PERFORMANCE OF INDUSTRIAL SENSOR DATA PERSISTENCE IN DATA VAULT

Florian Bachinger[(a)], Jan Zenisek[(b)], Lukas Kammerer[(c)], Martin Stimpfl[(d)], Gabriel Kronberger[(e)]

[(a),(c),(e)] Josef Ressel Centre for Symbolic Regression, School of Informatics, Communications and Media, University of Applied Sciences Upper Austria, Hagenberg
[(b)] Heuristic and Evolutionary Algorithms Laboratory, University of Applied Sciences Upper Austria, Hagenberg
[(d)] Miba AG, Dr. Mitterbauer Str. 3, Postfach 3, A-4663 Laakirchen, Austria
[(a)] florian.bachinger@fh-hagenberg.at, [(b)] jan.zenisek@fh-hagenberg.at, [(c)] lukas.kammerer@fh-hagenberg.at, [(d)] martin.stimpfl@miba.com, [(e)] gabriel.kronberger@fh-hagenberg.at

ABSTRACT

Today manufacturing companies are facing important challenges from the market in terms of flexibility, ever growing product mixes, small lot sizes, high competition, etc. To meet these market conditions, digitalization and the use of data are offering a viable toolset considering the advances in the field throughout the last couple of years.

The increasing use of sensor technology and the need for interconnecting data from different departments in smart production leads to a surge of recorded data. Persistence and integration of heterogeneous data, generated in a variety of software systems, is a key factor to gain value from data and its analysis. High flexibility in regards to the model is required to accommodate the data. Hence, application of the data vault modelling approach is a fitting candidate to design a data warehouse model. In this paper we present a data vault model for factory sensor data. We analyze the performance of the data warehouse in regards to bulk load of data and common analytic queries.

Keywords: industrial data warehouse, data vault, sensor data persistence, performance analysis

1. INTRODUCTION

Industry 4.0 and the Internet of Things (IoT) accelerate Digitalization of today's production plants. Modern production machines are now equipped with a variety of sensors measuring operational data of the machine and the product. In combination with data from material acquisition or quality assurance departments industrial companies aim to identify key input factors of the production process in a smart factory (Bauernhansl, ten Hompel, 2014). Persisting and connecting this data enables optimization of the process parameters and lays the foundation for applying new strategies such as Predictive Maintenance (PdM).

With the increasing volume and diversity of data, new challenges in data storage arise. Data is required to be stored in a structured and performant manner connected in a centralized data warehouse (DWH). The DWH enables the integration of various data sources and serves as data source for reporting, knowledge bases or PdM systems (Inmon, 2002). The data-model contained by the database management system (DBMS) must be able to handle insertion of data at a high bandwidth, while still providing fast analytic query-results based on the existing data.

Additionally, the model has to be flexible in order to easily incorporate changes in the structure of the operational data and it must stay well-defined to support query languages. The data vault modelling approach by Linstedt (2002) describes a system well suited for this scenario. Hultgren (2012) defines key indicators for determining if the application of the data vault modelling approach is a good fit.

1. Integration of multiple heterogeneous data sources
2. Accurate and complete time-slice history including audit information for full replicability
3. Frequent addition of new sources or change of existing ones
4. Commitment of the organization to incorporate an Enterprise Data Warehouse (EDW)

With all these indicators applicable, the data vault modelling approach fits the scenario at hand. In this paper, we investigate how the data vault model designed for this scenario scales and how reoccurring operations (bulk load, analytic queries) perform over increasing data volume. This should provide insight if the data vault modelling approach is a good fit for persisting sensor data of today's smart factories.

Related work in this area includes van der Veen et al., (2012) who compared sensor data storage performance of relational SQL databases with NoSQL DBMS systems in physical and virtualized environments. Collins and Shibley (2014) investigated applicability of high quality data modelling (HQDM), which define desirable properties of a DWH. They showed that the application of HQDM principles, as defined by West (2011), is well supported by the DV modelling approach.

This paper differs from previous work in that it focuses on applying the data vault modelling approach for sensor data storage and investigates the model's performance behavior over a growing data volume.

