PARTNERS
The Teradata User Group

How to compare tables between TD systems in a dual active environment

Ulrich Arndt
Managing Director, data2knowledge GmbH

The 2012 Teradata PARTNERS
User Group Conference & Expo

DECISION POSSIBLE

Who we are

- Small and independent, specialized and innovative German consulting company.
  - > DWH / Data Mining architecture
  - > Performance Tuning
  - > Prototyping and Development
  - > SQL specialists

- TD experience since 1996

www.data2knowledge.de

PARTNERS The Teradata User Group

DECISION POSSIBLE

Agenda

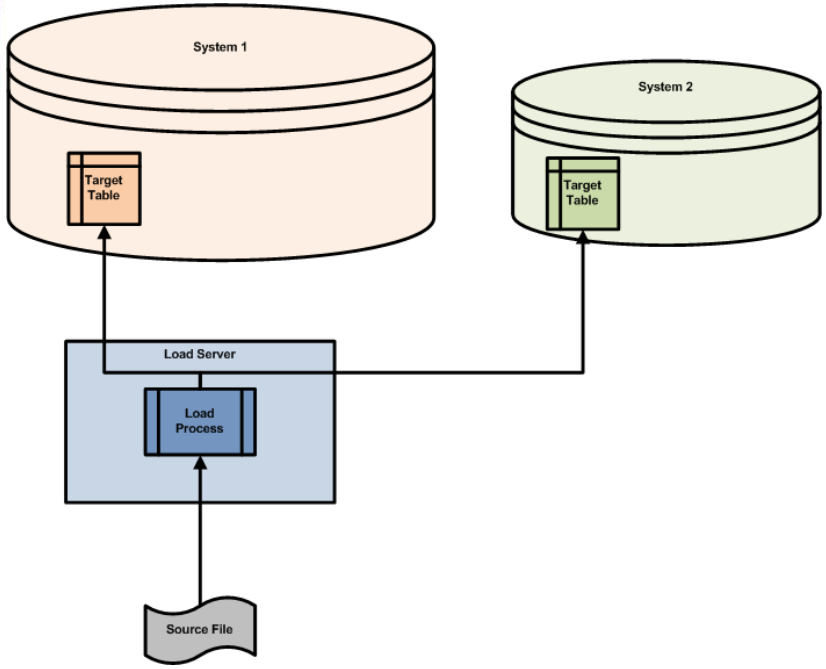
- **Introduction**
  - Problem Description
  - Classical Approach
  - Parent Child Problem
  - Alternatives
  - Test Environment / Test Setup
- Classical Hash based approach
- In database table hash calculation
- Outline / Other use cases

Introduction
Problem Description

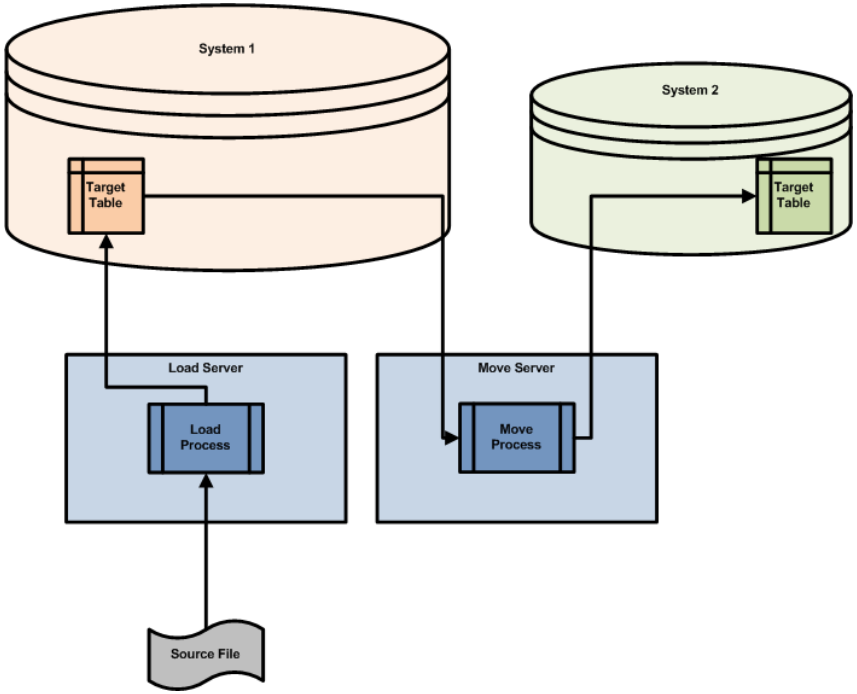
- You have a dual active environment where some tables exist on both systems and should have the same content.
- Different load and maintenance strategies are possible:
  - Dual load / Unity
  - Single load and transformation and copy final result sets to second system

Introduction
Problem Description

Dual Load



Load once and transfer



Introduction
Problem Description

- How can you assure that both tables have the same content?

  Joining is not possible.

- Can you sign with blood / can you prove that your tables are in sync?

Introduction
Classical Approach

- A currently widely used approach is to select Counts, SUMs, AVG, STDDEV_POP etc. for each column of the table and compare the results.
- Something like

```
Select COUNT(*),
       AVG (calendar_date - cast (' 2000-01-01' as date)),
       STDDEV_POP((calendar_date - cast (' 2000-01-01' as date))),
       AVG (day_of_week),
       AVG (year_of_calendar )
       from sys_calendar.calendar;

Count(*) calendar_date    calendar_date    day_of_week  ... year_of_calendar
73414    182.5            21192.           84.          ... 2000.
```

Classical Approach

- What are the issues with this approach?
  - **Precision**

```
Select count(*),
       AVG ((case when calendar_date =
                              cast (' 2000-01-01' as date)
                  then calendar_date - 1
                  else calendar_date end)
             - cast (' 2000-01-01' as date)) as AVG_VAL,
       STDDEV_POP(((case when calendar_date =
                              cast (' 2000-01-01' as date)
                         then calendar_date - 1
                         else calendar_date end)
             - cast (' 2000-01-01' as date))) as STDDEV_VAL
       from sys_calendar.calendar;
```

Change a single value

```
Count(*)        AVG_VAL        STDDEV_VAL
73414           182.5          21192.8
vs.
Count(*)        AVG_VAL        STDDEV_VAL
73414           182.5          21192.
```

Classical Approach Precision

- What is going to happen in a 1.6 billion row table?

```
Select count(*),
       STDDEV_POP(term_in_gutenberg) ref_Stddev,
       STDDEV_POP(case when term_in_gutenberg = 100209032
                       then term_in_gutenberg-1
                       else term_in_gutenberg
                  end) one_diff_Stddev
from MyDataDB.gutenberg ;
```

```
Count(*)          1,650,021,426
Ref_Stddev        4.7632015722695 42E8
One_diff_Stddev   4.7632015722695 42E8
```

