Continuous Information Processing Addressing Cisco's Pain Points by Enabling Real-Time Ad-Hoc Reporting Capability: An Energy Efficient Big Data Approach

Martin Zinner[*], Wolfgang E. Nagel[*]
[*] Center for Information Services and High Performance Computing (ZIH)
Technische Universität Dresden
Dresden, Germany
E-mail: {martin.zinner1,wolfgang.nagel}@tu-dresden.de

***Abstract*—Aggregation is a special type of association, in which objects are assembled/composed together to create a more complex object. Unfortunately, a considerable part of data aggregation during information processing in industry is still carried out in nightly batch mode, taking into account all its negative side effects. In particular, before starting aggregation, all data required for computation must be available and accessible. In contrast, our method termed Continuous Information Processing Methodology (CIPM), does not assume that the data to be aggregated is fully retrieved, the aggregation can be started as soon as the data collection is initiated. During the data collection process, partial aggregated values are calculated, such that, after the data collection phase has been completed, the final aggregated values are available for real-time ad-hoc evaluation. The existing aggregation methods apply only the usual aggregation functions such as sum(), avg(), max(), min(), count() as build in functions, on the contrary, the most common aggregation functions used in various field of industry and business can be easily adapted and used within CIPM. The major benefit of our method is the elimination of the extra time necessary for batch computation, as well as reduced and spread aggregation effort over the whole collection period and tightened and straightforward computational design strategies. To conclude, the CIPM supports a paradigm shift from more or less subjectively designed individualistic conceptions in software design and development towards objectively established optimal solutions.**

Index Terms*—Continuous information processing; Continuous aggregation; Energy efficient computation; Real-time capability; Real-time capabilities; Data Analytics; Data processing; Stream processing; Batch processing; Business Intelligence; Ad-hoc reporting; Big Data.

I. Introduction

Initially, within this section, the core of our aggregation theory is succinctly addressed, some definitions such as that of Big Data are tightened up and subsequently, the principal motivation of our paper, i.e., increased real-time requirements in the industry, is presented. Aggregation is a special type of association, in which objects are in general assembled/composed together to create a new and a more complex object. The aim of aggregating data is to create new knowledge and to hide useless information. For example, by aggregating data delivered in millisecond to cycles of minutes and by calculating the sum(), avg(), max(), min(), count(), and other statistical functions like the standard deviation, new information regarding the smoothness of the data delivery or their homogeneity, can be generated. In the mean time, useless information such as the initial information tracked in milliseconds cannot be any more referenced. Usually, data from multiple data sources and different domains are aggregated, for example equipment and production-line data can be combined in order to deliver useful information to engineers regarding production bottlenecks, but also reports for the upper management. Commonly, in the industry, the aggregation is performed on daily bases, using batch jobs. Usually, the batch jobs are started at night, after all data has been collected.

Cisco Systems, Inc. identified the deficiencies of the classical nightly batch jobs aggregation strategy, disclosed and summarised them within five Pain Points in the White Paper "*BI and ETL Processes Management Pain Points; Understanding the most pressing pain points and strategies for addressing them*". All Pain Points, except for Pain Point 3 regarding ad-hoc reporting, have been addressed in a conference paper [1]. Within this paper, the theoretical background presented in [1] is extended, such that it equally covers the ad-hoc reporting functionality. Ad-hoc reporting is a *Business Intelligence (BI)* process used to quickly create reports on an as-needed basis. Ad-hoc reports are generally created for one-time use to find the answer to a specific business question and offers a wealth of values to business across industries. Ad-hoc reports assures the required flexibility to be able to follow and adapt to the continually changing business environment. Furthermore, ad-hoc evaluation strategies enable the users to autonomously create new reports without involving highly qualified IT personnel.

In order to illustrate our methodology, we will present a typical adjustment regarding the statistical function *Standard Deviation (SD)*, such that it can be used within the continuous aggregation strategy. The standard deviation is simple enough to gain a good overview concerning the difficulties that arise in practice, but it is not trivial and exemplifies the immanent problem of the continuous information processing methodology. The usual representation of the standard deviation is adapted to fit our needs.

Aggregation is an operation to obtain summarised information by using aggregate functions. A new approach for information aggregation based on a very simple and straightforward starting point is formulated in this paper – namely, that within the information flow, *the process of information aggregation should be started as early as possible, best as* *soon as the collection phase is initiated*. This strategy assumes a strict and clearly defined architectural design strategy of the computational framework and enables real-time capability of the system, therefore, the new methodology is termed *Continuous Information Processing Methodology* (CIPM). In order to be able to in-depth analyse the CIPM, a formal, mathematical model is set up, the conversion of the underlying structure is defined and the pros and cons of CIPM, as opposed to the classical batch jobs strategy, are discussed.

As defined in [2] "Big Data is the information asset characterised by such a *high volume*, *velocity* and *variety* to require *specific technology* and *analytical methods* for its transformation into value". According to the definition above, Big Data is much more that high volume of data and needs unconventional methods to be processed.

At the same time, a cultural change should accompany the process of investing in interdisciplinary Business Intelligence and *Data Analytics* education [2], involving the company's entire population, its members to "efficiently manage data properly and incorporate them into decision making processes" [3].

