In-Memory BLU Acceleration in IBM's DB2 and dashDB: Optimized for Modern Workloads and Hardware Architectures

Ronald Barber, Guy Lohman, Vijayshankar Raman, Richard Sidle
IBM Research – Almaden
San Jose, CA, USA 95120-6099
{rjbarber, lohmang, ravijay, rsidle}@us.ibm.com

Sam Lightstone, Berni Schiefer
IBM DB2 Development, IBM Canada, Ltd.
Markham, ON, Canada L6G 1C7
{light,schiefer}@ca.ibm.com

Abstract – **Although the DRAM for main memories of systems continues to grow exponentially according to Moore's Law and to become less expensive, we argue that memory hierarchies will always exist for many reasons, both economic and practical, and in particular due to concurrent users competing for working memory to perform joins and grouping. We present the in-memory BLU Acceleration used in IBM's DB2 for Linux, UNIX, and Windows, and now also the dashDB cloud offering, which was designed and implemented from the ground up to exploit main memory but is not limited to what fits in memory and does not require manual management of what to retain in memory, as its competitors do. In fact, BLU Acceleration views memory as too slow, and is carefully engineered to work in higher levels of the system cache by keeping the data encoded and packed densely into bit-aligned vectors that can exploit SIMD instructions in processing queries. To achieve scalable multi-core parallelism, BLU assigns to each thread independent data structures, or partitions thereof, designed to have low synchronization costs, and doles out batches of values to threads. On customer workloads, BLU has improved performance on complex analytics queries by 10 to 50 times, compared to the legacy row-organized run-time, while also significantly simplifying database administration, shortening time to value, and improving data compression. UPDATE and DELETE performance was improved by up to 112 times with the new Cancun release of DB2 with BLU Acceleration, which also added Shadow Tables for high performance on mixed OLTP and BI analytics workloads, and extended DB2's High Availability Disaster Recovery (HADR) and SQL compatibility features to BLU's column-organized tables.**

***Keywords** – in-memory; Business Intelligence; analytics; multi-core; compression; query processing; SIMD; cache-conscious; BLU; dashDB; DB2*

I. Introduction & Motivation

Most major database management system (DBMS) products first appeared several decades ago, and so were engineered primarily for the hardware of those times. In the interim, hardware has undergone remarkable changes. CPUs improved first by increasing their clock rates, but hit a heat barrier, forcing the current growth in the number of cores, rather than clock rate. Disks have increased storage density, and, to a lesser extent, transfer bandwidth, but disk arm access times have not improved appreciably in decades. Meanwhile, the speed, capacity, and cost of the DRAM used in main memories of CPUs have continued to improve exponentially according to Moore's Law. This has increased the relative speeds for accessing main memory (DRAM) versus disk by orders of magnitude, and concomitantly made terabyte-sized main memories on CPUs commercially viable.

Consequently, DBMS vendors are increasingly retrofitting their systems to exploit these enormous main memories and the parallelism that multi-core architectures make possible, and even necessitate, to continue to improve performance [1, 2, 3, 4].

However, many academicians, and even some products, naively assume that all of these abundant resources will be available to the processing of a single, complex query against a single database in a Business Intelligence (BI) or analytics workload. They therefore conclude, incorrectly, that the entire database fits in memory and that there will be unlimited memory to perform all query processing, so one no longer needs to bother with mundane considerations of finite memory, memory hierarchies (especially disk), spilling, buffer pools, paging, or even memory-efficient algorithms. "It all fits!", they exult. While this may be true for limited proof-of-concept systems, the economic realities of most modern enterprises dictate that the production systems hosting BI or analytics workloads are inevitably shared among a number of databases, each hosting a large number of applications, each of which can have hundreds to thousands of concurrent users competing for those supposedly "plentiful" resources. Furthermore, storing the base tables in a database is only the beginning of the demands upon the main memory. Indexes; temporary tables and materialized views; the overheads of meta-data tables (e.g., system catalogs, statistics), logs, monitoring structures, and working memory for processing queries, such as hash tables for joins and grouping, all compete for memory.