No change is indicated

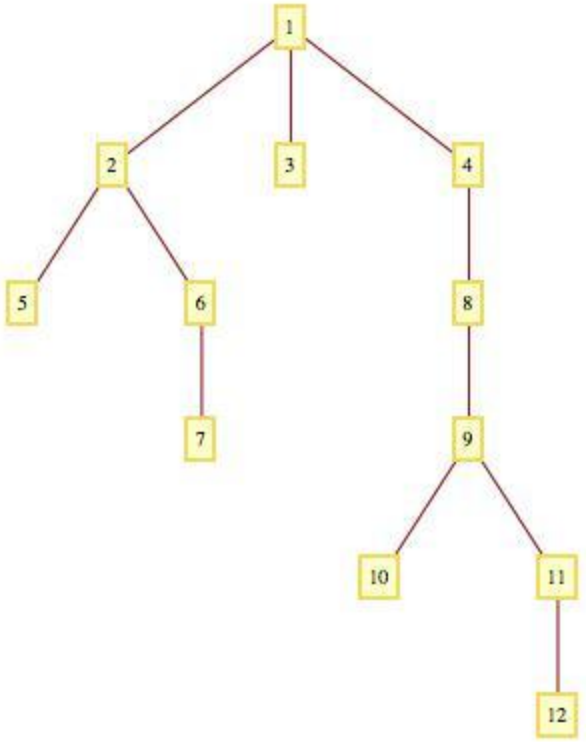
Classical Approach

- What are the issues with this approach?
  - Precision 
  - Data Types
    - How to measure Chars?
      Only length? – Any change in content not affecting length would never be detected.
    - Periods?
    - Dates/Timestamps?
  - Some specific problems will never be detected 

Introduction
Parent Child Problem

- Consider a simple Parent Child relation table.

Parent	Child
1	2
1	3
1	4
2	5
2	6
6	7
4	8
8	9
9	10
9	11
11	12

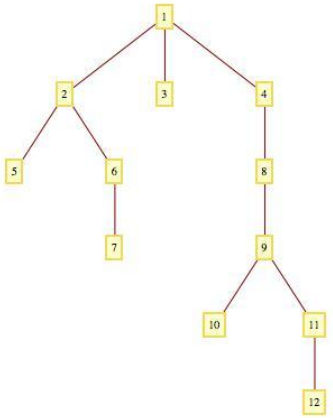


Introduction
Parent Child Problem

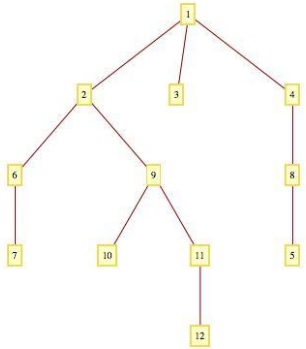
- Now 9 becomes a child of 2 and 5 of 8.
- This represents clearly a different graph or structure.

Parent	Child
1	2
1	3
1	4
8	5
2	6
6	7
4	8
2	9
9	10
9	11
11	12

before



after



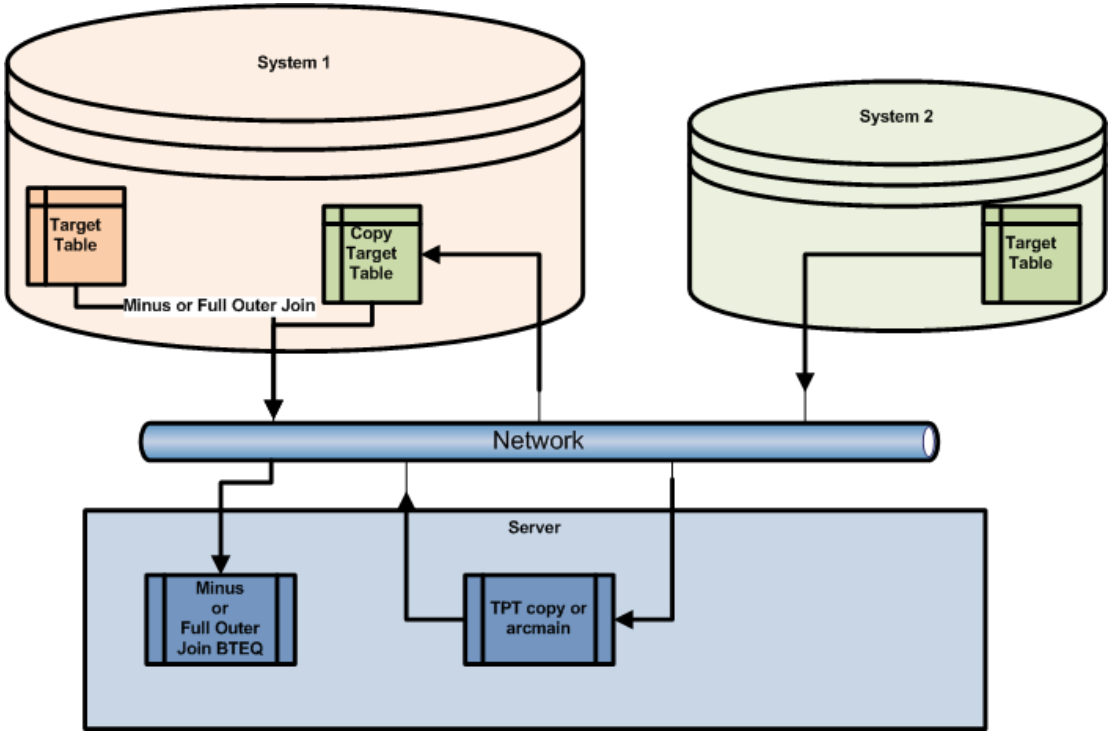
- But any column measure would be unchanged.

Introduction
Other Checksum

- On http://developer.teradata.com/ecosystem/articles/introducing-tdm-into-a-production-dual-system-environment we see a different approach.
- select SUM(CAST(HASHBUCKET(HASHROW(columnlist)) AS DECIMAL(38,0))) from source_db;
- Heading into the right direction but HASHBUCKET(HASHROW()) has far too much hash synonyms! >4000 in sys_calendar.calendar – 73414 rows.
- Easy to end up with the same checksum.
  Change 1990-11-30-> 1990-11-29
  and 2032-10-07->2032-10-06
  and table will have the same checksum 38499506369

Introduction
Alternatives

- Move one of the tables to the other system and compare via MINUS or FULL OUTER JOINS



- Good idea?

