshold (typically 1 MB) would be exceeded by adding the current row image. Instead of adding the row image to the current 1 MB block, the block is written (spilled) to disk. It contains a header keeping track of the block id and some additional administration information enabling the system for easy removal of row images having been compensated or identified as unnecessary during net-effect processing. The rows register for removal during net-effect processing are associated with the block id, an offset marking the start of the record and the record length. This allows for easy skipping when row images are written (drained) to the data structures passed to the queries. A new block is allocated, prefilled with the header information and collects the subsequent row images added. As a consequence, a maximum of 1 MB is used for each – inserts and deletes – per table per transaction in memory. In memory critical situations, the block threshold can be adjusted to smaller values which will cause the row images to be spilled to disk earlier. This process is done before the merge (5.3) of the changes is conducted. Note that this technique is vital to support long running transactions. InSync keeps collecting the data for these transactions until they finally commit and are added to a batch after hours or even days. In the worst case, only 2 MB per table of this transaction will be kept in memory at any time.

At some point in time, the processing pipeline will be ready to prepare the row data for application to the target database. To write the row images into the source data structures to be applied, the spilled row images are processed by memory mapped I/O. A 1 MB block is mapped from disk into memory one by one to keep memory consumption as low as possible. While a block is mapped into memory, it is written (drained) into the source data structures for database application. Not all row images are written - instead, row images detected as unnecessary (compensation, net-effect) are skipped during the write process using the associated block id, offset and length of the record. This technique makes it possible, to remove rows identified as unnecessary and draining the surviving row images into the source data structures for target database application in just one pass.

7. DYNAMIC APPLY PATH SELECTION

Depending on the workload, InSync is confronted with different demands defined by workload characteristics. These characteristics vary along dimensions like number of changes per transactions, number of changes per table and number of tables per transaction. InSync chooses from two different paths to feed data to the target database - "external table" and "memory table". Both paths have their strengths and weaknesses defining their area of application.

An external table is a table stored in a database external file (or pipe) while still being accessible through SQL. The external table is expensive to use in terms of time consumption for preparation. A new (fenced) process is spawned for each external table operation, leading to a preparation time of approximately 200ms without having processed a single row. Once the external table is build however, it processes rows very fast due to heavy internal parallelization and pipelining. Consequently, a particular amount of rows has to be processed by an external table for the preparation time to be amortized.

On the other hand, the memory table is designed to avoid any form of preparation times, like plan prepare, thread or memory preallocation etc. It operates on fixed-size chunks that represent a number of rows in source-internal format, and does all of its processing streamlined on this chunk of data. In return, for large number of rows its processing is slower than the external table. Still, memory table is capable of processing a significant amount of rows during the preparation period external table processing requires.

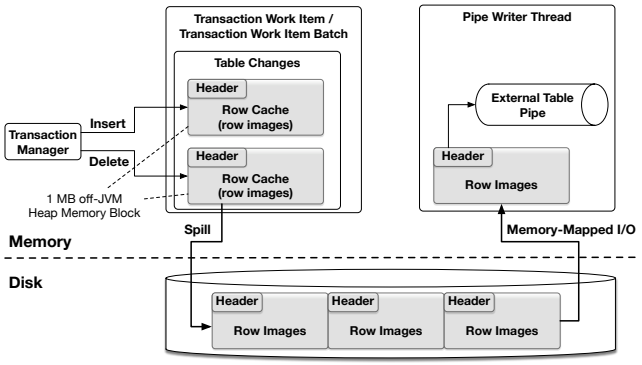

Figure 8: Memory Management

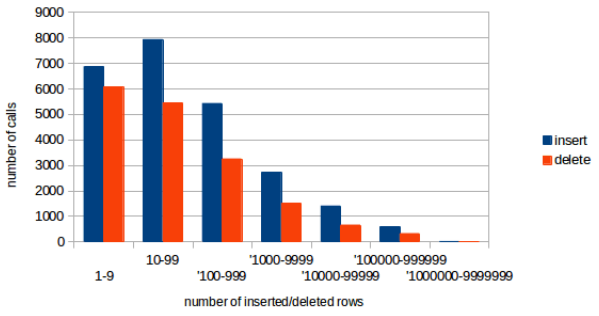

Figure 9: Workload Characteristics

To decide what query path to employ, InSync uses a pre-configured row threshold defining the amount of rows that can be processed before external table should be used. The decision is done for both, insert and delete operations on a per-table level. This way, the apply paths operate at their sweet spot guaranteeing high speed and low latency data replication.

