Near Real-Time Analytics with IBM DB2 Analytics Accelerator

Daniel Martin
IBM DB2 Analytics Accelerator Development
IBM Deutschland Research & Development GmbH
Schoenaicherstrasse 220
71032 Boeblingen, Germany
danmartin@de.ibm.com

Oliver Koeth
IBM DB2 Analytics Accelerator Development
IBM Deutschland Research & Development GmbH
Schoenaicherstrasse 220
71032 Boeblingen, Germany
okoeth@de.ibm.com

Johannes Kern
IBM DB2 Analytics Accelerator Development
IBM Deutschland Research & Development GmbH
Schoenaicherstrasse 220
71032 Boeblingen, Germany
johker@de.ibm.com

Iliyana Ivanova
IBM DB2 Analytics Accelerator Development
IBM Deutschland Research & Development GmbH
Schoenaicherstrasse 220
71032 Boeblingen, Germany
iivanova@de.ibm.com

ABSTRACT

The IBM DB2 Analytics Accelerator (IDAA) implements the vision of a universal relational DBMS that processes OLTP and analytical-type queries in a single system, but on two fundamentally different query engines. Based on heuristics in DB2 for z/OS, the DB2 optimizer decides if a query should be executed by "mainline" DB2 or if it is beneficial to offload it to the attached IBM DB2 Analytics Accelerator that operates on copies of the DB2 tables. In this paper, we introduce the "incremental update" functionality of IDAA that keeps these copy tables in sync by employing replication technology that monitors the DB2 transaction log and asynchronously applies the changes to IDAA. This enables near real-time analytics over online data, effectively marrying traditionally separated OLTP and data warehouse environments. With IDAA, analytic queries can access data that is constantly refreshed in contrast to traditional warehouses that are updated on a daily or even weekly basis. Without any changes to the applications and without the need to introduce cross-system ETL flows, an existing operational data store can be used for data warehousing as well. The analytic query performance provided by IDAA makes it possible to execute reports directly against the transactional schema, thus avoiding the need for costly design and maintenance of a separate reporting schema. Additionally, the Accelerator shields DB2 for z/OS as the transactional system from performance degradation caused by the analytical workload and the replication component synchronizes all data changes in near real-time. We present the architecture of the integrated replication component of IDAA and discuss design decisions that we made when combining the different technologies as well as performance characteristics of the resulting system.

EDBT/ICDT '13 March 18 - 22 2013, Genoa, Italy

Categories and Subject Descriptors

H.2.4 [**Systems**]: Relational databases; C.2.4 [**Distributed Systems**]: Distributed databases

General Terms

Design

Keywords

db2, idaa, replication, analytics, reporting, OLTP

1. INTRODUCTION

IBM DB2 Analytics Accelerator (IDAA)[1] is an evolution of IBM Smart Analytics Optimizer (ISAO)[13] which was using the BLINK in-memory query engine[11, 3]. As a result of a re-design from ISAO, IDAA is now using the "IBM PureData System for Analytics"[2] as its foundation. Figure 1 gives an overview of the system: the Accelerator is an appliance add-on to DB2 for z/OS running on an IBM zEnterprise 196 or EC12 mainframe; it comes in form of a purpose-built software and hardware combination that is attached to the mainframe via redundant 10GB fiber channel connections to allow DB2 to dynamically offload scan-intensive queries.

[1] `http://www-01.ibm.com/software/data/db2/zos/analytics-accelerator/`
[2] `http://www-01.ibm.com/software/data/puredata/analytics/system/`

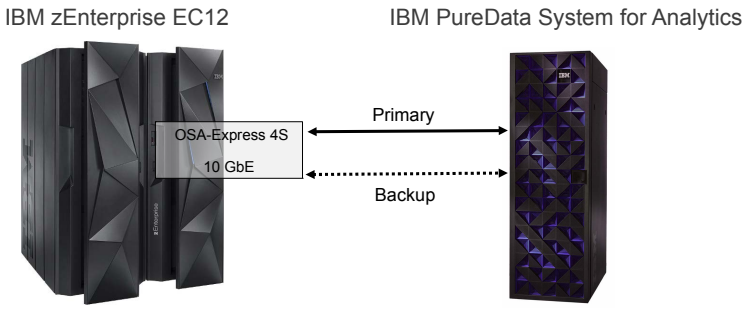

Figure 1: IDAA Hardware and Connectivity Overview

The Accelerator enhances the DB2 database with the capability to efficiently process all types of data warehousing workloads, standardized reports as well as ad-hoc queries and also extraction of pre-transformed data for data marts in specialized cubing applications. At the same time, the combined hybrid database retains the superior transactional query performance of DB2 for z/OS. Query access as well as administration use a common interface – from the outside, the hybrid system appears almost like a single database.

The acceleration factor compared to DB2 for z/OS standalone for such analytic queries is very significant, because IDAA processes table scans on all of its disks in parallel, leveraging FPGAs to apply decompression, projection and restriction operations before the data hits the main processors of the cluster nodes (called "SPU" in Figure 2). On the Accelerator, tables are hash-distributed across all nodes and disks based on a selected column (to facilitate co-located joins) or in a random fashion. A query is decomposed into so-called "snippets" that execute the scan portions on all 92 active disks (on a single-rack system) in parallel, resulting in a total scan rate of about 9GB/s (36GB/s with compression). The cost-based optimizer is aware of the throughput of the internal network fabric and decides for instance if a table is to be broadcast to all nodes or redistributed (based on the join column) between individual nodes before the actual join operation is executed. The engine itself is optimized for scans, besides Zone Maps (a form of skip-lists to avoid the overhead of scanning non-matching blocks on disk) there are no other structures (e.g. indices) that optimize the processing of predicates.