Introduction
Alternatives

- Cost factors
  - SQL to fetch the data from first system
  - Network traffic to unload
  - TPT / Archive server resource cost
  - Network traffic to load
  - Load / Restore on second system
  - Minus or Full Outer Join SQL

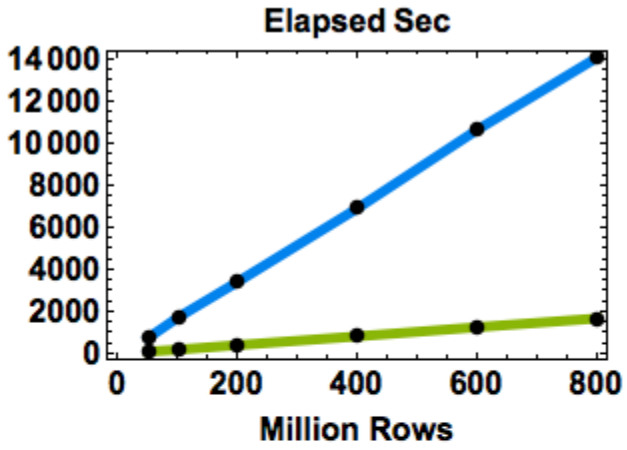
Introduction
Alternatives

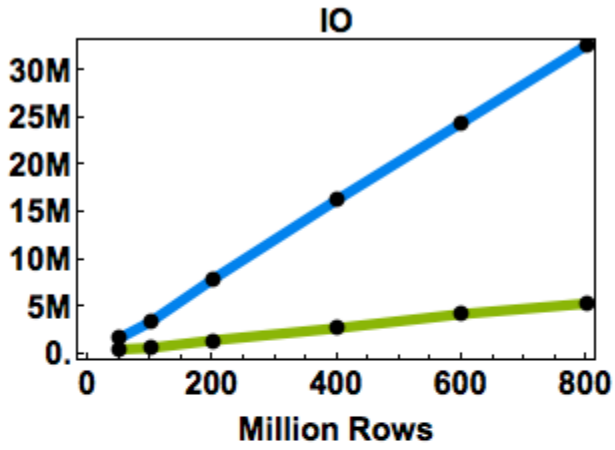
- Benefits
  - In case of differences the **differences are part of the output**.

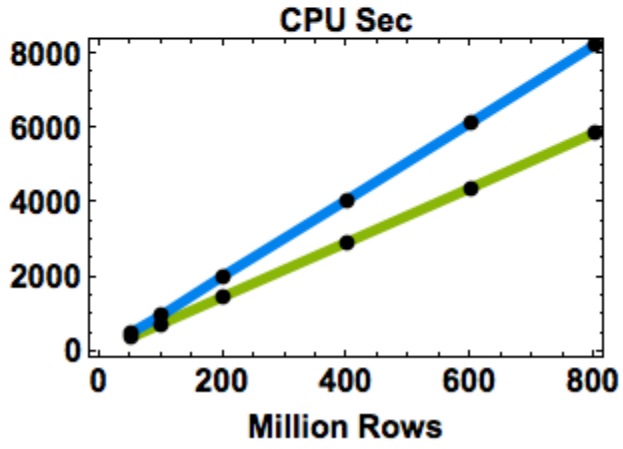
- Disadvantages
  - **Complex** multi step process.
  - **Not efficient** for many tables.
  - **Costs on different levels** (DB, Network, ETL Server)

    Very likely that you **hit your bottleneck** – wherever it is.

Introduction
Alternatives - Resource Consumption

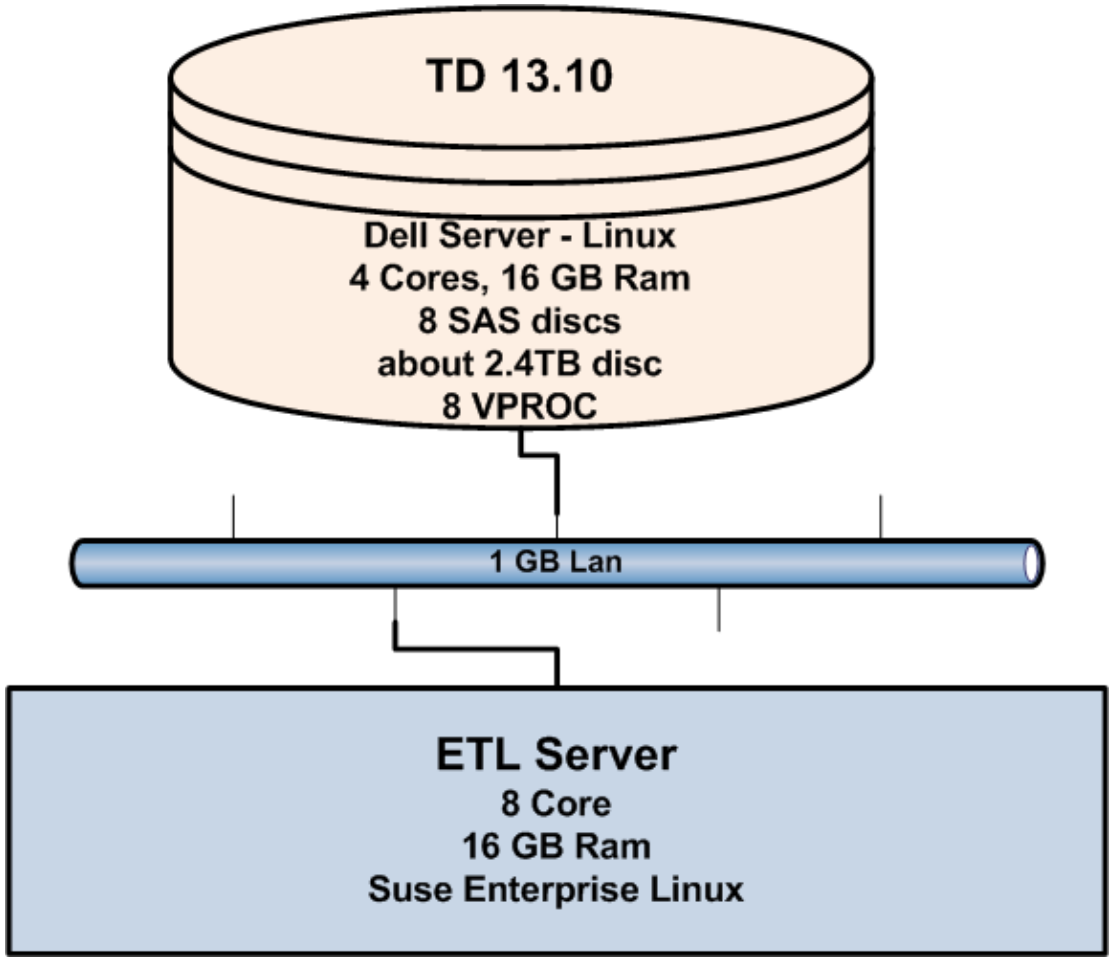






Introduction
Test Environment

- Single user dedicated system



Introduction
Test Set Up

- One common table DDL with two scenarios:
  - Different number of rows (50, 100, 200, 400, 600, 800 million)
  - Same number of rows (100 million) but different avg length in ITEM column (50, 150, 250, 450 CHAR's)

```
CREATE SET TABLE tbl_100m_src
      (
       item_group CHAR(1) ,
       item_subtype CHAR(10),
       item_base VARCHAR(100),
       item_type CHAR(10),
       item VARCHAR(500),
       file_id SMALLINT,
       sentence_in_file SMALLINT,
       sentence_in_bnc INTEGER,
       item_in_sentence SMALLINT,
       item_in_file INTEGER,
       item_in_bnc INTEGER)
UNIQUE PRIMARY INDEX ( item_in_bnc )
;
```