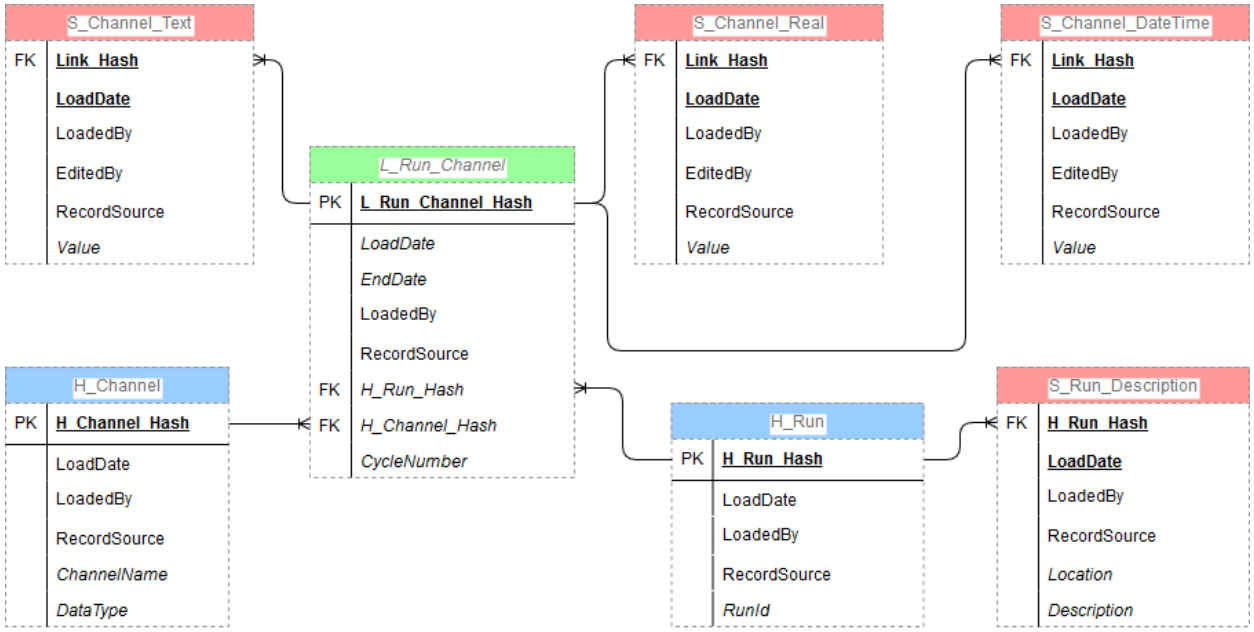

Figure 1: The devised data vault model for channel oriented storage of production machine sensor data

This paper is organized as follows. First, the data vault modelling approach is described in Section 2. Next, the data vault model used for the experiments and the kind of data to be stored in the DWH is defined in Section 3, with focus on how the data in this scenario differs from simple high resolution sensor data. Section 4 describes the performance-test setup and the results. Finally, Section 5 will draw conclusions and describe future work in this area.

2. DATA VAULT

The data vault (DV) modelling approach was defined by Linstedt (2002) and was since then further evolved into data vault 2.0 (Inmon & Linstedt, 2014; Linstedt & Olschimke, 2015). DV supports the requirements of next generation data warehouses (DWH 2.0), as defined by Inmon et al. (2010). DV differs significantly from 3rd Normal Form (3NF) model first defined by Codd (1970), which is the widest spread modelling approach. However DV is still compliant with the technical restrictions of a relational database management system (RDBMS) consisting of tables and their relationships only.

2.1. Table types

Each RDBMS table represents either a hub (H), link (L) or satellite (S) entity, which are used to store and model business concepts and their relationships. These three core components of DV are covered briefly in the next subsections. The description is based on the model depicted in Figure 1 where the different table types are also marked in different colors and with the type abbreviation as prefix.

2.1.1. Hub

Each logical business concept modelled by DV is represented by a hub entity. Besides the audit attributes (covered in Section 2.2), a hub only stores the business key used to uniquely identify a single instance of the business concept.

Sensor channels (as stored in the table *H_Channel*) for instance are identified by their channel-name and the type of data they store (e.g. the combination of name *'surface temperature'* and data type *'real-number'* identifies a specific channel). Only on first occurrence of this key-combination a new *H_Channel* hub entry is created, no duplicated hub entries are allowed. As this example illustrates, the business key can also be a composite key. Per definition of data vault 2.0 the business key is combined and subsequently reduced by a hash function like MD5 to a single identifier (128 bit sized, in the case of MD5). This calculated hash is stored as the primary key of the hub entity alongside the business key. The business key can still be used for searching by queries but for references only the calculated hash may be used.

2.1.2. Link

Link entities represent relationships between business concepts and logically connect an arbitrary number of hubs or other link entities. The link entity therefore stores the primary hash keys of all connected hub instances. This allows representation of n:m cardinality for any business relationship in data vault, effectively providing important flexibility to incorporate cardinality change in the logical model. A link's business key consists of the sum of all foreign key hash references which are subsequently combined and hashed to create the link's primary key.