Table maintenance operations (e.g. RUNSTATS or REORG) are fully automated and scheduled autonomically in the background based on internal disorganization measures or if bulk table changes, such as a table re-load, have occurred. An IDAA installation inherits all System Z attributes known from DB2 for z/OS itself: the data is "owned" by DB2, i.e. security and access control, backup, data governance etc. are all managed by DB2 itself; the Accelerator does not change any of the existing procedures or violate any of the existing concepts. As a result, the combination DB2 for z/OS and the Accelerator is a hybrid system that consists of two architecturally very different databases systems: DB2 as a traditional RDBMS that has its strengths at processing highly concurrent OLTP workloads, using traditional serialization and indexing concepts, and IDAA as an accelerator engine available to DB2 to offload scan-intensive queries, which do not benefit from indices but essentially require a scan of all or a large fraction of the rows of all referenced tables.

Naturally, the design principles behind both systems are drastically different: DB2 for z/OS is based on a "shared everything" approach, i.e. all members of a so called "Data Sharing Group" have access to the same data. DB2 members use the System Z Sysplex facilities that offer e.g. distributed lock and buffer management to coordinate the work. The Accelerator, in contrast, is based on a shared nothing architecture, where each node is responsible for a fraction of the data and a centralized component (a so called "host" computer) orchestrates and distributes the work across all of the nodes. The main difference between these architectures is that, in the DB2 case, a single query can never occupy all resources of the system because it will get dispatched to a single data sharing group member and it will solely be processed by this member. DB2 has very efficient code paths to find and process the data from the storage subsystem so that a single member can process hundreds of queries at a time while still providing a very low response time. A single query in IDAA, in contrast, occupies all resources of the system because it is decomposed into snippets where the scan part is processed on all disks in parallel. Because it uses a distributed architecture where network interaction with all nodes of the cluster is required to process a single query, it is not possible to achieve very low response times. If there would not be any caches, a single query would take up all (scan-) resources of the system; running the same query 2-way parallel would result in twice the response time of that query.

Clearly, both systems not only complement each other perfectly, but can also shield each other from workloads that may have a negative impact like exhaustive use of resources if dispatched to the "wrong" system. For these query routing decisions, the DB2 optimizer implements a heuristic that decides if a given query should be processed by "mainline" DB2 or if it should be offloaded to the Accelerator. Because the Accelerator is a scan-optimized system, the heuristic bases its decisions on an estimate of the number of required fetches and how well existing indices match the predicates of that query.

It is very important to understand that nothing has to be changed on existing applications that are already using DB2 in order to get the benefits of the acceleration: they are still connecting only to DB2 and are using the DB2 SQL syntax. In fact, there is no indication to an application whether a query was processed by "mainline" DB2 or if it was offloaded

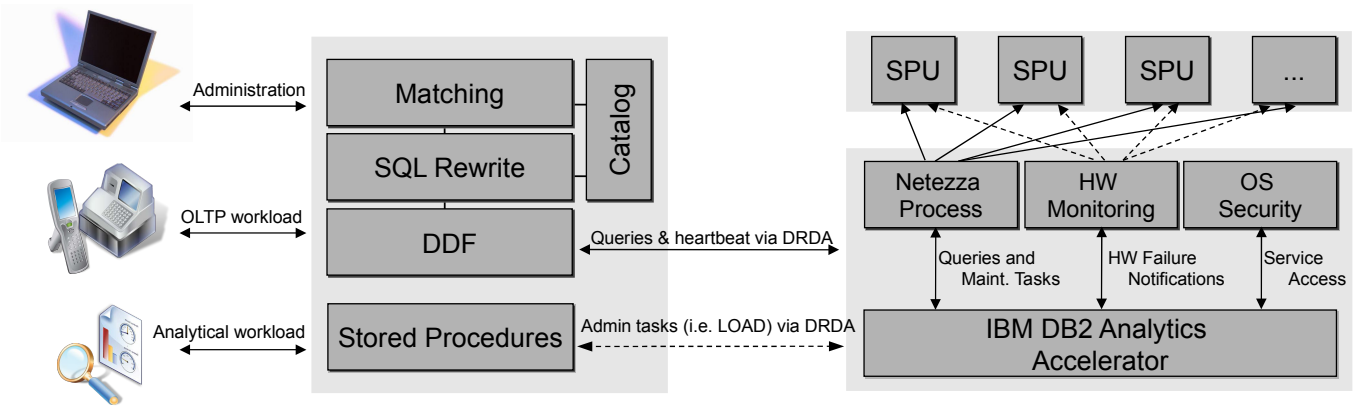

Figure 2: IDAA Architecture

to the Accelerator. Deep integration into existing DB2 components ensures that only little specific training is required to operate the Accelerator. DB2 remains the owner of the data; data maintenance, backup and monitoring procedures do not change.

Because IDAA operates on copies of the tables in DB2, any changes to these tables must also be reflected on the Accelerator. This is normally done by running an IDAA LOAD procedure that refreshes either an entire table or just the changed partitions of that table on the Accelerator. This refresh mechanism matches many use-cases of IDAA, as data warehouses are traditionally updated by ETL jobs that run in batch mode so that changes are applied in bulk on a scheduled basis. Users typically add another step to these batch procedures to invoke the IDAA LOAD procedures so that - alongside the data in DB2 - the data on the Accelerator is updated by either refreshing the entire table or by refreshing the changed partitions of a table.