A. Motivation

1) Rapidly increasing data amount: The total amount of data created, captured, and consumed globally is forecast to increase rapidly, reaching more than 180 Zettabytes in 2025, as opposed to 64.2 Zettabytes in 2020 and 15.5 Zettabytes in 2015 [4]. Real-time information processing has become a significant requirement for the optimal functioning of the manufacturing plants [5]. Worldwide by 2022, over 50 billion *Internet of Things* (IoT) devices including sensors and actuators are predicted to be installed in machines, vehicles, buildings, and environments and/or used by humans.

2) Real-time requirements: Demand is also huge for the real-time utilisation of data streams, instead of the current batch analysis of stored Big Data [6]. The operations of a real-time system are subject to *time constraints* (deadlines), i.e., if specified timing requirements are not met, the corresponding operation is degraded and/or the quality of service may suffer and it can lead even to system failure [7]. In a real-time system deadlines must always be met, regardless of the system load. Usually, a system not specified as real-time cannot guarantee a response within any time frame. There are no general restrictions regarding the magnitude of the values of the time constraints. The time constraints do not need to be within seconds or milliseconds, as often they are understood. There is a general tendency that real-time requirements are becoming crucial requisites.

Travellers require current flight schedules on their portable devices to be able to select and book flights; in order to avoid overbooking, the flight plans and the filled seats must be kept reasonably current. Similarly, people expect instant access to their business-critical data in order to make informed decisions. Moreover, they may require up-to-date aggregated data or even ad-hoc requests. This instant access to critical information may be crucial for the competitiveness of the company [8].

B. Aim

Cisco [9] identified a couple of *Pain Points* in the Business Intelligence (BI) area, but these Pain Points carry a more general validity:

1) *the race against time*; managing batch window time constraints,
2) *cascading errors and painful recovery*; eliminating errors caused by improper job sequencing,
3) *ad hoc reporting*; managing unplanned reports in a plan-based environment,
4) *service-level consistency*; managing service-level agreements,
5) *resources*; ETL resource conflict management.

Our approach is addressing all five Pain Points.

In conclusion, continuous information processing enables a new perspective on aggregation strategies, such that aggregation is performed in parallel to the data collection phase. Preliminary aggregated values corresponding to the current state of the retrieved data are available for ad-hoc evaluation, nightly batch aggregation becomes obsolete.

C. Outline

The remainder of the paper is structured as follows: section II gives an overview regarding existing work related to the described problem. An informal presentation of the continuous aggregation strategy is presented in section III, whereby section IV introduces the mathematical model and describes the methodology to transform the batch aggregation into continuous aggregation. The presentation of the main results and discussions based upon these results constitute the content of section V, whereas section VI summarises our contributions and draws some perspectives for future work.

II. Related work

Primarily, the focus of this section is on algorithmic approaches regarding the state of the art. The analysis of different one-pass algorithms [10] and their efficient implementation is beyond the scope of this paper as well as pure technical solutions based on database technologies.

A. SB-trees

A B-tree is a balanced tree data structure, that keeps data sorted and allows searches, sequential access, and deletions in logarithmic time; the tree depth is equal at every position, whereas the SB-tree is a variant of a B-tree such that it offers high-performance sequential disk access [11], [12]. Zhang [12] outlines the key challenges of spatio-temporal aggregate computation on geo-spatial image data, focusing primarily on data having the form of raster images. Zhang gives a very detailed overview of the state of the art regarding efficient aggregate computation. Zhang's approach is based on *aggregate queries* common in the database community, including data cubes, whereas our approach (CIPM) does not focus on database technology when calculating the aggregation functions. For example, the improved multi-version SB-tree [12] consumes more space than the size of raw data. Other approaches use only a small index, reducing the space needed, but supporting only count and sum aggregate functions. The main idea behind the SB-trees is to provide through a depth-first search, – by accumulating partial aggregate values – a fast look-up of computed values [12], [13].

B. Scotty

Scotty [14] is an efficient and general open-source operator for sliding-window aggregation for stream processing systems, such as Apache Flink, Apache Beam, Apache Samza, Apache Kafka, Apache Spark, and Apache Storm. It enables stream slicing, pre-aggregation, and aggregate sharing including out-of-order data streams and session windows [15]. The aggregate window functions are: avg(), count(), max(), min(), sum(). Being a toolkit, the out-of-the-box aggregate functions are restricted to the above. Implementation details are disclosed in a preprint paper [16]. Scotty can be extended with user-defined aggregation functions, however, these functions must be associative and invertible. Since Scotty is open source, additional user extensions are always possible.

Sliding window aggregation is also a main topic regarding this paper, even if sliding windows are not used for reporting/evaluation. There is always the possibility that erroneous data is captured and cannot be corrected automatically. This cannot be avoided, since a data set may look formally correct, but may be wrong with regard to its content. Such anomalies can be detected hours after the data has been processed and should be corrected.

According to [17] research on sliding-window aggregation has focused mainly on aggregation functions that are associative and on FIFO windows. Much less is known for other nontrivial scenarios. The question arises, whether is it possible to efficiently support associative aggregation functions on windows that are non-FIFO? Besides associativity and invertibility, what other properties can be exploited to develop general purpose algorithms for fast sliding-window aggregation? Tangwongsan et al. [18] present the Finger B-tree Aggregator (FiBA), a novel real-time sliding window aggregation algorithm that optimally handles streams of varying degrees of out-of-orderness. The basic algorithms can be implemented on any balanced tree, for example on B-trees.