Each link entry and its foreign key values represent an existing relationship between logical business concepts. To represent invalid relationships, the link entity stores an end-date to mark from when on the relationship is no longer existent. This is necessary since deletion of data, in any table type, is not permitted.

As depicted in Figure 1 for instance an entry in *L_Run_Channel* represents the assignment of a sensor channel value to a specific run instance. The run and channel hashed business keys are stored as foreign keys.

2.1.3. Satellite

Whereas hubs store only the key of a business concept, the connected satellites store all associated descriptive attributes. In the case of a sensor installed in a smart factory plant, the sensor satellite might store the location and a description. Each satellite stores the foreign key reference to the hub or link instance and the specific date and time when the current state of attributes was loaded, making up the primary key of a satellite. Consequently, each entry of a satellite is a time slice representing the states of the business concept stored in the hub and its connected satellites, with the latest entry representing the current state. Following the DV modelling approach the entire attributes associated with a hub entity are not necessarily stored in one single satellite table. Hultgren (2012) encourages to separate satellites by:

1. Subject Area / Data Context
2. Rate of Change
3. Source System
4. Type of Data

This means that new satellites are created to store additional attributes that are defined for a business entity instead of adding new columns to existing tables. Separating satellites by the above criteria reduces the amount of sparse values and duplicated data on insertion of a new time slice and therefore reduce the total memory space.
In the case of the depicted model of Figure 1, the satellites storing the sensor values are separated according to the last 3 separation indicators.

2.2. Historical data and auditing

Generally speaking the DV allows only additive changes to the model/schema and the data, to provide historically correct and complete data views, with the only exception to that rule being the link's *EndDate* column which is updated once a relationship is no longer valid. Each change in the source data is persisted by insertion of a new satellite data slice.
Apart from the business data the DV entities store audit fields. The attributes *LoadedBy*, *RecordSource* and *LoadDate* provide information about the data source system and the import service. Whereas *EditedBy* and the *LoadDate* provide information about the user, who conducted the data change.

2.3. Unique features of the data vault approach

The utilization of hashed business keys as primary key holds two main benefits in this scenario. First, it improves join performance of analytical queries. Key comparison is faster for a single primary key of fixed data type and consistent length when compared to the join performance of longer natural keys or utilization of composite keys (Linstedt, 2014). Second, the hash value of the business key can be calculated at load time of the data. This improves performance of bulk inserts, since hubs and satellites can be loaded parallel after foreign key constraints are deactivated for the load process or not present at all. In contrast, utilization of a *bigint* with auto increment also creates keys of equal lengths, but the hub entry has to be created beforehand in order to load the satellites.

3. MODEL AND DATA QUANTITY

This section aims to provide a clearer picture of the kind of data to be stored in the DV model. We describe a more complex scenario where the data differs from a simple log of recorded sensor data. A description of the origin of the data will highlight the need for storing audit information. Subsequently we describe why we chose to model the data as shown in Figure 1 and how we improved the performance of the model.

3.1. Data origin

Typically, a sensor records data in high frequency, resulting in a high volume of, mostly monotonous, data. This kind of raw sensor data is best stored as a compressed binary file or in a database table without the additional overhead of audit attributes, in order to handle the huge volume of data. However, in the scenario described in this paper, the high resolution sensor data is at first analyzed by domain experts and subsequently filtered and aggregated into lower resolution. Figure 2 illustrates this well-established process. The aggregated and filtered sensor data is read from the file share and imported in the DWH. Data analysts might even create several iterations of analyzed data by applying different thresholds and aggregation functions to the raw source data, thus requiring a DWH to store audit information for each data slice. In addition to the sensor data recorded at each run of the machine (e.g. actual oven temperature) the DWH also stores the production parameters set by the operator (e.g. target oven temperature) to provide a full representation of the production process.

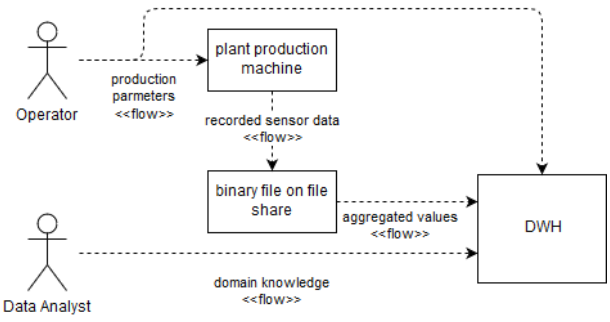

Figure 2: Schematic representation of scenario data flow

By combining parameters and recorded sensor values the DWH is able to provide a full picture of all relevant production data. The data therefore serves as a comprehensive source for analytical queries or PdM systems.