Another use case, however, is to allow reporting over "online" data, providing a solution that combines the features of traditionally separated OLTP and reporting systems that are connected by aforementioned ETL jobs. For such scenarios, the granularity of the refresh mechanisms is too coarse: if the total size of the changes is very 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 causes a lot of unnecessary work (in the form of MIPS consumption). Also, the practical minimum latency that can be achieved by running the refresh procedures is in the range of hours; it is impractical to run the UNLOAD-based refresh procedures every hour or even more often. As a result, we are introducing a complementary table refresh method called "Incremental Update" that offers a mechanism to refresh the copy tables on the Accelerator by asynchronously monitoring the DB2 transaction log for changes. The changes are staged in memory and are transferred over the network to the Accelerator once their associated unit of work has been committed. For efficiency reasons, the received changes are applied in "micro batches" that typically contain data from all transactions that committed during the last 60 seconds. Multiple changes to a row within a micro batch are consolidated to a single change during the 60 seconds collection phase.

For providing this functionality, we tightly integrate an existing IBM replication product called IBM Infosphere Change Data Capture (CDC)[3] into IDAA. While this integration does not break new ground for the actual replication technology, as we are using an existing product, it offers a much simpler administration compared to the manual set-up and maintenance of a general-purpose database replication product. In addition, since we build on existing IDAA data management features, we are able to integrate row-based log replication technology – which performs best for small, dispersed changes in the data – with batch-load data transfer, which is much more efficient for large-scale changes in the data. In this paper, we introduce the architecture of the integration of the replication components, discuss the properties of the resulting system and provide measurements that highlight the properties of the different table refresh mechanisms.

2. RELATED WORK

Relational database system vendors have been focusing on analytical DBMS appliances since a while; popular examples are Oracle Exadata[4], the Teradata appliance[5] EXASOL's EXASolution Appliance[6] and SAP's HANA[7] [7, 10]. Similarly, appliances running MapReduce implementations together with an analytical DBMS are becoming 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

[3] http://www-01.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

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 the optimization of 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 (shared everything) 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 and to synchronize and possibly re-distribute work at runtime. By design, these systems have a minimum response time of several hundred milliseconds, as opposed to OLTP systems where the lowest response times typically are in the single digit milliseconds range.

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, DB2 for z/OS and the Accelerator are tightly integrated so that users experience a single coherent system in terms of management, backup, monitoring, SQL dialect and connectivity.

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. [12] 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. Existing analytics systems are based on a shared nothing architecture whereas traditional OLTP systems use a shared everything design. 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, the IDAA approach is to implement separate systems, using the OLTP system at the front (so that the performance characteristic of OLTP workloads does not change) 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).

[8] `http://www.greenplum.com/products/greenplum-dca`
[9] `http://www.asterdata.com/product/big-analytics-appliance.php`

3. DESIGN AND OPERATION OF INCREMENTAL UPDATE

The following section describes the architecture of the IDAA incremental update solution and discusses significant aspects that had to be considered when integrating the CDC product that implements the log reading and change replication.

3.1 Architecture and Basic Assumptions

The CDC product consists of multiple components that are installed on different systems. On the source system – the z/OS installation running DB2 – a "capture agent" is responsible for reading the transaction logs, extracting the relevant update information, and buffering it until the corresponding transaction has committed. When the changes are committed, they are handed over to an "apply agent", which consolidates the changes into micro-batches to improve efficiency and then applies them to the target tables using a bulk load interface via JDBC. Both capture and apply agent are configured and controlled by the "access server" process, which is not involved in the actual log shipping, but serves as the management end-point for replication configuration and monitoring.

For our design, a basic requirement was to re-use these product components unchanged rather than modify them or extract and re-use only the relevant part's functionality. The main benefits of this approach are the following:

- The code and data flow for the actual data replication is not modified at all, but the IDAA integration happens only on the configuration and management level. We completely re-use the established functionality of CDC and there is almost no chance of introducing functional errors into the actual replication process that could lead to corrupted data on the Accelerator.
- By re-using complete, well-tested components, we were able to ship an industry-strength solution for log-based replication in relatively short time (6 months from first design to first customer beta) and with comparatively moderate testing effort. For example, there is no need to re-test the complete matrix of various factors that influence the writing of transaction log records on DB2.

Figure 3 depicts how these parts integrate into the existing IDAA architecture.

The placement of the capture agent on the z/OS side is dictated by the overall solution design. The apply agent and access server components can technically be installed on any system that has network connectivity to the target database. We found that it was possible to install the components on the Accelerator "host" server (cf. Section 1), as performance