C. Holistic functions

The median, which is the middle number in an ordered list of items, is a holistic function, i.e., its results have to rely on the entire input set, so that there is no constant bound on the size of the storage needed for the computation. An algorithm suitable for continuous aggregation based on heap technology can be found in [8]. For the sake of completeness, the main idea is presented hereafter. In order to store the data, two heaps are used, one for the higher part and one for the lower part of the data. The newly collected dataset is inserted into the corresponding heap; if the case arises, the heaps are balanced against each other, such that the two heaps contain the same number of items, etc. Hence, in some cases, holistic aggregation functions can be used with continuous aggregation techniques, they should however satisfy the foreseen time constraints.

Unfortunately, even such common functions as min() or max() have a holistic behaviour in some circumstances. If used for example in a sliding-window aggregation environment or if retrospectively data corrections are allowed, then all of the values have to be stored in order to determine the minimal or the maximal value of the stream. Similarly, a heap of sorted values can be used in order to implement real-time capability.

D. Quantile

A survey of approximate quantile computation on large-scale data is given by Chen [19]. In streaming models, where data elements arrive one by one, algorithms are required to answer quantile queries with only one-pass scan. Formulas for the computation of higher-order central moments or for robust, parallel computation of arbitrary order of statistical moments can be found here [20], [21], some of them are one-pass incremental approaches.

In conclusion, the main focus of the existing research has been to develop aggregate queries for efficient retrieval and visualisation of persisted data. However, with Scotty a general open-source operator for sliding-window aggregation in stream processing systems, – such as, for example, the Apache family – has been developed. Scotty incorporates the usual aggregate functions like avg(), sum(), etc., and it has the possibility to include special user defined functions. Tangwongsan [17] points out that much less is known for nontrivial scenarios, i.e., functions that are not associative and do not support FIFO windows. Our approach, however develops the strategy and technology for continuous information processing, abbreviated CIPM and shows that functions, which allow efficient one-pass implementations are suitable for CIPM. Moreover, holistic functions allowing appropriate implementation, for example median [8], can be used with CIPM.

III. Problem Description

The technical terms “information function” and “aggregation function” [22] are used synonymously within this paper. They highlight the same topic from different perspectives. Corporate reporting aims to provide all of the counterparties with the information they need in order to transact with a company. This can be termed the *information function* of corporate reporting [23]. Within this paper, we assume that the data collection and the subsequent data transformation are continuous processes, aggregation being the process that succeeds transformation. The terms “continuous information processing” and “continuous aggregation” are used alternatively, emphasising that within the continuous information processing, the continuous aggregation is the challenging part.

According to Cisco [9] “One of the biggest challenges facing an IT group is how to complete extract, transform, and load (ETL) and subsequently aggregate the traditional

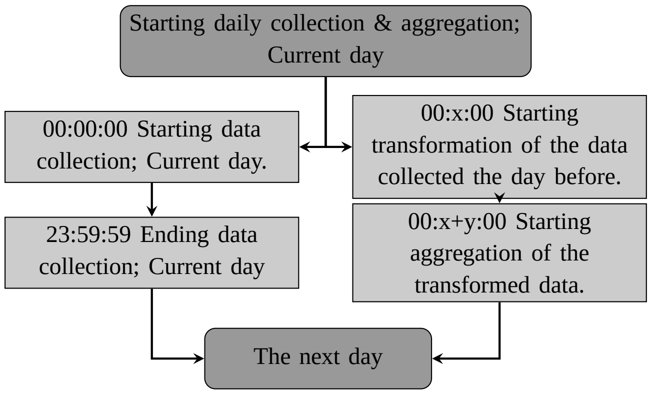
Starting daily collection & aggregation; Current day
00:00:00 Starting data collection; Current day.
00:x:00 Starting transformation of the data collected the day before.
23:59:59 Ending data collection; Current day
00:x+y:00 Starting aggregation of the transformed data.
The next day

Figure 1: Simplified flow diagram exemplifying the batch job strategy(x is the time gap due to the collection delay; y is the time gap due to transformation).

batch-based Business Intelligence (BI) processes within the constraints of an ever shrinking batch window." Although there is a trend toward real-time BI, the vast majority of BI today relies heavily on batch processing [9]. Cisco identified several factors which contribute to the difficulty in managing these processes in the foreseen time frame.

Firstly, there is a severe lack of visibility into the various processes themselves, which are very complex. This means there will always be bottlenecks, and it is very difficult to predict where they will occur.

Secondly, data streams are expected to arrive in a defined time window, if they are late or corrupted, then errors may occur. Thus, in these cases, the nightly aggregation routines have to be (re)started later, including also during the usual working hours. But Cisco does not identify two major factors that impact the time frame of the batch jobs, namely the impossibility to anticipate all the patterns of data to be processed and the imperfection of the executions plans, which can lead to performance degradation. Thus, the most accurate testing covers only the patterns of data retrieved in the past or anticipated. Accordingly, even the most accurate design and testing strategy cannot guarantee the time constraints of the batch jobs.