To grasp the impact of these "hidden" demands on memory, let's consider an example of a single "toy" database whose base tables consume a mere 200 GB, having only one application with just 10 concurrent users at a time. Since everything must now fit entirely in memory, we apply the usual rules of thumb for disk storage to the memory consumption. Applying a time-tested DBA rule of thumb for storage, the first three items in the above list (indexes, temporary tables and materialized views) roughly triple the storage requirements of any database, which increases the size requirement to 600 GB. Catalogs and compression dictionaries might require another 10 GB or so. The memory demands of the last category, working memory, increase with each concurrent user, and so it is harder to estimate and control. Suppose the database is a typical "star schema" dominated by a single, large fact table of just 160 GB (at 1000 bytes per row, this would contain only 160 million rows), leaving 40 GB for all the dimension tables. BI queries typically reference only a fraction (say, 10%) of each dimension table's columns, but the in-memory hash tables to join those dimensions with the fact table are typically decompressed, must have a fill factor under 50% to avoid long collision chains, and typically round up to a power of 2 hash buckets, so probably will consume memory almost the size of the base tables, or 40 GB per query. Ten concurrent queries therefore require 400 GB for joins, even before we account for any GROUP BY or ORDER BY clauses in those queries, and we've already exhausted the 1 TB of main memory without completing all of them!

Several major products not only assume that "it all fits", but require it! Such an in-memory requirement strictly limits the size of the database and/or the number of queries that can be run concurrently. In such products, the price of misestimating the combined memory demands for the database plus working memory can range from failing queries for want of working memory (as in the above example) to incomplete databases (missing tables). This paper describes BLU Acceleration, the analytics engine in IBM® DB2® for Linux, UNIX, and Windows (LUW) 10.5, and now also the cloud database called dashDB, that is optimized for in-memory execution of analytics queries, but is not limited to data that can be stored and/or processed in-memory. We first summarize in Section II what BLU Acceleration is, how it benefits users, and the extraordinary technology by which it achieves its speed, compression efficiency, and simple load-and-go operation. In Section III, we describe how BLU Acceleration both exploits abundant memory but also can carefully manage resources when concurrent users and/or databases compete for the available memory and/or CPUs. How BLU adapts to different machine architectures is the subject of Section IV. Sections V and VI highlight the latest features in DB2's new "Cancun" release (FP4) and dashDB, IBM's new cloud database that was announced in early November 2014. We give the availability of BLU Acceleration in Section VII, and our conclusions in Section VIII.

II. BLU Acceleration

BLU Acceleration is an execution engine for complex analytics queries that was designed and built from the ground up to read encoded (i.e. compressed), column-organized tables stored in a special BLU format on DB2 data pages, and to process that data, while still encoded, extremely efficiently by exploiting the characteristics of modern multi-core processors with huge memories. The BLU execution engine has been deeply integrated within DB2 LUW in a way that is almost completely transparent to the user, who only needs to decide which tables should be column-organized and which row-organized, but need not change any SQL statements in her applications. Although BLU is optimized for in-memory processing, it does not assume that all the pages of any column of any table fits or is currently located in memory – all pages of a table are stored on disk for persistence, and the storage system will access those pages on demand into a regular DB2 buffer pool using existing mechanisms. Main-memory synopses of the maximum and minimum value, per column, per every N rows, are automatically generated and maintained, to determine when pages need not be accessed. A novel probabilistic algorithm for buffer pool replacement determines which pages to victimize, ensuring that only hot pages of hot columns are retained in memory, maximizing the buffer pool's hit ratio.

Actually, the philosophy behind BLU Acceleration is that DRAM is too slow! Instead, all algorithms in BLU aim to keep data in the processor's (much, much faster) L3 or L2 caches by working on batches of rows called *strides*, and by partitioning data into L3 or L2 chunks for performing joins and grouping, as pioneered in Hybrid Hash Join [5] and MonetDB [6]. BLU attempts to keep data encoded (compressed) throughout query processing. As compressed pages are read, predicates are applied to values without having to decode them for most predicates [7], and values may even be re-encoded using "on-the-fly encoding" to perform joins or to minimize the bits required to represent smaller sets of values [8]. BLU employs several encoding schemes to reduce the values to mere bits, and packs them bit-aligned – not byte- or word-aligned – within words. As a result, multiple values for a column (sometimes tens of them) can usually be packed into a single word, achieving excellent compression, and can be processed together. Single-instruction, multiple data (SIMD) instructions are now common in modern processors, but are restricted to power-of-2 multiples of bytes. BLU enhances these SIMD instructions with novel software-SIMD algorithms to apply predicates simultaneously on all values in a word, for any code size. This SIMD parallelism is additional to that achieved by doling out strides of data to multiple threads running on multiple cores of each CPU [9]. To get good multi-core parallelism, BLU assigns to each thread independent data structures, or partitions thereof, designed to have low synchronization costs, such as a partitioned hash table or a latch-free hash set for assigning codes at query run time [10, 8]. BLU was a completely new design and carefully engineered to exploit this dense packing of values, cache-conscious algorithms, SIMD instructions, and assignment of independent data structures to each thread.