3.2. Data volume analysis

The performance of the devised model will be analyzed in regards to the data volume expected after up to 5 years of usage. As Figure 2 shows, each production machine creates a binary file that is stored on a file share. One file is created for each single run of the production machine. For analysis however only the average state (sensor values) of the machine at specific time slots is relevant. The machine runs can vary in duration but consist of up to 3,700 individual time slots at maximum run duration. Exactly one value for each installed sensor is assigned to the individual time slot. With up to 24 different sensors per machine, this results in 88,800 individual values per run. It is expected that the operator configures one daily run per machine, which would result in up to 780 per year, since this feature will be rolled out to 3 machines of the same type.

For the analysis we investigate the performance of key operations for every 500 runs, up to a total of 4000 runs saved in the DWH.

3.3. Description of the DV model

To accommodate the heterogeneous data described in Section 3.1 the devised DV model has to be generic and flexible. Sensor values are therefore not persisted in column orientation, with one table column storing a sensor value at any given point of time (as illustrated in Figure 3), but are instead stored in a channel-oriented manner. This allows easy addition, relocation or upgrade of sensors to the production machines without the need to change the DWH model or the importer process.

3.3.1. Column-oriented storage of sensor data

In column-oriented storage, each sensor channel corresponds to a single column in the data table. This storage variant of sensor data would be more efficient than the devised model, if the attributes of the table, respectively the different applied sensors themselves, are well known and change little over time. The frequent addition of new sensor channels or the removal / relocation of existing ones leads to fragmentation of the table and consequently to weaker performance and higher storage consumption in this model approach. Additionally, the importing tool would need frequent updates in order to accommodate the change in model structure.

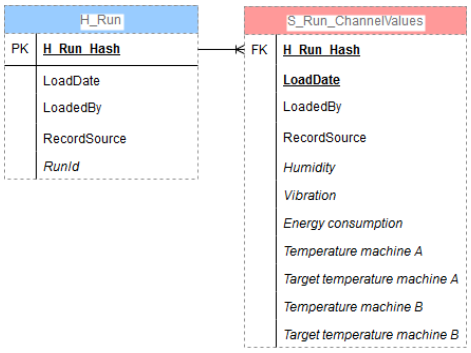

Figure 3: Alternative column-oriented storage of channel values

3.3.2. The devised channel-oriented model

Instead, a channel-oriented model approach was used, as depicted in Figure 1, wherein each run is assigned a number of different sensor channels. A single channel is identified by its name (e.g. 'Temperature Machine A') and the type of data stored in this channel (numeric, text or datetime values). Sensor channel values are assigned to a run by creating an entry in the link entity. The time slot dependent order of the sensor values is maintained by the *'CycleNumber'* attribute of each link entry. Each cycle represents a point of time defined by the data analysts (see Figure 2). All values assigned to a specific cycle represent the aggregated values, recorded in different frequencies by the sensor, of one pre-defined time interval. To provide type safe access and to reduce storage space each datatype (text, numeric, and timestamp) is stored in a separate satellite respectively, as opposed to one large binary column.

The data stored in this generic storage schema is subsequently transformed periodically into the, easily humanly interpretable, column-oriented storage form for on-line analytical processing (OLAP). Transformed data is stored as data mart, which form a subset of total data persisted (Chaudhuri & Dayal, 1997). The transformation operation is defined in Section 4 as one of the analyzed DBMS queries.

3.4. Applied performance improvements to the model

The performance improvements discussed in this section are in many cases specific to the software used in the experiments and production environment. Some improvements to ETL and Query performance however are generally applicable to any RDBMS. E.g., relationships between the hubs, links, and satellites were not enforced by foreign keys, reducing load time of the RDBMS. Instead, the importer tool is expected to take care of these restrictions. Besides enforcing the foreign key references, the importer tool caches the primary keys of entries. This increases error resilience of the importer tool for already seen values, since the unique constraints enforced by the DWH are not violated.