A. Overview of the CIPM

Following, the fundamental aspects of the continuous aggregation strategy are outlined by using two simple flow diagrams, Figure 1 depicts the principles of the classical nightly jobs aggregation strategy, whereas Figure 2 describes very succinctly the continuous aggregation strategy. It is assumed within these examples, that reporting is based on daily aggregated data. Data collection starts at 00:00:00 for the current day (i.e., the considered day) and it ends, retrieving data generated till 23:59:59 of the same day. Whenever applying the classical batch jobs strategy as depicted in Figure 1, the transformation/aggregation is started only after the data is fully retrieved/collected for the considered day. Considering this use case, the transformation/aggregation for the current day can be started only on the subsequent day.

On the other hand, the CIPM (exemplified in Figure 2) is carried out on small chunks of data, for example

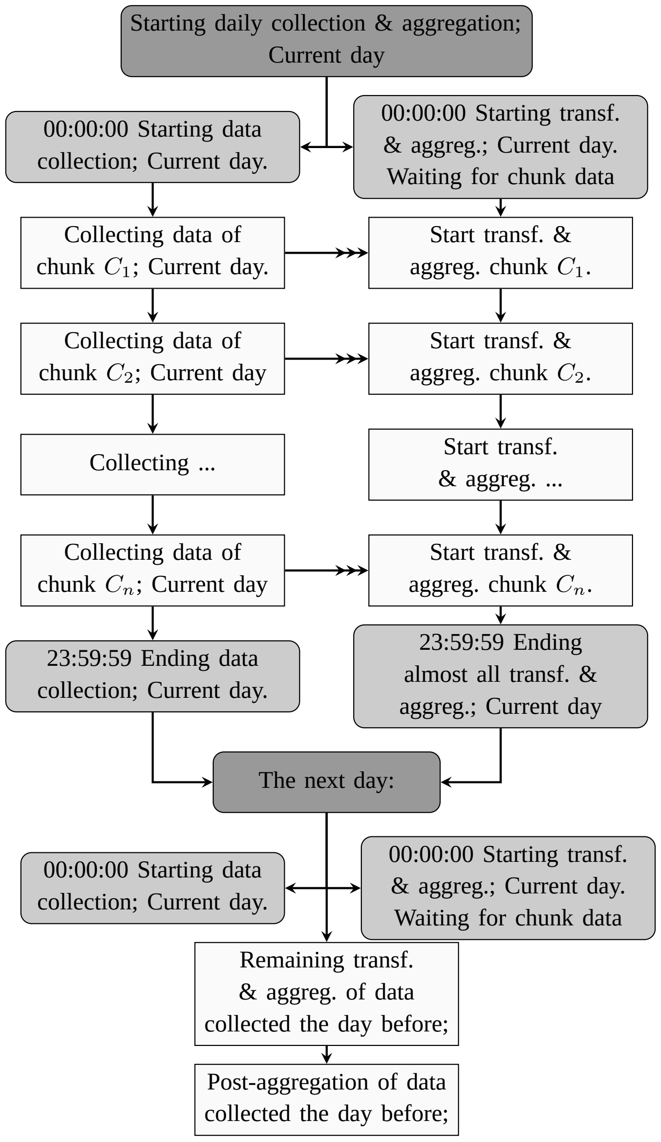
Starting daily collection & aggregation; Current day
00:00:00 Starting data collection; Current day.
00:00:00 Starting transf. & aggreg.; Current day. Waiting for chunk data
Collecting data of chunk C_1; Current day.
Start transf. & aggreg. chunk C_1.
Collecting data of chunk C_2; Current day
Start transf. & aggreg. chunk C_2.
Collecting ...
Start transf. & aggreg. ...
Collecting data of chunk C_n; Current day
Start transf. & aggreg. chunk C_n.
23:59:59 Ending data collection; Current day.
23:59:59 Ending almost all transf. & aggreg.; Current day
The next day:
00:00:00 Starting data collection; Current day.
00:00:00 Starting transf. & aggreg.; Current day. Waiting for chunk data
Remaining transf. & aggreg. of data collected the day before;
Post-aggregation of data collected the day before;

Figure 2: Simplified flow diagram exemplifying the continuous aggregation strategy. The arrow with three heads signifies that the aggregation phase waits till the respective chunk data has been collected. The post-aggregation phase of the previous day is not depicted.

$C_1, C_2, ..., C_n$, usually such that the transformation/aggregation is performed on data loaded into memory during the collection phase. This way, reloading data into memory for aggregation purposes is obsolete.

The continuous aggregation strategy is quite straightforward: after midnight, the collection phase for the current day is started, i.e., the chunks $C_1, C_2, ..., C_n$ are retrieved one after another in chronological order. While the second chunk C_2 is retrieved, transformation and aggregation are performed on the first chunk C_1, and so on and so forth. At the end of the current day, most of the collected data is aggregated. At the beginning of the subsequent day, the remaining chunk(s) are transformed/aggregated and a post-aggregation phase is started, during which the final calculations are performed. In the end, soon after midnight, the aggregated values are ready for reporting. Similarly, after midnight, the collection/transformation is resumed for the current day.