The result of this careful engineering is remarkable speed, compression, and simplicity. Entire workloads run on column-organized tables in BLU are typically 10 to 50 times faster than the same workloads run on the legacy row-organized run-time. Some long-running individual queries

TABLE I: IMPROVEMENTS OF BLU ACCELERATION (CANCUN RELEASE) VS. ROW-ORGANIZED TABLES OF DB2 10.1 FOR ENTIRE CUSTOMER WORKLOADS.

Customer	Improvement
U.S. medical insurance company	43x
U.S. transportation company	74x
U.K. consulting company	45x
German governmental authority	22x
German manufacturing company	20x

have seen speed-ups in excess of 1400 times! Table I summarizes some improvements of BLU over the previous version of DB2 (10.1) for different customer workloads.

BLU's compression schemes similarly improve storage savings of 2 to 3 times, on top of DB2's already excellent compression in 10.1. Some of this improvement can be attributed to the absence of secondary indexes in BLU, some to having dictionaries for each column rather than entire tables, and some to the variety of compression techniques that BLU automatically selects for each column. For many columns, BLU uses a scheme called *frequency compression* [7] that partitions the domain based upon the frequency with which values occur. Each partition has its own dictionary. The most frequently-occurring values are encoded using only a few bits and the less frequent values more bits. For high cardinality numeric domains, BLU uses *minus encoding*, i.e., encoding a value as its difference from one of a small number of dictionary values. Additionally, prefix compression can remove common prefixes from either numeric or character values. These compression schemes are also automatically customized at the level of a page, if the distribution of values on that page differs significantly from the table-wide distribution. A given column can use a mix of these encodings, and can also leave some values unencoded. These table-level compression schemes, which are in addition to the column-level schemes, also facilitate encoding new values introduced by updates since the creation of the column-level dictionary.

Lastly, and perhaps most importantly, BLU Acceleration significantly simplifies life for the DBA, achieving true "load and go" simplicity. Because no secondary indexes are needed or allowed, the usual tuning of queries is completely unneeded in BLU. In addition, BLU's outstanding performance eliminates the need for determining what materialized views to create and maintain. By setting a single configuration parameter, DB2_WORKLOAD to the value 'ANALYTICS', all the necessary database and system configuration parameters are optimized [11]. Statistics for optimization are automatically collected on all DB2 tables, including BLU tables, via a background daemon [12]. Even the number and speed of the CPUs and the size of their caches are automatically detected for BLU's cache-conscious algorithms. All of these result in significantly simplified administration for the DBA, and hence a very low time to value, while also ensuring accurate plan costing for the DB2 optimizer.

III. OPTIMIZED FOR IN-MEMORY, NOT LIMITED TO

Even though main memory sizes exceeding a terabyte are now common on servers, and have been increasing (almost) exponentially according to Moore's Law, DB2 with BLU Acceleration does not *require* that your databases, or any *part of it* (e.g., sub-range of a column), must be manually brought completely into memory, as some of its competitors do. There are a number of good reasons for this.

First of all, as detailed in the Introduction, memory is needed for much more than just the stored tables. And since memory now contains the primary copy of the data, the usual rules for storage consumption now apply to memory. Materialized views, temporary tables, and indexes often consume as much space as the base tables themselves, although BLU saves this space because none of these are required in BLU for good performance. Compression dictionaries, logs, and catalog tables will also consume significant main memory.

Secondly, any complex analytics query will require large amounts of working memory for hash tables used in large batch operations, such as joins and grouping, and temporary tables needed for sorts to implement ORDER BY, OLAP windowing functions, and some types of joins and grouping. Requiring the entire database to reside in memory effectively limits the remaining memory available for running queries, which therefore limits the number of queries that can be run concurrently. In the worst case, queries can fail for want of sufficient working memory.

Thirdly, the DRAM used for today's main memories is still orders of magnitude more costly per byte than magnetic disk, especially when we consider power consumption. Analytic queries typically access only a small fraction of the columns in each table, and only a sub-range of each column, so why pay dearly to store in DRAM the majority of "cold" columns that are rarely referenced in queries? Why not determine what rows of which columns need to be in the fast but expensive DRAM based upon what data queries reference, as BLU does, rather than dictating that all data – even data seldom referenced – must consume expensive DRAM? Some "in-memory" DBMSs even require the DBA to explicitly tell it which portions of which columns to "de-stage" to disk when available memory becomes tight!