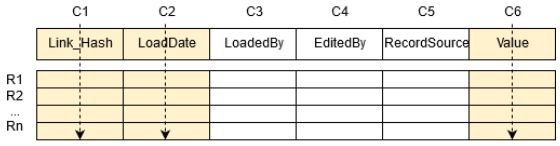

Figure 4: With columnar storage only the relevant columns have to be retrieved at query time (compare to Kamal & Gupta (2015))

3.4.1. MSSQL-Server specific improvements

In order to keep the disk size of the database manageable a clustered columnar store index was applied to the DV tables. This feature was introduced with MSSQL server 2012 Enterprise. Instead of a traditional row storage, the values are stored in a columnar storage (see Figure 4 and Figure 5).

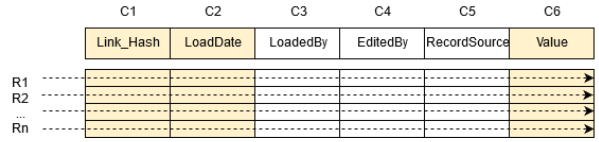

Figure 5: With standard row storage the complete row has to be retrieved (compare to Kamal & Gupta (2015))

Column storage allows compression of stored values. In their paper Kamal & Gupta (2015) showed that columnar storage not only reduced the page sizes of the DB but also improved performance in an OLAP environment.
Besides the clustered column store, the unique indices for primary keys were also compressed, which significantly reduced the database size.

3.4.2. ETL tool improvements

As it is typical for DWH data integration, the data is first *Extracted* then *Transformed* and subsequently *Loaded* (ETL) into the DWH. All three steps of this process are performed by a custom importer implemented in C# and running on the .NET Core Framework. A custom parser reads the proprietary binary format containing the preprocessed sensor data. The parsed data is then transformed to fit the DWH schema and loaded into the DB using entity framework (EF).
Memory consumption of the importer tool was reduced significantly by creating a new EF DbContext for every single imported binary file. This allowed the .NET garbage collector to remove the inserted entities and the EF object graph.
A first version of the importer tool kept a set of all hash keys to enforce the unique constraint during ETL, which quickly exceeded memory limits for larger data volumes. To reduce memory consumption of the tool, only hashes associated with the current binary source file were kept in memory.

4. PERFORMANCE ANALYSIS

Typically, a simple high frequency sensor data store must support regular small writes and occasional large read bursts, as described by van der Veen et al. (2012), whereas in this scenario both reads and writes are only occasional and in large volume.
For this performance analysis though the source data was not read from a binary file but was instead generated to simulate the estimated data volume the next 1-5 years of production usage. Existing sensor data was analyzed beforehand, to ensure that the generator samples from an equally distributed value range, which holds representative values for each individual sensors. For each of the 5 years the following performance analysis was made:

a) Average time elapsed on a single ETL file import.
b) Time elapsed for total index rebuild.
c) Execution time of transformation query for the single imported run.
d) Execution time of transformation queries for total data volume for data-mart creation.
e) Sum of used storage space for the DV data.
f) Sum of storage size for compressed indices in the DWH.
g) Total size of the DB.

The time measured for (a) of a single ETL file import task is comprised of time spent on generating values (comparable to time spent reading the binary file) and the transactional write to the DWH. In the best case, this metric changes little over increasing DWH size. This would indicate that the DWH is able to handle bulk inserts in short time over all years. Due to increasing index size however the import duration will increase over total DWH volume.
After successful execution of the ETL process the fragmentation of indices was checked. Following recommendations by Microsoft documentation, if critical fragmentation is reached (Microsoft, 2017), a stored procedure executes statements as listed in Algorithm 1 for every table affected by the ETL process. The index is rebuilt with data compression to reduce DB size needed for indices to a minimum. The elapsed time for execution is represented by measure (b).

Algorithm 1: snippet for rebuild of compressed index

```
ALTER INDEX dv_L_Run_Channel_unique_hash
        ON dv.L_Run_Channel
    REBUILD PARTITION = ALL WITH (DATA_COMPRESSION = PAGE)
```

Measure (c) shows the elapsed execution time of pivot transformation of the data of the single imported run. The data is transformed from full historical channel oriented storage of sensor data (as described in Section 3.3.2) to column-oriented storage for OLAP, showing only latest state of values. The transformed values are then added to the transformed data mart table *(dm.RunData)*. The transformation query, as part of listing Algorithm 2, is prepared as a view and queried after every finished ETL process (to improve readability of the code snippet the view is included as the actual select query). In order to calculate the measure (c) the view is filtered by *RunId*. If however the structure of the data mart table has to change, the complete script of Algorithm 2 is executed. The elapsed time of this script is indicated by measure (d).