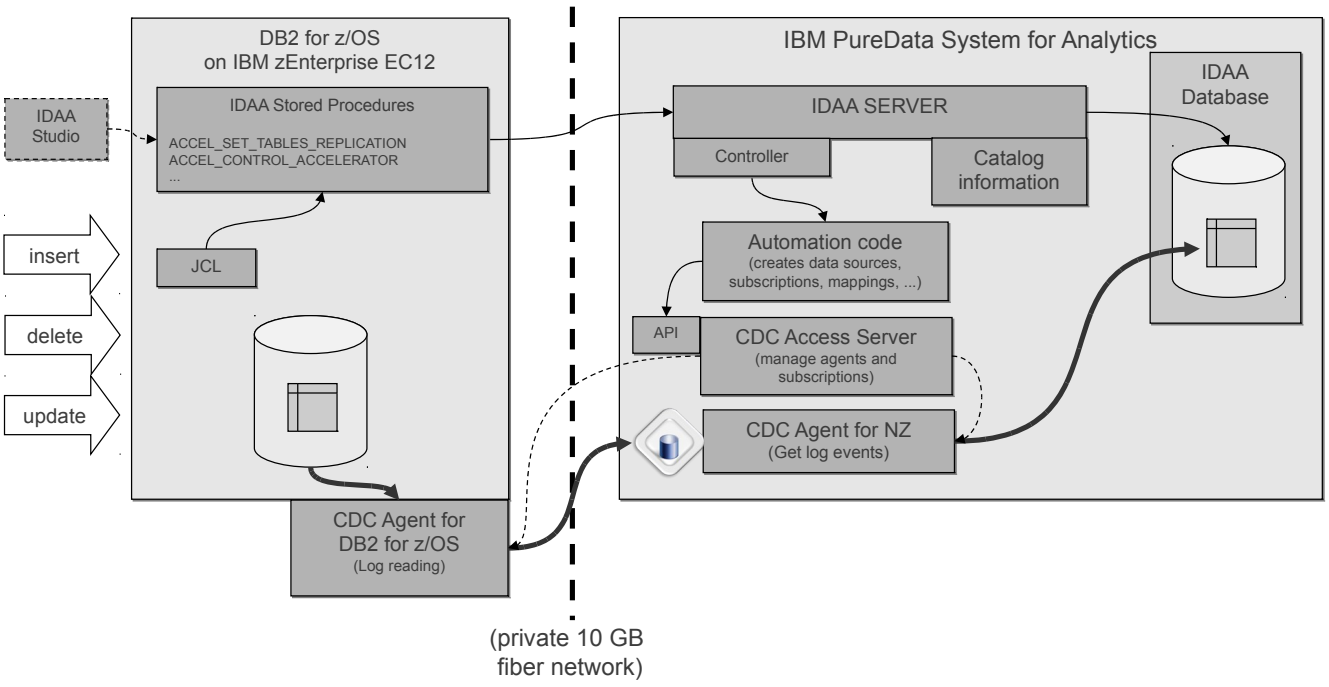

Figure 3: Architecture of the "Incremental Update" Feature

testing proved that the host system had sufficient resources to handle these additional processes under normal workload conditions. By keeping the components tightly integrated on the same server that also runs the IDAA server software connecting to DB2, we can minimize network communication and shield the installation and management of these components. So the administrator of the IDAA system – who works on z/OS with skills for that platform – only needs to install the CDC capture agent for DB2 and does not need to explicitly manage other systems.

3.2 Management Interface

In a stand-alone CDC installation, the setup is explicitly managed from a Windows GUI connecting to the CDC access server. This GUI allows the administrator, for example, to define which tables should be replicated (including column mappings), to start and stop replication and to monitor the process for errors. In the IDAA context, administration of the replication function should be integrated with the general administration of the Accelerator and the number of necessary tasks should be reduced to a minimum. Therefore the CDC administrative GUI is replaced by automation code that directly calls the same underlying APIs. Some of the CDC administrative operations can be fully automated; for example it is not necessary to define column mappings any more, since the schema of the tables on the Accelerator is fully controlled by IDAA code so the mapping can be automatically configured in CDC. Two important new explicit actions that were added to the existing IDAA studio admin GUI for incremental update support are defining which tables should be replicated and starting and stopping the replication process. A screenshot can be seen in Figure 4.

All administrative actions in the IDAA GUI are executed by calling IDAA administrative stored procedures, which are deployed in DB2 and communicate internally with the Accelerator server. It is also possible to call these stored procedures directly from batch environments using the z/OS job control language (JCL). This allows to integrate the control of IDAA incremental updates into batch processing, for example with ETL flows that create new tables or make schema modifications which cannot be handled automatically.

In addition to these explicit new actions, several of the existing administrative operations in IDAA had to be extended to allow seamless interaction with the new CDC processes, because the unchanged CDC apply agent is not aware of the IDAA server code and operates directly on the underlying database. Most importantly, any schema manipulations (like modifying the key for table distribution across SPUs) must be blocked for copy tables that are currently being modified by the apply agent – for replicated tables, these actions only become available when replication is stopped. Note that stopping replication does not "lose" any updates; once replication is restarted, it will pick up at the DB2 log position where it left off so any changes that happened in the meantime are then propagated.

3.3 Integration with Bulk Load Mechanisms

Another important aspect is the integration with the existing IDAA LOAD mechanism. CDC would itself support transferring the contents of an entire table, e.g. after it has been first defined. However, for cases where large amounts of data have to be transferred, it still makes sense to use the existing IDAA LOAD mechanisms: These have been highly tuned to maximize performance, e.g. by using the DB2 UNLOAD utility that bypasses normal SQL handling and by handling multiple partitions in parallel, so that data trans-