Agenda

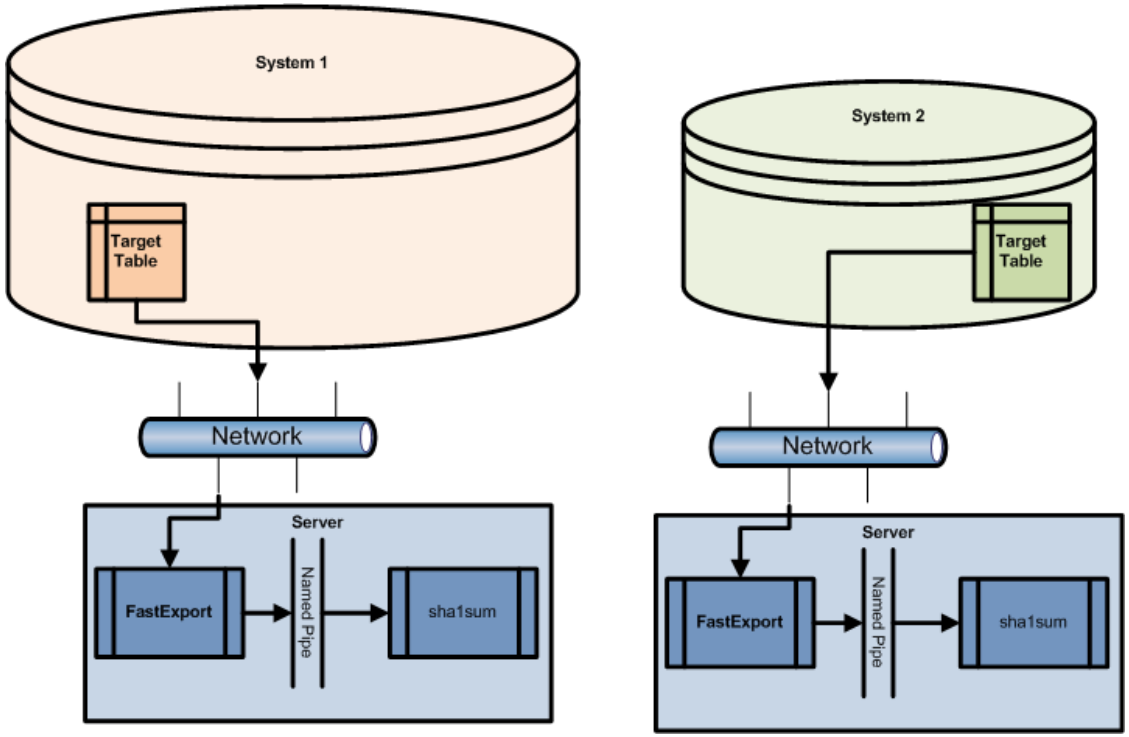
- Introduction
- **Classical Hash based approach**
- In database table hash calculation
- Outline / Other use cases

Classical Hash Based Approach Question

- Why can’t we calculate a **hash value** for a **table** as we do it for **files** for a long time?
- Because **hash algorithms** like MD5 and SHA1 depend on the **order** of information!
- Within the **relational database we don’t have ordered information**.
- Especially between two different systems where we might have different HW configurations and number of Vprocs.

Classical Hash Based Approach

- What we can do is to export the **sorted** data into a pipe and calculate the SHA1 checksum there.

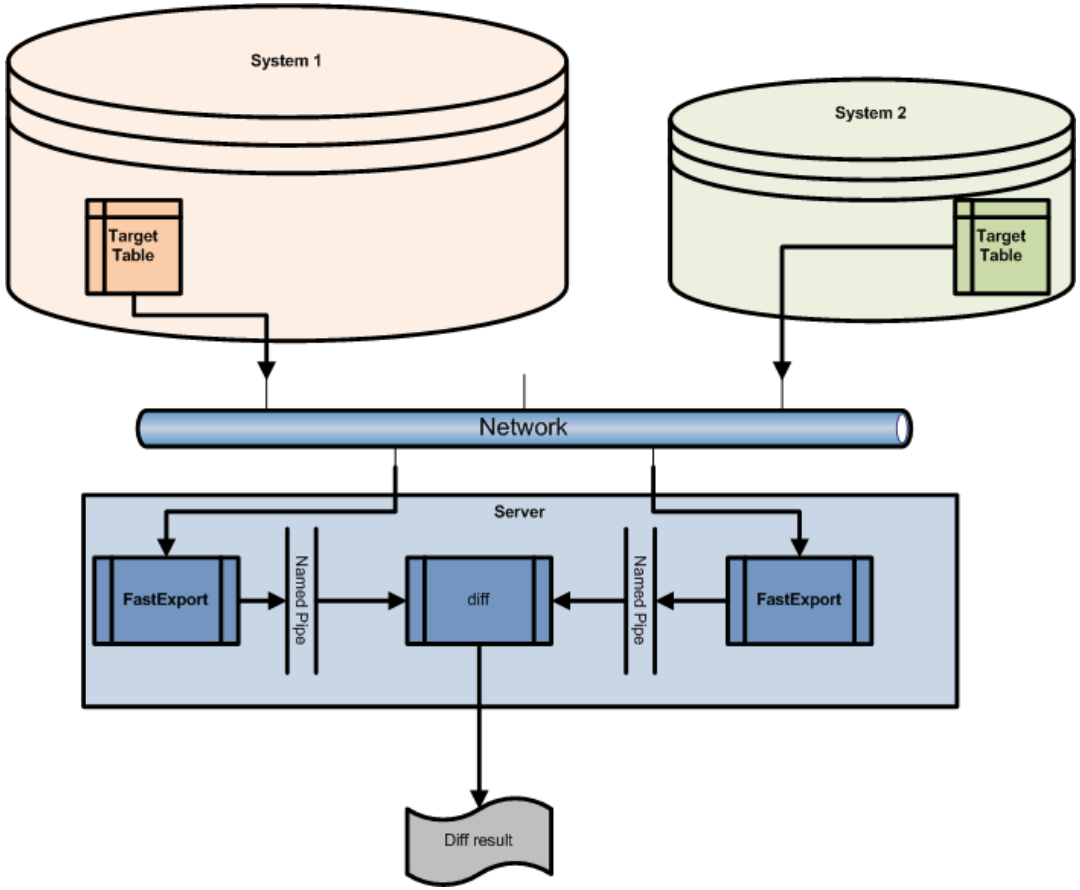


Classical Hash Based Approach

- Cost factors
  - 2 x **SQL incl. Order By** to fetch the data
  - 2 x **Network traffic** to unload
  - 2 x checksum calculation
- Disadvantages:
  - **Costs of Sort**
  - **A multi step process**
  - Only table content “is the same” or “is different” info.
- Advantages:
  - No additional space requirements on second system due to data replication.