Figure 9 represents an exemplary workload distribution. Most of the query execution calls, no matter if deletes or updates, contain a small amount of changed rows only with an emphasis between 1 and 9999 changed rows. This area is a perfect match for employing memory table. The less frequent cases from 100000 to 9999999 would be processed by external table.

8. AVOIDING DATA CONVERSION

To be able to support combinations of different source and target data sources, general-purpose replication products typically employ an internal, canonical data format to mediate between different source and target systems. Each replication source then transforms all captured rows into this intermediate format, sends it to the target which then transforms it again into a target-native format. It's a trade-off between interoperability and performance, favouring interoperability.

InSync is not a general purpose replication product, its focus is row to column-store synchronisation with a strict identity mapping (exactly the same schema on source and target) for tables. Interoperability between different source and target systems is a secondary priority. For this reason, we use the source-native data format as the general data encoding scheme for data exchange between replication source and target. This not only avoids any form of row-level data conversions in the capture and apply parts, but also comes with the benefit of making it extremely lightweight to capture changes on the source system. The source system is just forwarding transaction log records to the target in addition to flushing them to stable storage.

On the target system, the transaction log is parsed, but only up to the point where the actual before- and after row images are stored. The rows itself remain encoded in source-native format and the target replication system does not touch them. Instead, these are treated as opaque bytes, stored, staged and fed into the target system by means of the external- or memory table (cf. 7).

The target database system has been enhanced with conversion logic to its external- and memory table operations to be able to parse and interpret the source-native row format. This has proven to increase efficiency for small "trickle" style operations and for large, bulk-style changes alike. The transformation operators run directly in the insert codepath, and share the source-format parser. For low-latency trickle operations (memory table), the parser control block is cached between calls by means of a binary descriptor, to minimize per-row latency added by the row parser and conversion logic. Putting the data transformation logic into the target database also decreased the complexity of the replication system. It is unaware of the actual encoding used by source or target, and "just" moves bytes between both sides.

9. EVALUATION

The performance evaluation was conducted on an IBM zEnterprise z14 system with 6 general processors, 2 co-processors, and 96 GB of main memory, connected via 10 GB network to a 1-rack IBM Integrated Analytics System (IIAS) (M4002-010). Software-wise z/OS 02.02.00 together with Db2z v12 was applied. The used test workload consisted of 500 tables concurrently receiving inserts and updates with a peak transaction rate of 64,400 inserted rows per second and 64,400 updates per second and 30 concurrently running HTAP query threads.

9.1 Log Reader Results

For Capture Agent / Log Reader tests, the recovery log has been pre-populated to measure log reader work in isolation. Figure 10 shows the performance result, comparing executions times of different phases in the CDC and InSync log reader pipelines (cf. Figures 3 and 4). Processing shares on general processors and co-processors are shaded in different colors.

The CDC log processing phase, which reads log records, decrypts, decompresses, sorts, and filters them before converting them into the CDC-internal canonical format and applying change compensation, consumes the largest amount of processing time. The InSync log reader, which skips data conversion operations and change compensation, just requires roughly half of CDC's processing time. Because more data is transmitted via the network in the InSync-case, more time is spent in this phase compared to CDC, which is negligible compared to the overall processing costs.