Fourthly, DBMSs that argue "it all fits" force the DBA to determine which tables to store in memory and which to relegate to disk, which is a complicated, time-consuming, and error-prone process. As detailed above, the base tables are only a minor part of this calculation, and probably the easiest. The DBA also must calculate the space required for indexes, materialized views, metadata, logs, etc. Then she must allow for growth due to updates and new data, and additional indexes and materialized views needed in the future to enhance performance. Lastly, she must try to estimate the working memory needed for each query, and multiply this by the maximum number of queries that may run concurrently. None of this is required for BLU Acceleration, because BLU does this determination automatically based upon demand. It pages into buffer pools in memory only the pages for the columns that are needed, and uses automatically-created synopses to avoid accessing pages that can't possibly contain any data relevant to a query. And when the buffer pool becomes full, it automatically pushes out just "cold" pages, using time-tested victim selection algorithms.

Lastly, even in-memory databases need to be persisted to magnetic disk or solid state disks (SSDs) anyway, because DRAM is volatile. So mechanisms to move data to and from disk are needed for recovery from power outages or software failures.

Recognizing that main memory is still a scarce and expensive resource that must be utilized prudently, DB2 with BLU Acceleration employs a couple of mechanisms to limit needless overconsumption of resources.

Because a query can scalably exploit additional memory and/or cores to speed up its processing in BLU, a single query could readily "hog" the entire system. Conversely, allowing too many concurrent queries could deprive each query of sufficient cores and working memory to achieve BLU's dramatic performance for that query. Therefore, DB2 uses its Workload Manager (WLM) to manage the number of concurrent queries referencing BLU tables, so that each will have enough cores to achieve good parallelism and enough working memory to avoid unnecessary partitioning and spilling to disk, while allowing sufficient concurrency to optimize throughput and meet user expectations and service level agreements (SLAs).

BLU uses hash tables for major operations such as joins and grouping, and depending upon the size of tables and the query, these can dominate working memory.

For joins, BLU uses a novel and particularly frugal type of hash table called a Compact Hash Table (CHT), or, for dense domains, a Compact Array Table (CAT). These compact data structures construct a virtual hash table that has an extremely large number of buckets, and hence a low fill factor, to avoid costly chaining or processing due to collisions. However, only the non-empty buckets are actually stored, along with a bit vector that indicates which of the virtual buckets are filled. The stored bucket is indexed by the number of "1" bits in the bit vector before the bit corresponding to the desired bucket, which is first determined by the usual hashing. In Compact Array Tables, even the key value is omitted from each bucket, because keys are mapped to buckets isomorphically [10].

For grouping, compact hashing cannot be used, because the groups are inserted into the hash table incrementally and randomly. An additional challenge is that the number of distinct groups is difficult for the optimizer to estimate reliably, and can range from just a handful (as in TPC-H queries) to as many groups as there are input rows. BLU uses a two-phase algorithm. In the first phase, rows are hashed to a small initial hash table, gambling that few distinct groups occur. If collisions occur, they are spilled to a set of overflow blocks, which are partitioned by the grouping key for better cache locality. In the second phase, these overflow blocks and the initial hash table are then merged into global hash tables for each partition, whose size is estimated very accurately using low-memory probabilistic counters maintained during the first pass, and which can then be simply UNIONed together [9].

These mechanisms to use memory frugally, together with keeping the data values encoded and densely packed in words as long as possible, permit BLU to carefully orchestrate high throughput by maximizing concurrency and system utilization.

IV. Multi-Platform Architecture

DB2 with BLU Acceleration is one of the few columnar memory databases that have been optimized to run on multiple distinct computer architectures. Originally released in 2013 for both the IBM POWER and Intel x64 architectures, it was extended in 2014 to the IBM System z mainframe architecture. IBM's POWER8 processor is now bi-endian and able to support little-endian Ubuntu 14.04. IBM has released a beta of DB2 BLU for little-endian Linux on POWER, and announced the formation of the OpenPOWER Foundation open development community dedicated to creating an open ecosystem using the POWER architecture [13]. The wide range of hardware architectures on which BLU can run creates both opportunities and challenges. Two particular dimensions of note are the degree of simultaneous multi-threading (also known as *hyper-threading*) and the differences in the number and size of the caches in the processor cache hierarchy.