Classical Hash Based Approach Alternative - Diff

- Export the **sorted** data into two pipes and calculate the diff which is stored in a file.



Agenda

- Introduction
- Classical Hash based approach
- **In database table hash calculation**
- Outline / Other use cases

Database Table Hash Calculation Question

- The **order** requirement **prevents** us from using the plain **SHA1 or MD5 algorithm** as **DB internal hash function**.

- Can we come up with a different algorithm which has the same nature as SHA1/MD5 hash but does not depend on the order of columns?

  > Yes

Database Table Hash Calculation What is needed?

- A **deterministic** function
  (same input -> same output)
  which **generates pseudo random numbers** based on **row input** where the likelihoods of collisions (different input generates the same output) is **very, very, very small or which is collision free**.

Database Table Hash Calculation What is needed?

- A **commutative** and **associative** function which **aggregates the row results** of the deterministic pseudo random function.

- **Commutative**
  - > **A ^ B = B ^ A**

- **Associative**
  - > **(A ^ B) ^ C = A ^ (B ^ C)**

Database Table Hash Calculation
What is needed?

- Sounds complicated?
  - > We want to get an aggregation function which calculates row hash values and “aggregates” these.
  - > Lets get concrete!

- Do we have candidates for the two needed functions?

  - > We use **SHA1** for the row transformation.

Database Table Hash Calculation SHA1

- SHA1 gives a 160 Bit array for each input.
- SHA1 is designed in a way that small (a single bit) differences in input will result in very different output.
- The function has a result room of 2^160 different values.

  or **1:1461501637330902918203684832716283019655932542976**

  49 digits – near the number of atoms on earth.

Database Table Hash Calculation

- So we calculate a SHA1 hash value per row.

Row	calendar_date	day_of_week	day_of_month	day_of_year	day_of_calendar
Row01	1987-09-25	6	25	268	32 044
Row02	1931-01-13	3	13	13	11 335
Row03	1978-03-19	1	19	78	28 567
Row04	1933-08-16	4	16	228	12 281
Row05	1930-07-05	7	5	186	11 143
Row06	1963-07-30	3	30	211	23 221

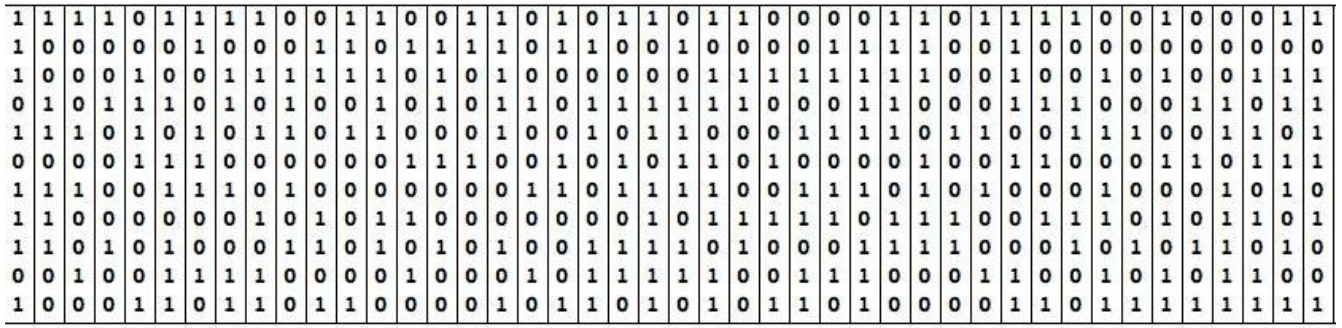
becomes

```
Row01: 11100110011010110110000110111100100011101110111100110110110111101...
Row02: 10001101111011001000011110010000000000011010101001111111110111000...
Row03: 01111110101000000111111110010010100111101001010101010100010101110 0...
Row04: 01010010101101111110001100011100011011001101111111011110000011011 1...
Row05: 10110110001001011000111101100111001101011111100001001101110010110 1...
Row06: 10000001110010101101000010011000110111011010011111101111110110101 1...
```

We did this by concatenating all colums to one string – incl. special null handling via coalesce.

Database Table Hash Calculation Aggregation

- The bitwise **XOR** has the required characteristics of being commutative and associative.
- Within the two dimensional bit array XOR operates on columns. The output is 1 in case the count of 1 is odd, else 0.



Result:

0	0	0	1	1	1	0	0	0	1	1	1	0	0	1	1	1	1	1	0	0	0	1	0	1	0	0	0	0	1	1	1	0	1	1	1	0	1	0	0	1	0	1	1

- We will represent this hash as hex value of length 40.

PARTNERS The Teradata User Group

DECISION POSSIBLE

Database Table Hash Calculation

- The new aggregation hash function has the following properties
  - If two output values are different then the table content is different!
  - If two output values are the same then the table content is different with the probability of 1/2^160 for SHA1.

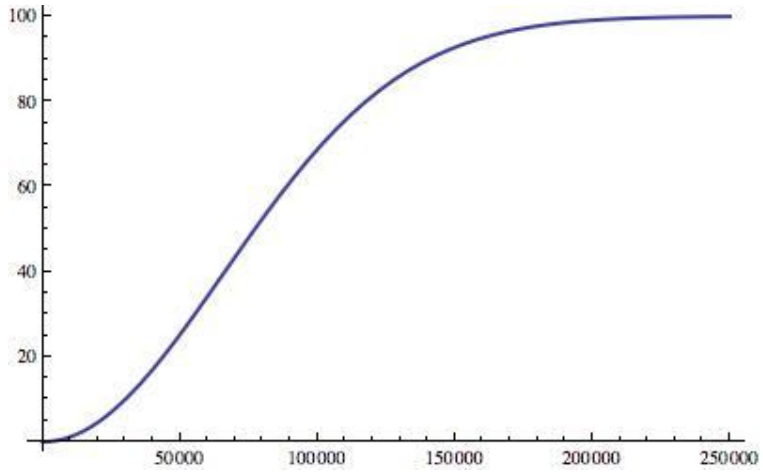
So the function has similar properties as SHA1

Database Table Hash Calculation
Some Theory

- We can consider each bit of the hash output as a Bernoulli-distribution with Prob=1/2 (Ж) distributed pseudo random variable.
- A XOR combined list of Ж distributed random variables is also a Ж distributed random variable – independent from the length of the list!
- That means we have the same likelihood that two different tables will result in the same output – 1:2^160
  Where this **likelihood is independent from the number of rows** in the tables.

Database Table Hash Calculation Likelihood of Hash Collisions

- The likelihood of hash collisions (different input result in same output) increases with the number of rows in the table.
  N: Number of possible value
  k: Number of rows
  1 – e^((-k(1-k)/2N)



- For 32 bit hashes
  we get a 50% Chance
  for hash collisions
  with 77163 rows - 0.00179657%
  of all possible hash values.

Database Table Hash Calculation Likelihood of Hash Collisions

- That means we need long hash values to minimize the risk of hash collisions.