Algorithm 2: rebuild and pivot transformation of channel oriented data into table oriented representation of data mart

```
BEGIN
 DROP TABLE dm.RunData

 SELECT * INTO dm.RunData
 FROM (SELECT c.[Name] AS ChannelName
    , p.RunNr
    , l.CycleNumber
    ,  d.[value] AS DataValue
  FROM dv.L_Run_Channel l
  JOIN dv.H_Channel c
   ON l.H_Channel_Hash = c.H_Channel_Hash
  JOIN dv.H_Run p
   ON l.H_Run_Hash = p.H_Run_Hash
  LEFT JOIN dv.S_Channel_Double d
   ON d.L_Run_Channel_Hash = l.L_Run_Channel_Hash
   AND c.DataType = 'Double'
   AND d.LoadDate = (SELECT MAX(LoadDate)
```

```
        FROM dv.S_Channel_Double sub
        WHERE sub.L_Run_Channel_Hash =
          l.L_Run_Channel_Hash)) p
  PIVOT
  (
   MIN(DataValue)
    FOR ChannelName IN (
          <comma separated list of relevant channel names>
         )
  ) AS PivotTable;
 CREATE NONCLUSTERED INDEX dm_RunNR ON dm.RunData(RunNr)
END
```

Measures (e) and (f) where calculated by executing the '`exec sp_spaceused`' command, which provides those exact measures, whereas (g) is simply the sum of (e) and (f).

4.1. Test Setup

The model was implemented on a Microsoft SQL Server 2017 running on a Windows 10 machine with an Intel i5 7300U @ 2,6GHz, 32GB RAM and a 512GB SSD. Both the DB and the importer tool ran on the same computer simultaneously. To measure the time elapsed during DB operations or queries the MSSQL statistics time was turned on. Queries were executed using the MSSQL server management tool.

4.2. Test Results and Discussion

The main goal of the conducted experiments was to investigate if the presented model for sensor storage is able to handle the load of several years in production usage. To investigate the behavior both elapsed time of regular operations and storage consumption was observed.

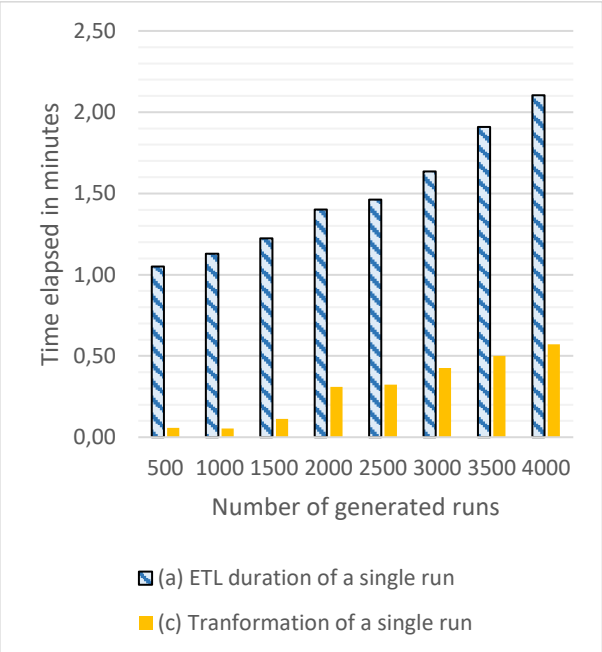

Figure 6: comparison of operation timings of regular measures (a) and (c) over number of runs

Measures (a) and (c) indicate the timings of regular operations of the DWH. A single experiment is expected to be inserted every day, and subsequently has to be transformed into the data mart table. The timings as depicted in Figure **6** show that these regular operations can be performed in reasonable time, with less than 3 minutes elapsed after the maximum of 4000 imported runs. The import of a new run into the DV and the addition of transformed run data into the data mart causes fragmentation of the applied indices. In production environment, indices will be monitored and if the fragmentation exceeds 30% the indices will be rebuild.