As shown in Table II, there is a significant difference in the possible degree of multi-threading. While typically beneficial in traditional environments, the cache-aware design of DB2 BLU Acceleration reduces the benefit of multiple hardware threads. More threads in fact create more pressure on the caches. The 2x or greater variance in L1-L3 caches, plus the additional benefit of the L4 cache in POWER8, give that processor an edge in retaining larger strides in cache.

BLU Acceleration uses low-level operating system calls and deciphers the information returned by the CPUID call to understand the cache characteristics, the presence of hyper-threading, and the capabilities, if present, of SIMD on the chip.

IBM has already started exploring additional acceleration ideas, such as NVidia GPUs in S824L POWER8-based servers [14]. This will present additional opportunities for further accelerating DB2 BLU on certain hardware architectures for which GPU support is available.

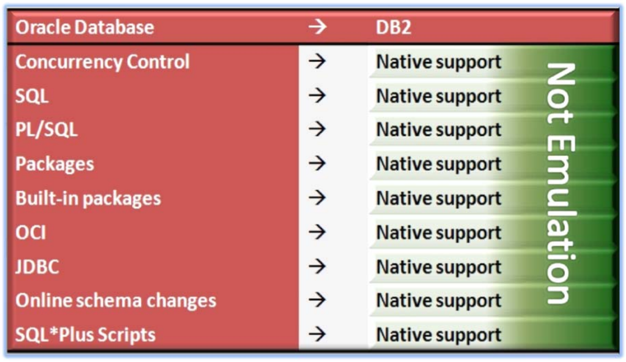
TABLE II: Hardware properties for various BLU platforms.

	Haswell EP E5-26xx v3	**Ivy Bridge EX E7-88xx v2**	**POWER 7+**	**POWER8**	**System z EC12**
Clock rates	**1.7-3.7GHz**	**1.9-3.4 GHz**	**3.1-4.4 GHz**	**3.0-4.15 GHz**	**5.5GHz**
SMT options	**1, 2**	**1, 2**	**1, 2, 4**	**1, 2, 4, 8**	**1**
Cores per socket	**18**	**15**	**8**	**12**	**6**
Max Threads / socket	**36**	**30**	**32**	**96**	**6**
Max L1 Cache	**32KB**	**32KB**	**32KB**	**64KB**	**160KB**
Max L2 Cache	**256 KB**	**256 KB**	**256 KB**	**512 KB**	**2MB**
Max L3 Cache	**45 MB**	**37.5 MB**	**80 MB**	**96 MB**	**48MB**
Max L4 Cache	**0**	**0**	**0**	**128 MB**	**384MB/ book**

V. New Features in “Cancun” Release of DB2

The DB2 Cancun release in August 2014 (officially DB2 10.5 FP4) added major extensions to BLU Acceleration for query performance, ELT (extract, load, and transform), high availability (HA), mixed workloads for OLTP and reporting, and SQL compatibility for applications [15]:

- **Query Performance.** DB2 Cancun is significantly faster than the original version of BLU Acceleration. The TPC-DS workload is notably more than 3 times faster. There are multiple examples of customers reporting more than 100x workload speed-up, and individual query speed-ups in excess of 1400x. The performance improvement since the original introduction of BLU Acceleration in DB2 10.5 come from a variety of enhancements, including improvements to scan efficiency, aggregation, join, common subexpression handling, and query rewrites.
- **ELT.** Point UPDATE and DELETE processing have improved by as much as 112 times. Central to this speed-up has been the exploitation of enforced primary keys and uniqueness constraints, along with a concomitant ability to perform singleton lookups against them in log-n processing time.
- **High Availability.** DB2 Cancun extends the High Availability Disaster Recovery (HADR) feature of DB2 to BLU’s column-organized tables. As reporting and business intelligence functions become increasingly mission critical, the industry as a whole is seeing reduced tolerance for system outages for data warehouses and data marts. By extending DB2’s mature HADR capability to BLU’s column-organized tables, users now have a tried-and-true environment for keeping the benefits of BLU Acceleration available in the presence of system failure or geo-local disasters.
- **Mixed OLTP & Reporting.** DB2 Cancun introduces *Shadow Tables* to support high performance on workloads mixing OLTP and analytics reporting. Shadow Tables allow unhindered OLTP transactions for low-latency operation under high concurrency, while simultaneously enabling simple high-performance reporting and BI on the same schema by leveraging BLU Acceleration. Applications perceive a single schema, while the physical implementation includes dual representation of the data in row and columnar format. As SQL statements arrive, statements with write operations are routed to the base table (row-organized) while read-only operations are conditionally routed to either the base table or its columnar shadow based on a costing decision determined at statement compilation time. The columnar shadow can be defined as a subset of the schema from the row-organized data, i.e., a subset of the tables and/or a subset of the columns in any individual table. The Shadow Table does not require all of the user data to fit into memory for either the row-organized base table or its column-organized shadow. In most cases, database memory for the column-organized data can be less than 5% the size of the original user data and run with in-memory optimized performance – customers have reported examples of 10-40 times speed-up on reporting queries. In many cases, the OLTP performance with Shadow Tables has exceeded that of a purely row-organized table, because the large number of secondary indexes that are required for efficient queries on the row-organized table are no longer needed with Shadow Tables. The concomitant reduction in index maintenance is usually larger than the processing required to synchronize the dual representation of Shadow Tables. Similarly, Shadow Tables generally require a smaller disk footprint than a pure row-organized architecture, due to the reduction in secondary indexes.
- **SQL Compatibility.** DB2 Cancun extends to BLU’s column-organized tables the existing SQL Compatibility mode that was introduced in DB2 9.7 for row-organized tables. These extensions enable several non-standard language and scripting elements commonly used in the industry by ISVs and DBMS products such as Oracle™. Examples include the PL/SQL stored procedure language, OCI, and SQL*Plus scripting. A short summary of these compatibility attributes is shown below in Figure 1. By providing native support, DB2 avoids the processing overhead that would likely be incurred by an emulation layer.