- Let's assume 1000 billion rows.
  - MD5 1.44329*10^-13%
  - SHA1 too small to compute…

-> For long hash values hash collisions are not an issue.
But everything shorter then MD5 should not be considered.

Database Table Hash Calculation

- Any known issues with **XOR** function?
- **XOR can only be used for SET tables** as aggregation function!

  Duplicate rows will end up in a **constant 0** bit array which would mean that they can be changed in any way without resulting in a different hash value.

1	0	1	0	1	0	0	1	0	1	0	0	1	0	1	0	1	0	0	0
1	0	1	0	1	0	0	1	0	1	0	0	1	0	1	0	1	0	0	0

BitXOR

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Database Table Hash Calculation

- Do alternatives exist for Multiset tables?
  -> yes, with theoretical constraints
  - > **Addition modulo bit array length**

- ADD MOD 2^160 for SHA1

1	1	1	1	1	0	1	1	1	0	1	1	1	1	0	1	0	0	1	1
1	1	1	1	1	0	1	1	1	0	1	1	1	1	0	1	0	0	1	1

Addition mod Bit Length

1	1	1	1	0	1	1	1	0	1	1	1	1	0	1	0	0	1	1	0

- This function can loose some precision.
- We use the ADD MOD 2^160 function implementation for all following tests.

Database Table Hash Calculation

- We **implement this algorithm as a C Aggregation UDF** which is getting one string as input.
  The output is a CHAR(40) hex representation of the 160 bit array.
- As stated before this function calculates the **SHA1 value per row** and **"aggregates"** these via ADD mod 2^160 to the final table hash value.
- Typical call of the function is
  table_hash(
  coalesce(col1,'NULLVALUE')!!...
  coalesce(colN,'NULLVALUE'))

Database Table Hash Calculation Example Query and Result

- The table hash for sys_calendar.calendar is calculated as:

```
select myUDFdb.table_hash (
coalesce(cast(calendar_date as char(10)),
'0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce(day_of_week, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce(day_of_month, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce(day_of_year, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce(day_of_calendar,
'0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce(weekday_of_month, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce(week_of_month, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce(week_of_year, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')

…
!! coalesce(quarter_of_year, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce(quarter_of_calendar, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce(year_of_calendar, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')  )
from sys_calendar.calendar;
```

4abd29e7ecd1d31c3004511f088becc09f5a3929

Database Table Hash Calculation Example Query and Result

- Changing a single bit

```
select myUDFdb.table_hash ( coalesce (cast(calendar_date as char(10)),
'0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce (day_of_week, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce (day_of_month, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce (day_of_year, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce (case when day_of_calendar = 10001 then 10000 else
day_of_calendar end, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce (weekday_of _month, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce (week_of _month, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
...
!! coalesce(year_of _calendar, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')  )
from sys_calendar.calendar;
```

a2de5c7ed0146fbef7333e67919e4ea8b478bc60

vs. before:

4abd29e7ecd1d31c3004511f088becc09f5a3929

Database Table Hash Calculation Example Query and Result

- **How about our 1.6 billion row example?**

```
select myUDFdb.table_hash (
coalesce (case when term_in_gutenberg = 100231237
                    then 100231236
               else term_in_gutenberg
          end, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce (file_id, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce (sentence_in_gutenberg, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce (term_in_sentence, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce (term_in_file, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce (term, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce (term_tag, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce (lemma, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce (lemma_tag, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce(is_stopword_flag, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76'),
 count(*)
from MyDataDB.gutenberg;
```

84a0e78e45735c952db19ab4bc9519f4645d9336

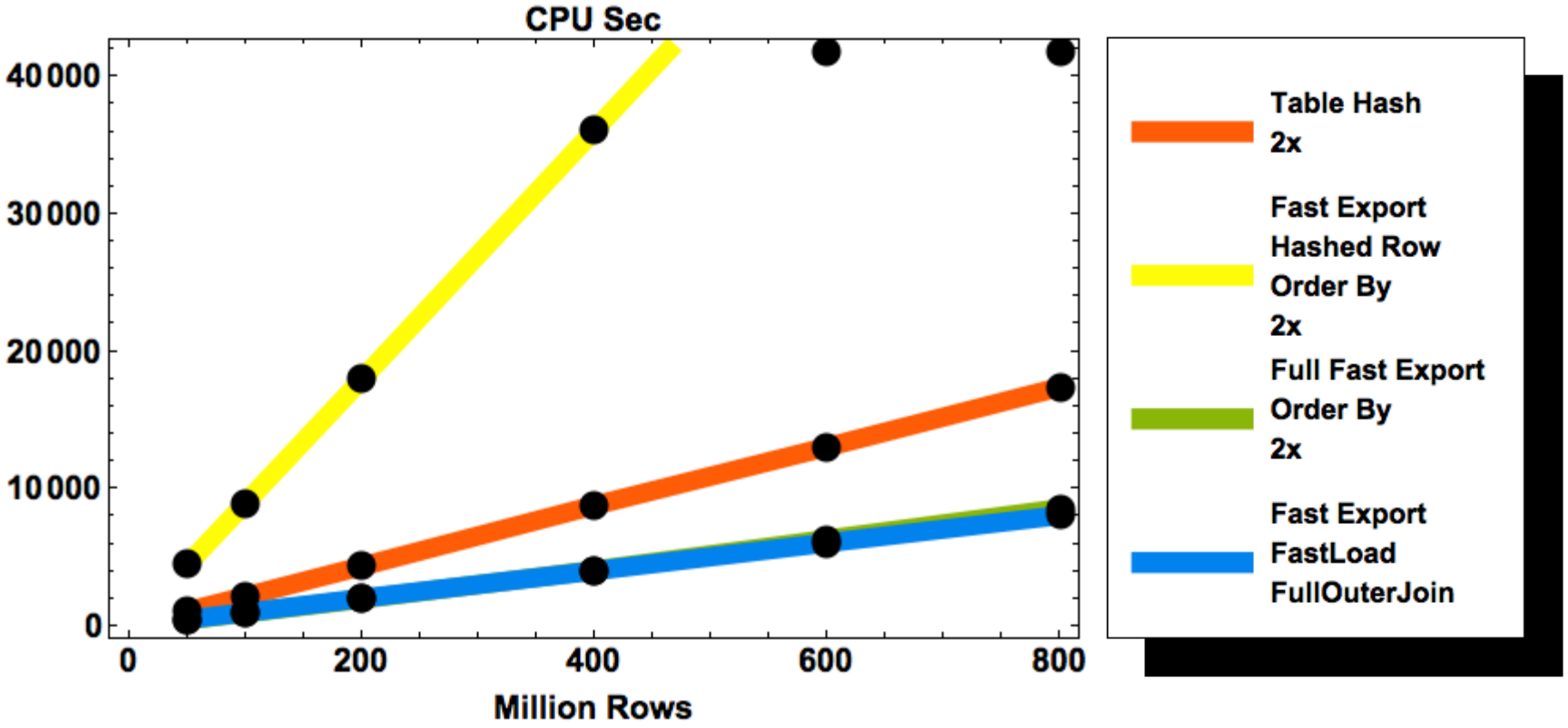
vs. before:

2abca62025dfd6df08ba4ebcbe1e050489402689

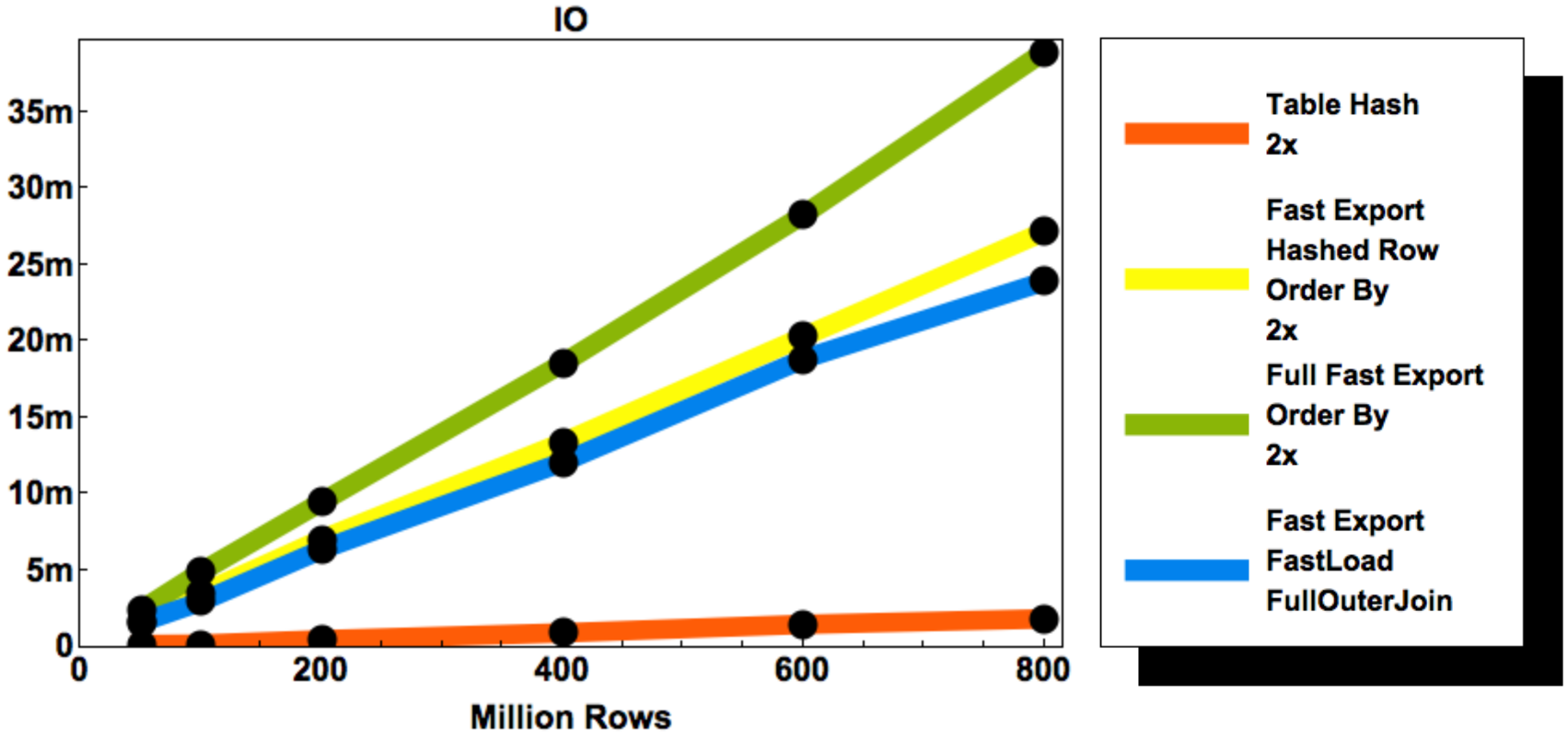
Database Table Hash Calculation

- Cost factors
  - 2 x **SQL without** ORDER BY
- Advantages:
  - **Simple and reliable**
  - **No network traffic**
- Disadvantages:
  - Only different or same indicator. In case of differences no information where the differences are or how big they are.