In order to keep the presentation simple and accessible and to avoid technical complications, it is assumed that the time to perform the transformation/aggregation of a chunk is slightly lower than the corresponding time of the collection phase. In real-world systems, under some circumstances, this is obviously not necessary. Let us suppose that the time to retrieve a chunk is t, but the time to transform/aggregate the values of a chunk is slightly lower than $3t$ and let A_i be the aggregation phase of chunk C_i. Then, the start of A_i is phase-shifted by t with regard to C_i, i.e., A_1 is started simultaneously with C_2, etc. As a consequence, A_i completes before $C_{(i+4)}$ is started. Hence, in this example there are three instances of the aggregation algorithm running in parallel. Possibly, information between the aggregation instances running in parallel need to be exchanged.

Additionally, it is assumed within this paper, that the chunks are of the same size and the time to retrieve them is independent of the particular chunk. Of course, this assumption is not necessary in real-world systems.

B. Exemplification using Standard Deviation (SD)

Next, the complexity of our approach is illustrated by exemplifying the technology on the *standard deviation of the sample*. It shows how much variation (dispersion, spread) from the mean exists. The SD has been chosen for exemplification, since the adaptations in order to be used within CIPM are straightforward to be presented without being trivial.

Let $\{x_1, x_2, \ldots, x_N\}$ be the observed values of the sample items, let $\bar{x} := 1/N \sum_{i=1}^{N} x_i$ be the mean value of these observations. The common representation of the (uncorrected sample) standard deviation is:

$$SD_N := \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2}. \tag{1}$$

At first glimpse, the above representation of the standard deviation cannot be applied using continuous computing techniques. The impediment is the term $\bar{x}$. In order to be able to apply the formula (1), all the data involved has to be first collected. Chan et al. [24], [25] called the above representation *two-pass algorithm*, since it requires passing through the data twice; once to compute $\bar{x}$ and again to compute SD_N. This may be unwanted if the sample is too large to be stored in memory, or when the standard deviation should be computed dynamically as the data is collected.

Regrouping the terms in the formula above, the well known representation is obtained:

$$SD_N = \frac{1}{N}\sqrt{\left| N\sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2 \right|}. \tag{2}$$

The alternative representation (2) of the standard deviation is suitable to be used within the continuous computation approach.

Let $1 \le n \le N$, let $A_n := \sum_{i=1}^{n} x_i^2$, let $B_n := \sum_{i=1}^{n} x_i$, and let $S_n := n \cdot A_n - B_n^2$.

During the data collection phase, the values of A_n and B_n are updated, either after each item x_n is collected and known to the system, or considering small batches. Thus, at each point in time, during the data collection phase, the values of $A_{(n+1)}$ and $B_{(n+1)}$ can be easily calculated by adding up the corresponding value of the new item. Hence,

$$A_{(n+1)} = A_n + x_{(n+1)}^2.$$

Similar results hold for $B_{(n+1)}$.

Accordingly, at each point in time, the standard deviation can be easily calculated, if needed, as a function of A_n, B_n and S_n. It follows:

$$S_{(n+1)} = S_n + A_n + n \cdot x_{(n+1)}^2 - 2x_{(n+1)} \cdot B_n.$$

Hence, intermediary results and trend analysis are possible during the data collection phase .

For example, almost all *Key Performance Indicators* (KPIs) used in the semiconductor industry can be adapted to be applied within CIPM [26]–[29]. The same is true in other areas of the industry or business.

C. Reasons for choosing SD

The considerations above were drafted merely to illustrate the continuous computation technology, in real-world systems the representation (2) without using absolute values in the square root function can lead to negative values. When A_N and B_N are calculated in the straightforward way, especially when N is large and all of x-values are roughly of the same order of magnitude, rounding or truncation errors may occur [30]. Please note that the representation (2) using absolute values, has been adopted in order to avoid negative values under the square root. Using double precision arithmetic can possibly avoid the occurrence of anomalies as above.

D. Counterexample

Unfortunately, there are also some simple and well known functions, such as the *Average Absolute Deviation (AAD)*, which, generally speaking, cannot be used with continuous computing techniques; AAD is calculated as the mean of the sum of the absolute differences between a value and the central point of the group:

$$AAD_N = \frac{1}{N}\sum_{i=1}^{N}|x_i - M|.$$

The central point M can be a mean, median, mode, etc. For some distributions, including the normal distribution, AAD can be related to or approximated with the corresponding standard deviation [31]–[33].

Under some circumstances holistic functions, such as the AAD presented above, can be used within real-time applications. As already mentioned, there is no constant bound on the size for the storage needed to compute holistic functions. If the task is to convert the nightly batch aggregation to continuous aggregation and real-time constrains are only required for the data of the last 24 hours or the current day, then obviously the size of the required storage is bounded, since the number of items expected during 24 hours can be very well estimated. Hence, if the time required to calculate the value of AAD is within the time constraints imposed, then real-time capability of the application is given. On the contrary, if real-time capability is required for example for streaming data, then this depends heavily on the size of the streamed data comprising the time interval required and the possibility to predict the size of the data. In such cases, approximations can help, see [31]–[33] for approximations based on standard deviation.

E. Flow Factor

Next, in order to illustrate the power and significance of the continuous aggregation strategy, a real-world example from the semiconductor industry [27], [29], [34] is presented. For the sake of completeness, a slightly modified part of the model is taken over from [29]; the model has been originally described for data centres. In order to be able to understand and follow the presentation, some basic data structures are introduced.