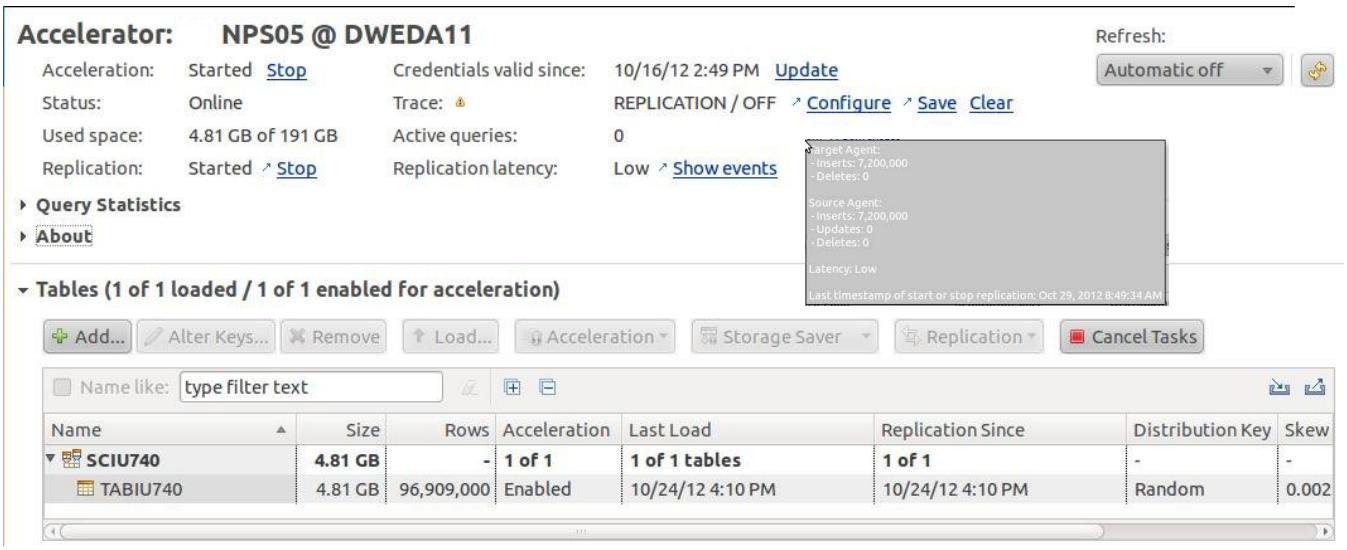

Figure 4: Screenshot of the IDAA Studio User Interface

fer rates greater than 1TB/hr can be achieved. In addition, IDAA LOAD minimizes CPU consumption on the System Z side because rows are extracted in DB2 internal format and directly sent over the network, so that no transformations occur on System Z; all decoding work is handled on the Accelerator.

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 DB2 to the Accelerator. When LOAD completes, it sets a "capture point" for CDC so that only subsequent updates are propagated (changes during LOAD are prevented by a table lock in DB2).

2. After that, CDC-based incremental updates keep the offloaded table data in sync with DB2 as it is modified by inserts, updates and deletes.

3. When large scale changes for the table occur or the table is modified by DB2 utilities that bypass the log (so CDC cannot capture the changes), the administrator can temporarily suspend replication and reload the table or the affected partitions with IDAA LOAD which will then set a new capture point for CDC. Replication must remain stopped while the table is loaded to avoid the risk of applying an update twice (once as part of the IDAA load and again via CDC).

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.

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. As long as the IDAA LOAD is used for full table reloads only, the integration guarantees that no changes can be "lost" or accidentally applied twice. However, if the administrator chooses to perform only a partition-level update in step 3, it is her responsibility to ensure that no updates on other partitions have been missed. All refresh options allow concurrent query access to the table, i.e. the old data of the table is still available for query access while the refresh is in progress.

To actually enable this level of integration, an enhancement of the base CDC product was necessary: The IDAA LOAD operates at the granularity of DB2 table partitions because this is the smallest unit of data which can be efficiently unloaded from DB2 (technically, a table partition corresponds to a single data set/file in z/OS). In order to map the partition granularity to the copy database in IDAA, which does not provide a corresponding explicit management unit, the table copies created on the Accelerator are augmented with a hidden column that IDAA uses to record the DB2 partition ID for each row, thus allowing reasonably efficient deletion of one or more partitions that have to be reloaded. In order to keep this partition replacement logic functional even in the presence of incremental updates, CDC needs to maintain the hidden partition ID column as updates are propagated from DB2. For example, updates affecting the partition key column in a range-partitioned table might move the updated rows into a different partition. The partitioning information for each modified row can be deduced from DB2 transaction log information, since it is required for "normal" rollback or log replay in DB2, but CDC had to be extended to allow mapping this information into an explicit column.

3.4 Other Integration Aspects

Monitoring of operations is an aspect where a seamless integration of the underlying CDC product presents particular difficulties. On the user-interface level, this integration has been achieved by providing access to the CDC event log via IDAA administration mechanisms. CDC events are presented in the IDAA Studio GUI, which can serve as a single point of access for monitoring. In addition, some monitoring information is also available in the job log of the capture agent running on z/OS which integrates well with

established monitoring procedures of customers. However, the actual event messages are provided by CDC and are of course at a lower level of abstraction than the administration, which is wrapped by IDAA. For example, the events may refer to CDC administrative concepts that are not apparent to the user (because they are implicitly managed by IDAA) or they may show internal errors that are irrelevant because they have been recovered at the IDAA administration level (like stopping replication again "to be safe" even if it appears to be already stopped). Further integration work remains to be done in this area.

For the software management side, on the other hand, integration into IDAA works seamlessly and automates almost all of the work that an administrator would have to perform in a stand-alone installation of the CDC product: High availability of the access server and apply agent is guaranteed by the integration into the high availability mechanisms of the Accelerator – failing processes are automatically restarted and if the entire Accelerator host system should become unavailable, the Accelerator will automatically fail over to a second stand-by host. IDAA supplies the necessary mechanisms to restart all affected software, including CDC access server and apply agent. The existing error handling of CDC will then cause the apply agent to re-connect to the capture agent on z/OS and request retransmission of all changes that have not been committed in the target database.