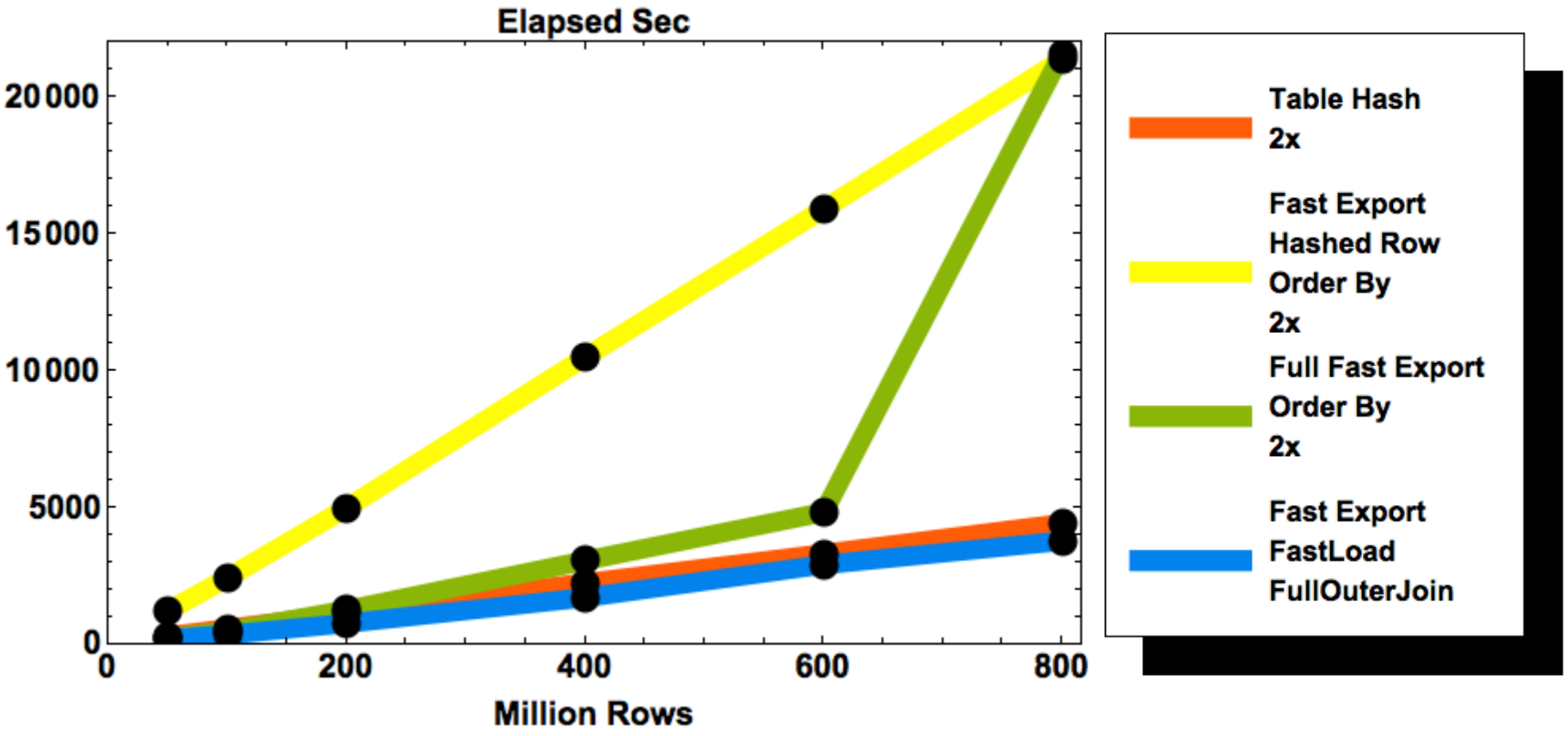
Resource Consumption
CPU



Resource Consumption
IO



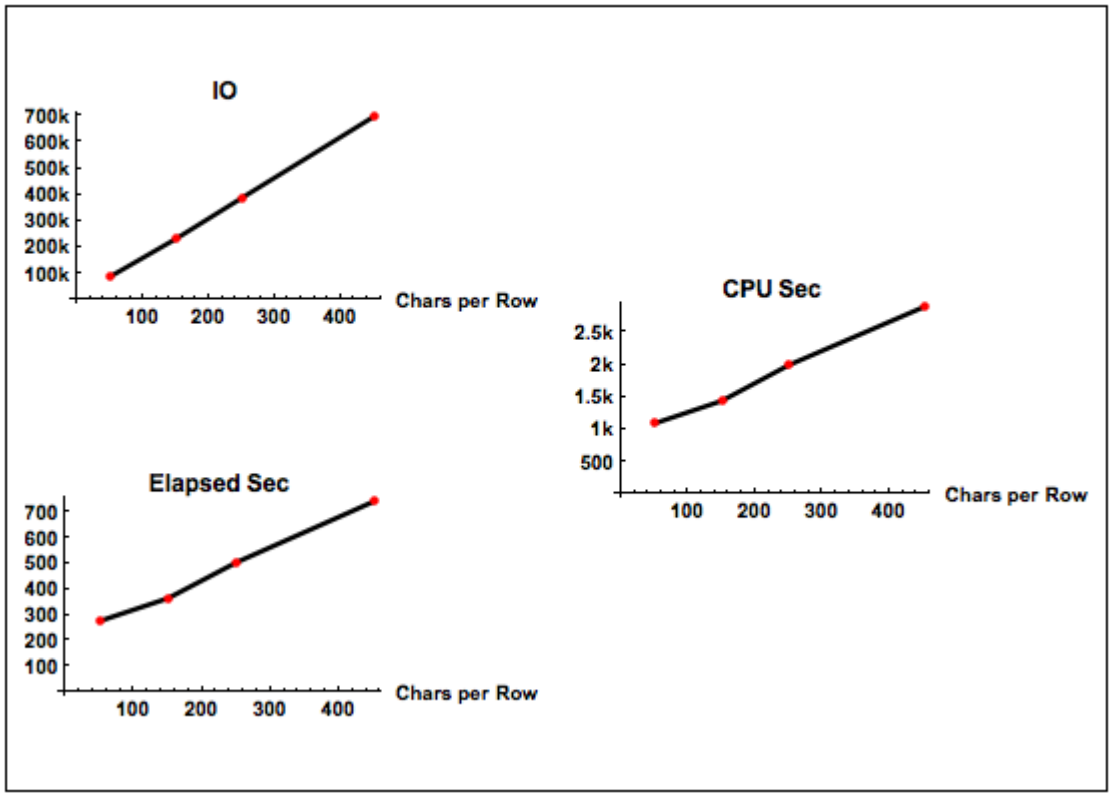
Resource Consumption
Elapsed Seconds



Resource Consumption
Table Hash Function

- The table hash function is showing linear scalability in IO, CPU and in elapsed time (in single user mode) for longer rows.

Table Hash with different row length – 100 Million rows



Agenda

- Introduction
- Classical Hash based approach
- In database table hash calculation
- **Outline / Other use cases**

Outline / Other Use Cases

- The current implementation requires **Null handling** and **TO CHAR** conversion handling. This makes the function not too easy to handle.

```
select myUDFDB.table_hash (
coalesce (term_in_gutenberg, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce (file_id, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce (sentence_in_gutenberg, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce (term_in_sentence, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce (term_in_file, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce (term, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce (term_tag, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce (lemma, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce (lemma_tag, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
!! coalesce(is_stopword_flag, '0001B3BE7E62BB4C654D28B1BB827F3F2C937D76')
 from d2k_bnc_prd.gutenberg;
```

Outline / Other Use Cases

- It would be great if Teradata would implement this function as a core function in the DB system similar to the HASHROW function.
  Main benefits would be:
  - Better Performance
  - Easier syntax as null handling and data type conversions are no longer needed.

```
select table_hash(term_in_gutenberg, file_id, sentence_in_gutenberg,
term_in_sentence, term_in_file, term,  term_tag,  lemma,  lemma_tag,
is_stopword_flag)
from d2k_bnc_prd.gutenberg;
```

Outline / Other Use Cases

- The current function requires a **complete recalculation** of the table hash if a single row/value change.

- Adding some constraint to the aggregation function it is possible to **maintain the table hash permanently during the table maintenance processes** as DB core functionality.
  This would make the table hash a permanent info of the table itself - either on the whole table, or on partition level.

Outline / Other Use Cases

- Other Use Cases are:
  - Regression testing.
    Test result is documented / validated via table hashes and not table content.
    Makes it easier/possible to automate regression tests.
    Initial test need still manual validation of the results but any test rerun should result in the same table hashes.
  - **Validation of successful application installation via compare of table hashes.** Much easier to implement.

Outline / Other Use Cases

- Source to DWH testing:
  The described **algorithm is not Teradata specific** and can be built up in any DB system which allows aggregation UDFs.
  If a **common data type transformation to CHAR** is used it is possible to do “easy” end to end test of the ETL proceeds by comparing OLTP source DB hashes with DWH table hashes - current view.

Outline / Other Use Cases

- Archive:
  If the table hash exists on partition level the history can be build up and be used to decide which partitions need to be archived.
- Restore:
  Table / Partition hash will be archived and after restore it can be validated that restore was successful.

Questions?

Ulrich Arndt

ulrich.arndt@data2knowledge.de

Give us your feedback!
To complete a Session Survey, log-in to the PARTNERSmobile app from your smart phone, go to the InfoHub kiosks, or visit teradata-partners.com.