Oracle Database	→	DB2
Concurrency Control	→	Native support
SQL	→	Native support
PL/SQL	→	Native support
Packages	→	Native support
Built-in packages	→	Native support
OCI	→	Native support
JDBC	→	Native support
Online schema changes	→	Native support
SQL*Plus Scripts	→	Native support

Not Emulation

Figure 1: SQL Compatibility attributes in DB2 Cancun

VI. BLU for the Cloud: dashDB

IBM recently announced the launch of dashDB [15], its premier data warehouse service in the cloud, which includes the BLU Acceleration technology from DB2 Cancun combined with the in-database analytics of the Pure Data for Analytics (Netezza) [16] line, and having several important features:

- dashDB is a managed, pay-as-you-go, use-only-what-you-need service. As a managed service, it provides load-and-go simplicity, eliminating not only the need for user-initiated configuration and tuning, but it also eliminates the need for manual provisioning of components (servers, storage, etc.), software installation, or routine administration such as backups. As a managed service, IBM takes care of the deployment, configuration, security, and high availability for users, as well as regular maintenance tasks such as backup and restore.
- It is in-memory optimized for high performance, with BLU Acceleration in its kernel.

- It provides in-database analytics for statistical analytics (R) and spatial analytics (Esri), by leveraging the Netezza technology in Pure Data for Analytics, bringing your analytics to the data, not the data to the analytics.
- It is Cloudant and JSON ready [18]. Cloudant and dashDB services are tightly coupled, providing a trivial interface to instantiate a dashDB instance and populate the database with JSON automatically extracted from Cloudant.
- It's a composable service on IBM BlueMix [19], so you can easily connect your dashDB data with other cloud services from IBM, such as IBM DataWorks, Cloudant, and others.

A. *Always secure and encrypted*

A major consideration for cloud adoption is confidence that the data will be secure when it is stored in the cloud, and when it is transferred in and out of the cloud. dashDB always stores data and transfers data in encrypted form. The encryption of data at rest (stored) applies both to data stored in the database and data inside backup images. The data in the database and the backups (which are automated) are encrypted with AES 256, which uses the strongest key size known today within the industry-standard symmetric cryptography. Database activities are continuously monitored, and that monitoring is made available to customers easily through the dashDB console. The activity reports include who is connecting to the database and what actions they have performed. The dashDB user console supports HTTPS, so these communications are also encrypted. Add to that the classic access control that you can grant to decide deliberately who and what has access to your data, and you have data at rest and in transit that is always trusted and only available to whom it should be.