Management of software updates is also integrated with IDAA and is handled the same way as all other software updates for the Accelerator server. The update packages are deployed in the z/OS file system and the IDAA Studio GUI is then used to transfer the update packages to the Accelerator and activate the new software versions. To do so, IDAA wraps the installation images for CDC and provides the necessary scripting to allow a fully-automated "headless" invocation of the CDC install and update functionality.

For the query processing on IDAA, the introduction of incremental update means literally no change. The decision to offload queries is based only on the availability of data on the Accelerator and not on the currency of the data, so the heuristics are not affected. As transactions are continuously propagated from DB2 to the Accelerator, they are isolated from concurrently running queries by the normal transaction mechanisms on IDAA. 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 now return different results – the same way as if the report query had been executed twice on DB2. In other words, the Accelerator cannot serve as a mechanism to isolate the reporting facilities from changing data, as some customers like to use it. Instead, if snapshot semantics are desired in combination with incremental updates, they can only be achieved by modifying the queries accordingly, for example by not running them against a changing fact table but against a view that restricts on a time dimension. By updating the view definition, the range of data that becomes visible to a particular class of reports can be explicitly controlled. This way, the Accelerator again becomes fully transparent and the decision whether a query is offloaded does not carry any semantic consequences and can be done purely on performance considerations.

4. ADMINISTRATION DECISIONS AFFECTING INCREMENTAL UPDATE

As shown, the introduction of incremental updates in IDAA requires very few additional administrative steps. Still, there are some particular aspects that an administrator should take into consideration in order to obtain the desired query semantics as well as optimal performance.

4.1 Configuring Table Keys for Replication

One important aspect of incremental updates in IDAA is the need for tuning to achieve good update/delete performance. In general, for any log-based database replication solution, transferring inserts is straightforward, while for updates and deletes it must rely on unique table keys combined with an efficient lookup mechanism to locate the affected rows in the target table. For traditional transactional RDBMSs, this means that the rows in the target table should be indexed by a unique key from the source table. The Accelerator database is optimized for scan-based analytical query processing and therefore does not use or support indices. Reasonably efficient row update/delete performance can still be achieved by the use of Organizing Keys and Zone Maps (cf. Section 1): Organizing keys define columns that are used to cluster rows with similar column values together in the physical table organization on disk. Zone maps record the min/max column values for table extents on disk, allowing a table scan to quickly skip over all extents that are known to contain no matching rows for a given scan predicate.

In the context of IDAA incremental updates, we try to exploit these concepts by configuring CDC so that it uses a unique key on the source DB2 table (and a corresponding predicate for a delete or update on the target table copy) that matches the clustering of the target table and therefore allows for reasonably efficient update and delete performance. The problem is also somewhat alleviated by the fact that updates are not propagated row-wise but in micro-batches, so that a single table scan is sufficient to handle all the updates and deletes of a single batch. Also the largest tables in a warehousing or ODS context are usually fact tables which are almost exclusively modified by inserts, while updates and deletes are relatively rare. Still, cases of large dimension tables with high update frequencies, like accounts or marketing campaign members, do occur, and those require an appropriate choice of organizing keys to achieve good performance with incremental updates.

The selection of a unique key for propagating row updates is performed automatically and requires no manual tuning, but, of course, this approach requires that organizing keys on the target table have been defined at all and that a unique key on the source table exists which matches some subset of the organizing keys, so explicit setup by the administrator is still required. Organizing keys are also used to optimize query performance – by selecting organizing key columns that occur frequently as selection predicate in queries, the corresponding query scans can be accelerated – so there exists a conflict of objectives concerning the choice of organizing keys. Fortunately, IDAA allows up to four organizing key columns which are *independent*, because the table clustering follows a Hilbert space filling curve with regard to *all* organizing keys (instead of a lexicographical ordering which would first order by the leftmost key column and only then by the second etc.). In practice, most warehousing and ODS schemas use single-column surrogate primary

keys, so for these cases, it is possible to use these as one of the organizing keys and reserve the other three for query optimization.

5. MANAGING DATA COHERENCE IN IDAA

As discussed before, IDAA now offers two complementary and integrated mechanisms to keep the data on DB2 in sync with the data on the Accelerator – IDAA LOAD and incremental update. It is important to understand the properties of each of these mechanisms to be able to architect a solution that does not expose data latency issues to the applications that offload queries to IDAA. In most use-cases, it is not important that the data is synchronized at all times; it is important however that the "application perceived latency" is zero. If, for instance, an application does risk reports over a period that is at least a week in the past, it is sufficient to refresh the tables in the Accelerator every week. If such a report is run constantly every hour to analyze all transactions that occurred during the last 24 hours, there usually is no impact on the quality of the report if it misses the data that occurred during the last couple of seconds. IDAA in this case can be refreshed constantly using incremental update that operates at an average latency of about 60 seconds.

If however, a report requires "read your own writes" semantics – i.e. it must read data that was written right before the report was submitted – IDAA provides special support with its administrative stored procedures: A stored procedure call has been added that records the current transaction log position in DB2 and then blocks until all transactions that were committed at that log position have been replicated on the Accelerator. The procedure fails with an error if the transactions have not been applied on the Accelerator within a given timeout. In a SQL processing flow that contains both update transactions and reports (queries), such a stored procedure call effectively acts as a barrier that guarantees that the effects of preceding updates will be visible to subsequent queries. Only when the report must run in the same Unit of Work as previous update, query offloading to the Accelerator must be prevented because all data update mechanisms only transfer committed DB2 transactions.