1) Description of the data structure for WIP-data: In simplified terms, the lot/wafer enters the production line and it is processed consecutively on (different) operations/equipments according to the production plans (i.e., routes) and leaves the frontend area to be systematically examined/tested in the backend area. Afterwards, the wafers are cut into segments, also called dies, which are packaged as chips and are ready to be shipped. There are two major data structures, Work in Progress (WIP)-data and Equipment-data. The WIP-data models the successive processing of the lot/wafer at different operation in the frontend area and the Equipment-data models the data associated to an equipment.

A simplified data structure for WIP-data is: (*"unit-ID", "step", "next-step", "track-out-TS", "trans-code", "equipment", "product", "unit-type", "unit-desc", "unit-value", "route"*). The unit is the manufactured item, which is tracked by the Manufacturing Execution System (MES). It can be a lot, a wafer or a die/chip. This information is tracked by the attribute "unit-desc". The attribute "unit-type" is an additional distinction between the material units, such that the units are *Productive, Development*, *Test*, *Engineering*, etc. The transaction code ("trans-code") denotes the event that is performed at a specific "step" and "equipment" during the production process. Common transaction codes in the semiconductor industry are *TrInT, TrOutT, Create a Lot*, *Ship a Lot*, etc. The attribute "product" characterizes the manufactured item, (like technical specifications, etc.), which can be tracked within the production process. The "step" is the finest abstraction of the processing level, which is tracked by the reporting system. It is usually the operation [8]. The attribute "next-step" denotes the succeeding "step" (i.e., operation) to which the item is transferred. Additionally, the attribute "track-out-TS" stores the time stamp of the item when it left the "step" to be processed at the "next-step". This is triggered through the transaction code TrOutT. The attribute "unit-value" contains the number of items processed at the considered "step", whereas the "route" contains the processing specification for the "product" involved.

2) Description of the aggregation functions: The following definitions are taken from [29]. Let u be a unit. We denote by $TrInT_s(u)$ the *track in time* of u, i.e., the point in time when the processing of unit u is started at "step" s. Analogously, $TrOutT_s(u)$ is the *track out time* of u, i.e., the point in time when the processing of unit u has been finished at "step" s.

The *Raw Process Time (RPT)* of a unit u related to "step" s is the minimum processing time to complete the "step" s without considering waiting times or down times. We denote the raw process time of unit u related to "step" s by $RPT_s(u)$. We assume that for a "step" s, the function $succ_s(u)$, which identifies the succeeding "step" of s for the unit u is well defined. Analogously, we assume that the history of the production process is tracked, so the predecessor function $pred_s(u)$ of each "step" s is well defined.

By *Cycle Time (CT)* we denote the time interval a unit or a group of material units spent in the system or subsystem. The cycle time of a unit $u \in U$ spent at a "step" $s \in S$ in the system can be represented as:

$$CT_s(u) := TrOutT_s(u) - TrOutT_{pred_s(u)}(u). \tag{3}$$

In order to simplify the notation to be used within the Transact-SQL syntax for SQL Server we set

$$PrevTrOutT = TrOutT_{pred_s(u)}(u).$$

Hence, using the above notation, we have the expression for the cycle-time corresponding to a unit:

$$CT = Datediff(ss, TrOutT, PrevTrOutT). \tag{4}$$

where $Datediff(ss, TrOutT, PrevTrOutT)$ returns in seconds the specified time difference.

Algorithm 1 Sample search query for the calculation of the flow factor - standard method

```
Input: WIP-data-table_name;
Output: Flow factor;
Result: Compound aggregated value of the flow factor ready for
        reporting on week 6 of 2022;
/* Retrieving the Flow Factor(FF)                          */
select sum(CT)/sum(RPT) as FF
from from WIP-data-table_name
where TrOutT > ’06.02.2022 00:00:00’
and TrOutT <= ’13.02.2022 00:00:00’
group by "step", "equipment", "product", "unit-type", "unit-desc"
```

Figure 3: SQL-code based algorithm retrieving the flow factor and exemplifying the standard method for ad-hoc reporting strategy by enabling real-time capability. For this purpose, the representation (4) of the cycle time has to be used, whereas the representation (3) is inefficient.

3) Description of the problem: The SQL-query based on the representation in Figure 3 and representation (3) is inefficient due to the fact that the chronological order of the steps is not fully specified by the "route". The human operator has the possibility to determine the subsequent "step", for example to perform some additional measurements or not. Hence, to determine the previous "step", i.e., the attribute notated as "prev-step", appropriate joints have to be set up, such that in the end the value of the attribute "PrevTrOutT" is determined. Due to the fact that processing can be done in a loop for rework processes, the item with the greatest time-stamp has to be selected. In conclusion, using suboptimal data structures for reporting can do the task of retrieving the specified data to be very difficult or impossible.

4) Solving the problem: In order to circumvent the problem as above, the data structures have to be adapted according to our guiding principle such that within the information flow, the process of information setup should be started as early as possible. Two new attributes, "prev-step" and "prev-track-out-TS" are added to the structure of WIP-data and these attributes are filled during the collection phase. The attribute "prev-step" identifies the previous step with regard to the actual "step" and "prev-track-out-TS" holds the corresponding time stamp when the item left the previous step. Since data is collected chunk-wise, the cartesian product due to the corresponding joints is relatively small and the initialisation of those attributes can be done during the collection phase.