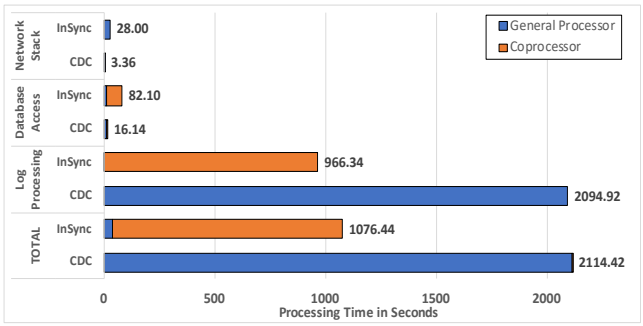

Figure 10: Log Reader Performance Comparison

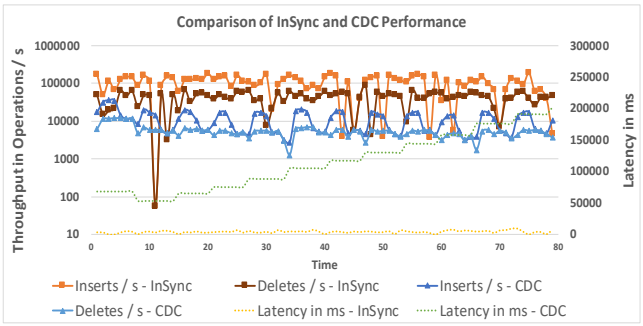

Figure 11: Performance Evaluation

The biggest advantage of InSync can be seen when comparing processing shares of general and co-processors. While only 0.13 % of CDC's workload is eligible for co-processor offloading, 99.87 % can be offloaded to inexpensive coprocessors for InSync. This results in both – a significant amount of saved time and reduced processing cost.

9.2 End-to-End Performance Results

In the end-to-end case, we focus on comparison of throughput and latency metrics.

For the workload described above, figure 11 shows CDCs latency continuously increasing until it reaches about 3 minutes. In fact, the figure only shows only an excerpt of the workload data for clearness. CDC latency builds up as high as 30 minutes. Throughput for inserts peaks at about 35,000 rows per second. Delete throughput reaches about 12,000 rows per second.

InSync latency peaks at about 10 seconds (cf. 11) which means that InSync is capable of keeping the latency 180 times lower than CDC despite the fact, that the latency does not constantly grow. Instead, the latency is stable and does not show any signs of falling behind. Throughput for inserts peaks at about 200,000 processed records per second while delete throughput reaches about 100,000 processed records per second.

The average query wait time has been determined to be 51 seconds in the CDC workload whereas InSync causes an average query wait time of about 6 seconds. Single HTAP query execution proved to exhibit stable wait times around 6 seconds for InSync.

Comparing above results, it turns out that InSync exhibits a 180× lower latency while increasing the insert throughput by about 6× and the delete throughput by about 8× for the test workload. This is a significant improvement over CDC replication in IDAA. For the sake of completeness, please note that the throughput numbers displayed are for only one thread. Both systems usually run with 8 parallel threads in a IDAA setup – the displayed number do only show a fraction of the actual system performance. The query wait time reduced by 8.5× and proved to be very stable with InSync.

10. LIMITATIONS

There are some limitations that exist with the introduction of Integrated Synchronization:

Data type support: IDAA does not support the data types ∗LOB, TIMESTAMP WITH TIME ZONE and XML due to their special internal storage concepts. This restriction existed in earlier IDAA versions.

Indexes are not enforced on the target database: As in earlier IDAA versions, we do not enforce referential integrity on the target database. It is enforced on the source database and therefore guaranteed on the target database as well. However, there are situations when referential integrity is and has to be violated on the target system (e.g. continuous replication: bulk load a table during replication)

Tables containing partitions with reordered as well as basic row format: As of this writing, Integrated Synchronization is not capable to replicate tables containing partitions in reordered as well as basic row format. This limitation will only be hit, when a transaction collects both row format types.

Replication delay: Very special workload characteristics might cause HTAP queries to unexpectedly time out. If the source system is flooded with exceptional high amounts of data irrelevant for replication you might observe this behavior as data collected in a batch might not get applied to the target database. This is a very unrealistic scenario in a customer environment but relevant for our testing.

11. SUMMARY

In this paper we have presented a new, fast, scalable, low cost replication technology to enhance IBM Db2 Analytics Accelerators' Incremental Update feature. Fast data replication from a high performance transactional database system to a column store database system needs to be guaranteed in high load situations while data integrity between the database systems needs to be maintained. The implemented solution turned out to be both, performant – latency dropped 180× while throughput increased 6× for inserts and 8× for deletes – and affordable – processing time decreased by about 50% for the test workload.