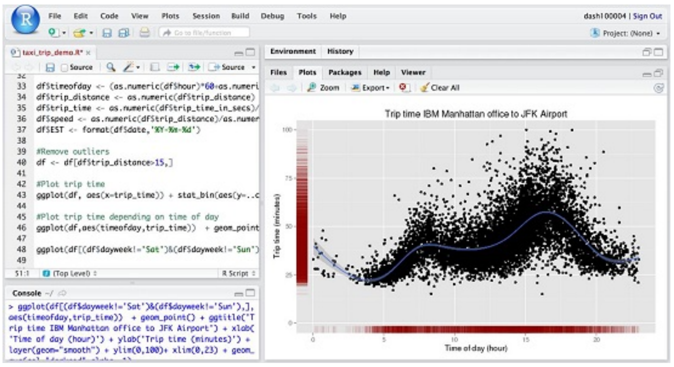
B. *Analytics for Mobile Apps with Cloudant and JSON data*

dashDB enables 'mobile analytics' with Cloudant. Cloudant is IBM's highly-available and elastic JSON store in the cloud. The connection with Cloudant.com means that a dashDB data mart or warehouse can combine the usual SQL-based services, such as Tableau or Cognos, with JSON data from Cloudant. A single button on the Cloudant console will automatically create the dashDB warehouse service, mine the existing JSON data to discover a relational schema for it, load the JSON data into dashDB as a bulk load task, and subsequently establish continuous replication of incoming JSON documents. This process provides a simple interface to leverage dashDB to execute reporting, data visualization, or geospatial analytics against data from a JSON store, and to leverage the in-memory optimized processing power of BLU Acceleration.

C. *Combining Pure Data for Analytics (Netezza), BLU Acceleration, and IBM's Cloud (BlueMix and Softlayer)*

IBM Bluemix is an implementation of IBM's Open Cloud Architecture. It leverages Cloud Foundry to enable a rapid development ecosystem for applications by providing a rich collection of composable cloud services and runtime frameworks. dashDB combines the best of Netezza simplicity and in-database analytics with the performance and functional richness of BLU Acceleration. It brings these qualities to the cloud as a managed service on BlueMix, so it is a composable component with many other cloud services. While BLU Acceleration has focused on simplicity and performance for software-based in-memory warehousing, the Netezza technology established itself with a focus on simplicity, raw performance, price performance, and in-database analytics in the appliance space. Drawing from this heritage of in-database analytics, dashDB has placed significant focus on both R and spatial analytics.

Figure 2 illustrates dashDB integration with R Studio, while Figure 3 illustrates dashDB usage with spatial analytics such as Esri. dashDB brings these technologies together to enable cloud simplicity, elasticity, availability with in-database analytics, and in-memory optimized SQL.


```
df$timeofday <- (as.numeric(df$hour)*60+as.numeri
df$trip_distance <- as.numeric(df$trip_distance)
df$trip_time <- as.numeric(df$trip_time_in_secs)/
df$speed <- as.numeric(df$trip_distance)/as.numer
df$EST <- format(df$date,'%Y-%m-%d')

#Remove outliers
df <- df[df$trip_distance>15,]

#Plot trip time
ggplot(df, aes(x=trip_time)) + stat_bin(aes(y=..c

#Plot trip time depending on time of day
ggplot(df,aes(timeofday,trip_time))  + geom_point

ggplot(df[(df$dayweek!='Sat')&(df$dayweek!='Sun')
```

```
> ggplot(df[(df$dayweek!='Sat')&(df$dayweek!='Sun'),],
aes(timeofday,trip_time))  + geom_point() + ggtitle("T
rip time IBM Manhattan office to JFK Airport") + xlab(
'Time of day (hour)') + ylab('Trip time (minutes)') +
layer(geom="smooth") + ylim(0,100)+ xlim(0,23) + geom_
```

Figure 2: R Studio use with dashDB

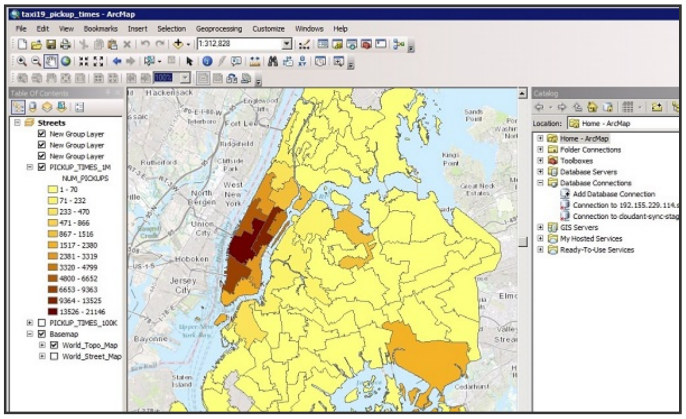


Figure 3: dashDB with Esri ArcMap

VII. BLU AND DASHDB AVAILABLE

IBM has had a long-standing initiative to bring IBM Software to universities for research and teaching [20]. DB2 BLU is available for download from the academic initiative web site, along with many other pieces of software. Furthermore, the associated End-User License Agreement (EULA) in DB2 has no "DeWitt clause" [21] that restricts the publication of performance results, so we welcome experimentation with DB2 BLU by students and researchers. Similarly, a free version of dashDB is available online [16].