The aggregated structure may look like this: ("step", "equipment", "product", "unit-type", "unit-desc", "cycle-time", "raw-process-time"). During the small scale aggregation phase, the values of the attributes "cycle-time" and "raw-process-time" are summed up according to the corresponding grouping. For example, if an aggregated daily table is set up, then the flow factor for the lowest granularity can be calculated by just dividing two numbers. As a consequence, the query to calculate the flow factor is reduced to a simple select on a single aggregated table, see algorithm Figure 3.

5) Example for the data structure for RTC-data : For the Equipment-data an oversimplified structure is: ("equipment-ID", "chamber-ID", "work-station", "time-stamp", "current-state"). Real Time Clock (RTC) systems record the current state of the equipment [35], hence the equipment data as above is also termed RTC-data. The current-status of the equipment can be among others [36]: *Scheduled-downtime, Unscheduled-downtime, Non-scheduled-time, Productive-time, Engineering-time*, etc. For reporting purposes, the structures of WIP-data with RTC-data are combined, for example to determine the standard deviation of the cycle time concerning an equipment or group of equipments, since a certain operation can be performed on different equipments.

In conclusion, the principles of the CIPM as opposed to the classical batch jobs strategy are presented in this section. For this purpose, two representations of the standard deviation are discussed The first representation is not suitable to be applied within CIPM, whereas on the second representation the principles of the CIPM are exemplified. For the sake of completeness, a holistic function is presented which cannot be generally applied within CIPM. Nevertheless, under some circumstances, holistic functions can be used in real-time environment, the strategy to be used depends on the use case. Lastly, a real-world problem from a semiconductor company is presented. Using some principles of CIPM such as redesigning the calculation methodology, a more energy efficient, less time consuming solution is presented, which can also be used within CIPM.

IV. The Formal Model

We further formalise the description of our methodology [1] by developing a mathematical model, in order to be able to use the advantages of the rigour of a formal approach over the inaccuracies and the incompleteness of natural languages. In order to keep our model simple, transparent, and easily comprehensible, some assumption are made, for example equal length of the chunks, etc. These assumptions are not strictly necessary in a real-world environment. Hence, within the continuous information processing strategy, ensuring the continuity of data collection phase and the continuity of the transformation phase within the CIPM are more or less straightforward. Hence, the terms continuous information processing and continuous aggregation are used synonymously within this paper.

A. General considerations

Assumption 1 (Finite streams) *We suppose that the streams are* finite, *i.e., there are two points in time, t_s, the starting and t_e, the ending point, such that within this time interval, the data is collected/transformed and aggregated.* ■

Assumption 2 (Synchronous data delivery) *We suppose that each stream delivers data at the same points in time $\{1, 2, \ldots, T\}$.* ■

Definition 1 (Large scale aggregation) *If the aggregation*

a) *occurs after the entire raw data has been previously collected,*
b) *involves technologies that process all of the collected data at once,*

then the process is termed batch aggregation mode or large scale aggregation. ■

Definition 2 (Small scale aggregation) *If:*

a) *the collected data within the interval $[t_s, t_e]$ can be split into $k \geq 2$ smaller units, also termed chunks,*
b) partial aggregation *is performed on these units, such that the final aggregation values are calculated out of the corresponding partial values of the chunks,*

then the process is termed small scale aggregation. ■

Assumption 3 (length of the chunks) *We suppose that data within the interval $[t_s, t_e]$ can be split into $k \geq 2$ disjunct equal units of length $l > 1$, i.e., $[t_s, t_e] = [t_s, t_{s+l}]$, $[t_{s+l+1}, t_{s+2l}]$, ...,$[t_{s+(k-1)l+1}, t_e]$.* ■

Remark 1 *Some authors specify the terms large scale aggregation or small scale aggregation regarding their ability to perform the computation in memory. Within this paper, a more algorithmic than a technical approach is followed.* ■

The term "transformation" has been introduced for the seek of completeness. In simplified terms, it is the process of harmonisations of the data structures of the raw data from different sources, such that it is best finalised for aggregation. Moreover, during transformation, the initial streams can be combined to new ones, in order to facilitate the aggregation process. During the transformation phase, the data is also verified for accuracy. In principle, the granularity of the data is not altered during the transformation phase. In order to avoid technical complications, the concept of data transformation is not considered within the formal model, but formally, it can be considered a part of the data retrieval. Assuring continuous computation during the transformation phase is more or less straightforward.

Assumption 4 (Aggregation time) *In order to be able to continuously compute – i.e., retrieve/collect and aggregate – the time to aggregate a particular small batch does not exceed the collection time of the same batch.* ■

Remark 2 *Obviously, if Assumption (4) is not met, the aggregation cannot be performed in all circumstances during the data collection phase.* ■

Notation 1 (Streams) *Let $l_X \in \mathbb{N}$ be the number of streams and let*

$$X := \{X^{(1)}, X^{(2)}, \ldots, X^{(l_X)}\}$$

be the set of streams.