The next steps of our work will be to gradually add schema change replication. At the time of this writing, IDAA has only rudimentary schema change support (ADD COLUMN) when using CDC. InSync has no support yet. Starting with ADD COLUMN we will add support for schema changes to InSync step by step. Performance-wise we strive to further decrease latency while increasing throughput. This is a vital step for bringing OLTP and OLAP workloads together with IDAA's own implementation of Hybrid Transactional Analytical Processing HTAP. The lower the latency, the less query delay time HTAP queries will exhibit to guarantee transactionally-consistent query results.

12. REFERENCES

[1] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. *PVLDB*, 2(1):922–933, Aug. 2009.

[2] K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and E. Paulson. Efficient Processing of Data Warehousing Queries in a Split Execution Environment. In *Proceedings of the 2011 international conference on Management of data*, 2011.

[3] R. Barber, P. Bendel, M. Czech, O. Draese, F. Ho, N. Hrle, S. Idreos, M. Kim, O. Koeth, and J. Lee. Business Analytics in (a) Blink. *IEEE Data Engineering Bulletin*, 2012.

[4] S. Conn. OLTP and OLAP Data Integration: A Review of Feasible Implementation Methods and Architectures for Real Time Data Analysis. In *SoutheastCon, 2005. Proceedings. IEEE*, 2005.

[5] J. Dittrich and A. Jindal. Towards a One Size Fits All Database Architecture. In *Outrageous Ideas and Vision Track, 5th Biennial Conference on Innovative Data Systems Research, CIDR*, 2011.

[6] S. Elnaffar. A Methodology for Auto-Recognizing DBMS Workloads. In *Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research*, 2002.

[7] F. Färber, S. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner. Sap hana database: data management for modern business applications. *ACM Sigmod Record*, 2012.

[8] M. Grund, P. Cudre-Mauroux, J. Krüger, S. Madden, and H. Plattner. An overview of hyrise-a main memory hybrid storage engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 2012.

[9] M. Grund, J. Krüger, H. Plattner, A. Zeier, P. Cudre-Mauroux, and S. Madden. HYRISE: A Main Memory Hybrid Storage Engine. *PVLDB*, 4(2):105–116, Nov. 2010.

[10] J. Lee, S. Moon, K. H. Kim, D. H. Kim, S. K. Cha, and W.-S. Han. Parallel Replication across Formats in SAP HANA for Scaling out Mixed OLTP/OLAP Workloads. *PVLDB*, 10(12):1598–1609, Aug. 2017.

[11] D. Makreshanski, J. Giceva, C. Barthels, and G. Alonso. BatchDB: Efficient Isolated Execution of Hybrid OLTP+OLAP Workloads for Interactive Applications. *Proceedings of the 43th SIGMOD international conference on Management of data*, pages 37–50, May 2017.

[12] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz. ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logginig. *ACM Transactions on Database Systems*, pages 94–162, March 1992.

[13] H. Plattner. A Common Database Approach for OLTP and OLAP Using an In-Memory Column Database. In *Proceedings of the 35th SIGMOD international conference on Management of data*, 2009.

[14] V. Raman, G. Swart, L. Qiao, F. Reiss, V. Dialani, D. Kossmann, I. Narang, and R. Sidle. Constant-Time Query Processing. In *IEEE 24th International Conference on Data Engineering (ICDE)*, 2008.

[15] J. Schaffner, A. Bog, J. Krüger, and A. Zeier. A Hybrid Row-Column OLTP Database Architecture for Operational Reporting. *Business Intelligence for the Real-Time Enterprise*, 2009.

[16] K. Stolze, F. Beier, and J. Müller. Partial Reload of Incrementally Updated Tables in Analytic Database Accelerators. *Datenbanksysteme für Business, Technologie und Web (BTW) 2019 : Tagung vom 4. - 8. März 2019 in Rostock*, pages 453–463, March 2019.

[17] K. Stolze, F. Beier, K. Sattler, S. Sprenger, C. Grolimund, and M. Czech. Architecture of a Highly Scalable Data Warehouse Appliance Integrated to Mainframe Database Systems. *Database Systems for Business, Technology, and the Web (BTW)*, 2011.