VIII. CONCLUSIONS

BLU Acceleration is a new run-time in DB2 LUW and the dashDB cloud offering that was engineered from the ground up to execute analytics queries in-memory, but not to require that all data be in-memory. Furthermore, it exploits the entire memory hierarchy, and especially CPU cache, to speed queries even more than in-memory, while also being cost-effective.

ACKNOWLEDGMENTS

The authors wish to acknowledge the entire DB2 LUW development team, who designed, developed, enhanced, tested, and made BLU the robust product it is today. We also thank Michael Kwok for reviewing an earlier draft of this paper.

REFERENCES

[1] M. Stonebraker et al. C-Store: "A column oriented DBMS", *Proc. of the Very Large Data Bases Conf.,* 2005.

[2] F. Färber, N. May, W. Lehner, P. Große, I. Müller, H. Rauhe, J Dees, "The SAP HANA Database -- An Architecture Overview", *IEEE Data Eng. Bull. 35(1)*, pp. 28-33, 2012.

[3] C. Diaconu, C. Freedman, E. Ismert, P. Larson, P. Mittal, R. Stonecipher, N. Verma, M. Zwilling, "Hekaton: SQL Server's memory-optimized OLTP engine", *Proc. Of the ACM SIGMOD Conf.*, 2013.

[4] A. Kemper, T. Neumann, J. Finis, F. Funke, V. Leis, H. Mühe, T. Mühlbauer, W. Rödiger, "Processing in the Hybrid OLTP & OLAP Main-Memory Database System HyPer", *IEEE Data Eng. Bull. 36*(2), pp. 41-47, 2013.

[5] D.J. DeWitt, R. Katz, F. Olken, L. Shapiro, M. Stonebraker, D. Wood, "Implementation techniques for main memory database systems". *Proc. ACM SIGMOD Conf* **14** (4): pp. 1–8 (June 1984).

[6] S. Manegold, P. Boncz, and M. Kersten. "Optimizing database architecture for the new bottleneck: memory access", *VLDB Journal*, **9**(3), 2000.

[7] V. Raman, G. Swart, L. Qiao, F. Reiss, V. Dialani, D. Kossmann, Narang, and R. Sidle, "Constant-time query processing," *Proc. IEEE Intl. Conf. on Data Engineering*, 2008.

[8] Jae-Gil Lee et al. "Joins on Encoded and Partitioned Data", *Proc. of the Very Large Data Bases Conf.,* 2014.

[9] V. Raman et al., "DB2 with BLU Acceleration: So much more than just a column store", *Procs. of the Very Large Data Bases Conf.* **6**, 2013.

[10] R. Barber, G. Lohman, I. Pandis, V. Raman, R. Sidle, G. Attaluri, N. Chainani, S. Lightstone, D. Sharpe, "Memory efficient hash joins*", in press, Procs. of the Very Large Data Bases Conf.*, 2015.

[11] E. Kwan, S. Lightstone, K. B. Schiefer, A. Storm, and L. Wu, "Automatic database configuration for DB2 Universal Database: compressing years of performance expertise into seconds of execution, *Proc. of BTW Conf.*, pp. 620–629, 2003.

[12] A. Aboulnaga, P.J. Haas, S. Lightstone, G.M. Lohman, V. Markl, I. Popivanov, V. Raman, "Automated Statistics Collection in DB2 UDB", *Proc. Of the Very Large Data Bases Conf. 2004*, pp. 1146-1157

[13] http://openpowerfoundation.org/

[14] http://cloudtimes.org/2014/10/08/ibm-introduces-openpower-super-computing-for-big-data-analytics/

[15] IBM BLU Acceleration http://ibmbluhub.com

[16] IBM dashDB http://dashdb.com

[17] IBM Pure Data for Analytics (Netezza) http://www-01.ibm.com/software/data/puredata/analytics/

[18] IBM Cloudant http://cloudant.com

[19] IBM BlueMix http://bluemix.net

[20] http://www.ibm.com/ibm/university/academic/pub/page/academic_initiative

[21] http://en.wikipedia.org/wiki/David_DeWitt