The most significant differences of the data update mechanisms are the following:

- Incremental update works on the granularity of a single row, regardless of the partitioning scheme of a table. Cost and performance are relative to the number of changed rows.
- IDAA LOAD works only at the granularity of partitions or unpartitioned tables. It offers significantly better throughput and lower processing cost per transferred row, but cost and throughput are relative to the size of the modified tables or partitions (regardless of the number of affected rows).
- Incremental update continuously updates tables on the Accelerator with changes captured from the DB2 for z/OS log. Change propagation can be temporarily suspended by the administrator.
- IDAA LOAD must be triggered explicitly. DB2 real-time statistics are used to inform the user about tables or partitions that have been changed since the table was last loaded into the Accelerator. It is possible to set up automated procedures that periodically trigger a reload of all modified tables or partitions.
- Incremental update has a typical latency of about one minute and informs the user what the current latency of the replication process is, i.e. how far behind is the table on the Accelerator in comparison to the table on DB2.
- The minimally feasible latency for periodic refresh with IDAA LOAD is in the range of hours, depending on the size of the modified tables or partitions. The last refresh time of each table is visible in DB2 catalog tables.

The decision when to use which table refresh method can be taken on the level of individual tables. It is heavily influenced by the application(s) that drives the changes that need to be reflected in the Accelerator and the application(s) that offload queries to IDAA. Note that there are many factors that influence that decision, e.g. the total volume of changes, the distribution of updates across partitions, latency requirements, etc. Figure 5 provides a generic decision tree that suggests a table refresh method based on the aforementioned factors.

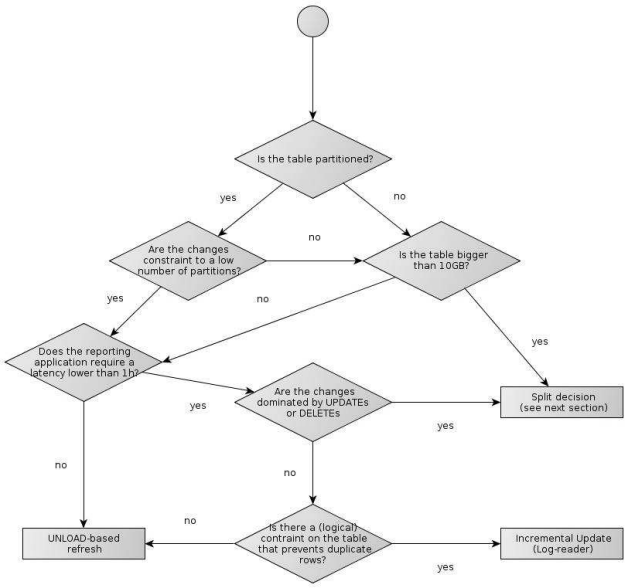

Figure 5: Decision Tree that Proposes a Table Refresh Method Based on Application Requirements and Table Characteristics

Sometimes you end at a split decision where a subset of the available refresh options would be applicable and it is not clear which one would be the better option for your use-case. In order to be able to make an informed decision we measured the CPU costs (very important because of System Z's utilization-based pricing model) and throughput of incremental update IDAA LOAD and present the results in the following section.

6. EVALUATION

Table 1 presents the results of a series of tests we conducted with an IBM zEnterprise 196 mainfra DB2 for z/OS on an LPAR that has 4 dedica signed and that is connected to IDAA using C 4S 10 GB Ethernet adapters. The IDAA har used is a full-rack system that internally uses 9(12 IBM HS-22 blades with 2 Intel Westmere 2. CPUs each and 12 FPGA daughter-boards o blades with 4 2-core Xilinx FPGAs each. Com in this FPGA-x86 cluster are coordinated usi 3650 M3 hosts with 2 Intel Nehalem 2.4 Ghz In total, the system has 112 Intel processor cor cores and 96 TB of raw disk capacity.

For the incremental update throughput n we created 8 tables on two different DB2 and ran a workload generator application th ning 8 connections in parallel that were doi in the form `INSERT INTO TABLE tab1 SELECT source_table WHERE (...)` with the WHERE- ifying 10.000 rows averaging 159 bytes of l IDAA LOAD we used a partitioned TPC-DS[1] (STORE_SALES) with 8 UNLOAD utilities wo table in parallel. Both of these scenarios produc put that is in the upper range of what we measu tests for the respective refresh methods.

Table 1: Comparison of the IDAA Refresh Options

	Incremental Update	IDAA LOAD
Throughput	<= 18 GB/h	<= 1200 GB/h
Latency	~ 60 seconds	> 1 hour
CPU usage per 100 MB	~ 130 CPU seconds	~ 0.4 CPU seconds

To measure the CPU usage of each of the refresh mechanisms, we generated 100 MB of changes on a table and loaded these changes to IDAA using both methods. By design, IDAA LOAD uses a lot less CPU in comparison to incremental update because it directly reads the table data in internal DB2 format. Incremental update in contrast scrapes the DB2 log and filters and extracts the changes from the log records – a far more expensive operation.