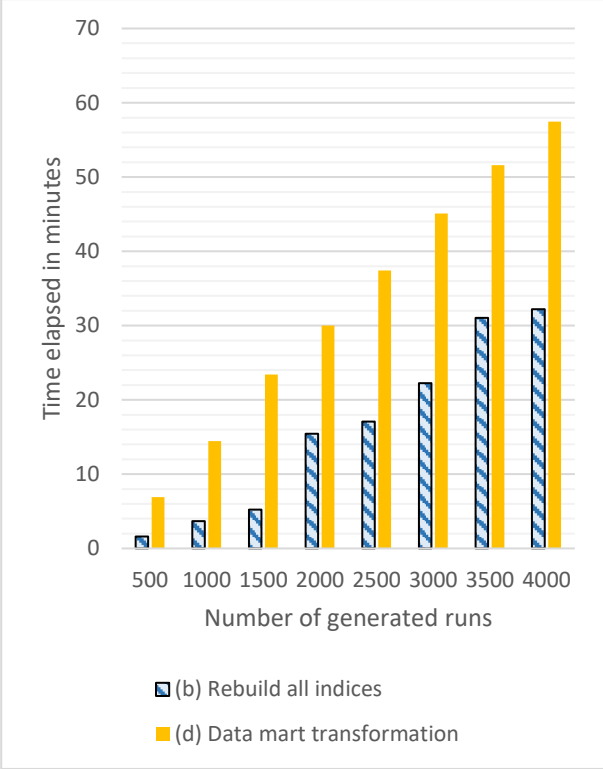

Figure 7: comparison of operation timings of irregular measures (b) and (d) over number of runs

In Figure 7 the timings of measures (b) and (d) are presented. These measures represent DWH operations which occur only irregularly, but require longer runtimes. For high amounts of data (355.320.000 single values in value satellites and the link table, at 4000 runs stored in the DB) the operations are still manageable from the perspective of elapsed time. Especially, since the experiments were conducted on standard, off the shelf, hardware. Index rebuild (b) and data mart transformation (d), executed on all data stored, were both observed to have spiked CPU usage and RAM consumption to 100% for the total duration of the operation. As observed during the experiments, a total rebuild of indices (b) after reaching >30% fragmentation of the unique indices will be needed after roughly every ten imported runs.

As Figure **8** shows, the DB size scales linearly with the number of stored runs. It also shows very nicely the effect of column store compression from 500 to 1.000 where the index size increased but data itself could be compressed more efficiently. With 53.58GB of total DB size at 4.000 imported runs, the DB remains manageable.

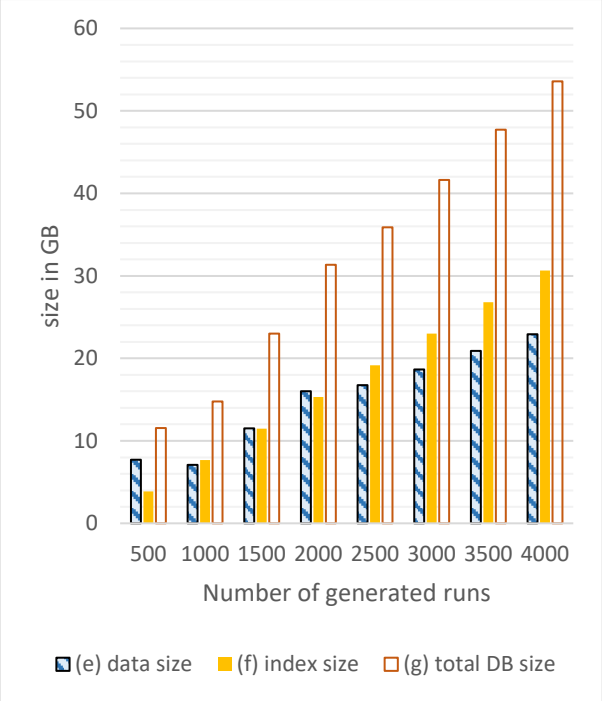

Figure 8: comparison of data size (e), index size (f) and total DB size (g) over number of runs

Besides the total DB size, the positive effects of the applied compression techniques can be very nicely observed in Figure 9. Where the disk space required for storage of one singe run is shown to decline to as low as 13.71MB, whereas the proprietary binary format used to store preprocessed runs on the file share, as discussed in Section 3.1, requires roughly 34MB of storage. These files however store only the actual sensor values and no overhead in form of additional audit or info attributes.

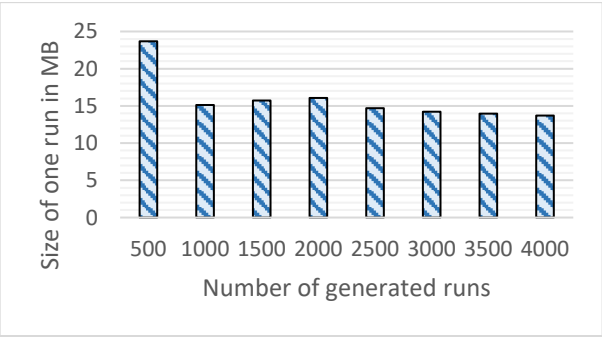

Figure 9: DB size needed for data of one single run

5. CONCLUSIONS

The DV modelling approach provides a high degree of flexibility in regards to the schema. One first has to get accustomed to the concept of hubs, links and satellites though. The inherit flexibility of this concept proved to be beneficial during the development of the schema and the development of the importer tool. Additional attributes in need of persistence were easily added as new satellites.