6.1 Impact of Incremental Update on Concurrently Running Queries

In order to measure the impact of incremental update on concurrently running queries, we conducted a series of tests with different query characteristics and incremental update workload profiles. We ran 10 queries in parallel, 5x a query that is sending large result-sets back to DB2 (a so called "streaming query") and 5x a query with aggregates and `GROUP BY`s over a high number of rows. In parallel to this query workload, we either did not run incremental update at all ("No-CDC-LOAD"), a medium incremental update workload ("Medium-CDC-LOAD"), or a heavy incremental update workload ("Full-CDC-LOAD"). The "medium" workload was about 300.000 INSERTs and 200 UPDATEs per minute from one DB2 subsystem into the Accelerator. The "full" workload was about 1.000.000 INSERTs and 600 UPDATEs per minute from two DB2 subsystems in parallel,

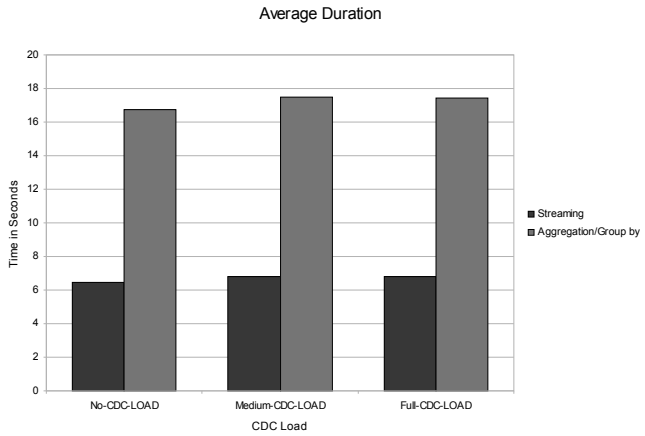

Figure 6: Impact of "Incremental Update" on Concurrently Running Queries

The response times for the two query characteristics under these incremental update workloads are presented in Figure 6: we see a very low impact on the response times for both kinds of queries, even with heavy incremental update traffic being applied concurrently to query execution on the Accelerator, the response times of the streaming and aggregation-type queries changed only marginally in comparison to running them alone on IDAA without replicating changes at the same time.

7. CONCLUSION AND FUTURE WORK

In this article, we have shown how incremental update capabilities were added to the IBM DB2 Analytics Accelerator that integrate seamlessly with the existing product and open up significant new usage scenarios. IDAA already excels at managing combined transactional and analytic workloads: data for both purposes can be stored in the same database management system and, depending on its particular characteristics, each query is executed on an execution engine that is either optimized for highly concurrent read-/write workloads with index access (mainline DB2) or high-performance execution of complex, scan-intensive queries (Accelerator). Keeping both operational and analytic data in the same system can greatly simplify Extract-Transform-Load (ETL) processing – in the extreme case, analytical reports can even run on the very same tables as daily operations, because the offloading of analytical queries to IDAA protects the transactional workload from negative performance impacts by business analytics processing.

In such scenarios, incremental update can show its full strength. On a technical level the separation of transactional and analytic processing and data (in DB2 vs. on the Accelerator) is still maintained, so the transactional workload stays protected. From an application and administrative viewpoint, the separation becomes almost invisible, as the latency for propagating updates is reduced from

[10]http://www.tpc.org/tpcds/default.asp

hours or days to minutes. Of course, having a well-designed, physically separated data warehouse with star or snowflake schemas has distinctive merits, both on a conceptual and integrity level and also with regard to performance of analytical operations, and we are not disputing that. Still, the high up-front cost for doing a full warehouse design and ETL implementation is often prohibitive from a business and/or cost perspective. In these cases, running analytic reports directly on the operational schema becomes a possibility with IDAA and incremental updates. There are some modeling deficiencies that cannot be solved easily in an operational schema (e.g. modeling changes without losing history data is usually not possible) and there is a higher cost for running analytical queries in a non-optimized schema, but this is still a better option than renouncing analytic queries altogether.

For "operational analytics" scenarios, where analytic queries are integrated with specific business flows, IDAA with incremental updates is a perfect match. A prototypical example is the customer representative that can see the latest up-to-date aggregate information for her clients and region as she is handling her calls – in such situations, even a one-day time lag in reporting can be hard to tolerate. For example, we have encountered a customer situation where legal requirements dictated that a "no advertising" policy request from their clients was to be "immediately" propagated through to the reporting facilities.

As we have laid out in previous sections, there are still shortcomings in the current solution that we want to overcome in the future. Currently, we need to establish a read lock on tables during an initial or update IDAA LOAD if the tables should afterwards be replicated to avoid the possibility of "lost updates" or "double inserts". This can be solved by integrating deeper with the replication technology, which would allow us to merge the changes that happened during unload into the offloaded table after the unload phase completes without risking "double inserts" for those changes that were already picked up in the initial transfer.

Another potential future enhancement is consideration of data latency in offloading decisions. As stated previously, the current query processing logic does not consider the currency of offloaded data in the offloading decision, but it is a conceivable improvement to allow routing policies like "offload this query only if all affected tables have been synchronized within the last 5 minutes, otherwise execute on DB2 data". To facilitate such decisions, information about the current incremental update latency must be passed back to the DB2 optimizer – latency is usually stable around 1 minute, but it may temporarily increase due to heavy change activity in DB2 or replication may be temporarily suspended due to administrative actions.

As the IDAA incremental update feature is now moving from beta into full availability, we are looking forward to learn more from our customers about new application scenarios that they see and enhancements that can be added to support them.

8. NOTICES

9. 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. *Proceedings of the 35th International Conference on Very Large Data Bases (VLDB)*, 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. *Proceedings of the 36th International Conference on Very Large Data Bases (VLDB)*, 2010.

[10] 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.

[11] 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.

[12] 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.

[13] 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.