Moreover, the utilization of calculated hash values for primary keys instead of DB generated sequence IDs proved very helpful for ETL, manual correction of the data and also for tuning queries (e.g. calculating hash value of a RunId on client side frontend instead of using the business key in the query). Especially when the business keys consist of strings of variable length the hash values even provide performance improvements. This technique is not limited to the DV modelling approach and can only be endorsed.

The proposed channel oriented model provides a generic storage for industrial sensor values, resulting in low to no maintenance effort needed when additional sensor channels get introduced in the source files. The model was observed to be able to handle the expected load of 5 years of production usage.

However, the amount of single rows in the value satellites and the link table *L_Run_Channel* might present a performance bottleneck in the future. One possible enhancement is to scale the DWH horizontally by separating the stored runs of the three different machines onto three different DWH instances.

Another solution to counter the potential performance bottleneck of the value satellites would be extract all types of sensors shared by the different machines into a separate satellite (e.g. all machines measure energy consumption, temperature). Effectively storing all common channels and their values in a column-oriented way but keeping the uncommon channels stored in the devised generic model.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development within the Josef Ressel Centre for Symbolic Regression.

Jan Zenisek gratefully acknowledges financial support within the project “Smart Factory Lab” which is funded by the European Fund for Regional Development (EFRE) and the state of Upper Austria as part of the program “Investing in Growth and Jobs 2014-2020”.

REFERENCES

Chaudhuri, S., & Dayal, U. (1997). An overview of data warehousing and OLAP technology. ACM SIGMOD Record, 26(1), 65–74. https://doi.org/10.1145/248603.248616

Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. Communications of the ACM, 13(6), 377–387. https://doi.org/10.1145/362384.362685

Collins, G., & Shibley, M. (2014). Data Vault and HQDM Principles. SAIS 2014 Proceedings.

Hultgren, H. (2012). Modeling the agile data warehouse with data vault. New Hamilton.

Inmon, W. H. (2002). Building the data warehouse. John

Wiley & Sons, Inc.
Inmon, W. H., & Linstedt, D. (2014). Data Architecture: A Primer for the Data Scientist: Big Data, Data Warehouse and Data Vault. Morgan Kaufmann.
Inmon, W. H., Strauss, D., & Neushloss, G. (2010). DW 2.0: The Architecture for the Next Generation of Data Warehousing. Morgan Kaufmann.
Kamal, A., & Gupta, S. C. (2015). Query based performance analysis of row and column storage data warehouse. 9th International Conference on Industrial and Information Systems, ICIIS 2014. https://doi.org/10.1109/ICIINFS.2014.7036537
Linstedt, D. (2002). Data Vault Series 1 – Data Vault Overview. Retrieved July 10, 2018, from http://tdan.com/data-vault-series-1-data-vault-overview/5054
Linstedt, D. (2014). datavault 2.0 hashes versus natural keys. Retrieved May 7, 2018, from http://danlinstedt.com/allposts/datavaultcat/datavault-2-0-hashes-versus-natural-keys/
Linstedt, D., & Olschimke, M. (2015). Building a Scalable Data Warehouse with Data Vault 2.0. Morgan Kaufmann. https://doi.org/10.1016/C2014-0-02486-0
Microsoft. (2017). Reorganize and Rebuild Indexes. Retrieved July 10, 2018, from https://docs.microsoft.com/en-us/sql/relational-databases/indexes/reorganize-and-rebuild-indexes?view=sql-server-2017
Thomas Bauernhansl, Michael ten Hompel, B. V.-H. (2014). Industrie 4.0 in Produktion, Automatisierung und Logistik. (T. Bauernhansl, M. ten Hompel, & B. Vogel-Heuser, Eds.). Wiesbaden: Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-04682-8
van der Veen, J. S., van der Waaij, B., & Meijer, R. J. (2012). Sensor Data Storage Performance: SQL or NoSQL, Physical or Virtual. 2012 IEEE Fifth International Conference on Cloud Computing, 431–438. https://doi.org/10.1109/CLOUD.2012.18
West, M. (2011). Developing High Quality Data Models. Morgan Kaufmann.