Let $\{1, 2, \ldots, T\}$ be the points in time when the data is collected and known by the system, let $1 \leq t \leq T$ and let $1 \leq l \leq l_X$. The value of the stream $X^{(l)}$ collected at time t is denoted by $x_t^{(l)}$. ■

Representation 1 *The streamed values can be represented as a matrix*

$$(x_{tl})_{1 \leq t \leq T; 1 \leq l \leq l_X} \text{ such that } x_{tl} = x_t^{(l)}.$$

■

Notation 2 (Grouping, bundle of streams) *Let $l_F \in \mathbb{N}$ be the number of the aggregation functions.*

In order to perform the computation of the streams – the aggregation functions are in general functions of several variables – a grouping

$$B := \{B^{(1)}, B^{(2)}, \ldots, B^{(l_B)}\}$$

is defined on the space of the streams, such that

$$B^{(l)} := \{X^{(l_1)}, X^{(l_2)}, \ldots, X^{(l_i)}\}.$$

and $l_F = l_B$.

Accordingly:

$$b_t^{(l)} := x_t^{(l_1)} \times x_t^{(l_2)} \times \cdots \times x_t^{(l_i)}$$

is the value of the grouping $B^{(l)}$ at time t. ■

This way, new compound streams are created.

Assumption 5 (No. of grouping = No. of functions) *In order to keep our model as simple as possible, it is supposed – without limiting the generality – that the number of groupings is equal to the number of aggregation functions.*

Notation 3 *Let*

$$F := \{F^{(1)}, F^{(2)}, \ldots, F^{(l_F)}\}$$

be the set of the aggregation functions, such that for $1 \leq l \leq l_F$

$$F^{(l)} : B^{(l)} \to \mathbb{R}.$$

■

Remark 3 (Aggregation strategy) *In order to keep the model as general as possible, small scale aggregation is considered as the overall approach.* ■

This means especially, that data is collected and aggregated in small batches.

Notation 4 (Disassembling the standard deviation) *Let $1 \leq l \leq l_F$ and let $F^{(l)} : B^{(l)} \to \mathbb{R}$ be the standard deviation, see representation (2) and let $x \in B^{(l)}$ a particular stream. Let $1 \leq t \leq T$ and let:*

$$\begin{aligned}
f_t^{(l,1)}(x) &:= \sum_{i=1}^{t} x_i^2, \\
f_t^{(l,2)}(x) &:= \sum_{i=1}^{t} x_i, \\
f_t^{(l,3)}(x) &:= t \sum_{i=1}^{t} x_i^2 - \Big(\sum_{i=1}^{t} x_i\Big)^2 \\
&= t \cdot f_t^{(l,1)}(x) - (f_t^{(l,2)}(x))^2. \qquad (5)
\end{aligned}$$

Let $F_t^{(l)}(x)$ be the value of the function $F^{(l)}$ applied on the values subscripted by $1 \leq t \leq N$. ■

Proposition 1 (Calculation of the standard deviation) *The value $F_t^{(l)}(x)$ of the function $F^{(l)}$ can be calculated out of the values of $f_t^{(l,1)}(x), f_t^{(l,2)}(x)$, i.e., by considering $f_t^{(l,3)}(x)$, namely:*

$$F_t^{(l)}(x) = \frac{1}{t}\sqrt{\left|f_t^{(l,3)}(x)\right|}. \qquad (6)$$

■

B. Information processing

In the following, it is considered that the data is retrieved and aggregated in chunks, and that before performing aggregation, the chunks are not altered.

Notation 5 (Chunk-wise processing) *Let $j, q \geq 1$, such that j is the index and q is the length of the chunks and let us suppose that the streams are retrieved in small chunks:*

$$C_j := \{C_j^{(1)}, C_j^{(2)}, \ldots, C_j^{(l_X)}\}$$

of q items, i.e., the chunk $C_j^{(l)}$ of the stream $x \in B^{(l)}$ consists of the values:

$$C_j^{(l)} := \{x^{(l)}_{((j-1)q+1)}, x^{(l)}_{((j-1)q+2)}, \ldots, x^{(l)}_{(jq)}\}.$$

■

The information is processed chunk-wise, first C_1 is retrieved. As long as the next chunk C_2 is retrieved, aggregations is performed on C_1 simultaneously, then chunk C_3 is retrieved by simultaneously aggregating chunk C_2, and so on and so forth.

During the chunk-wise processing, the values of $f^{(l,1)}_{(j+1)q}$ and $f^{(l,2)}_{(j+1)q}$ corresponding to the subsequent chunk, can be easily calculated out of $f^{(l,1)}_{jq}$ and $f^{(l,2)}_{jq}$, for example:

$$f^{(l,1)}_{(j+1)q}(x) = f^{(l,1)}_{jq}(x) + \sum_{i=1}^{q} x^2_{jq+i}.$$

The value $F^{(l)}_{jq}$ corresponding to the standard deviation at time $t = jq$ can be calculated at each "step" corresponding to the points in time $\{1, 2, ..., T\}$, or alternatively, after having reached the end of the collection phase. This phase is termed *post aggregation phase*.

Definition 3 (Post aggregation phase) *The additional calculation, which is performed after all chunks have been retrieved and aggregated, is termed post aggregation phase.*

Since the small scale aggregation should be as fast and effective as possible, the functions $f^{(l,3)}_{jq}, F^{(l)}_{jq}$ must not necessary be calculated for each chunk, as it is retrieved – if there is no requirement in this direction – they can also be calculated on a case by case basis by the tool that visualises intermediary results.

C. Truncation errors

A discussion regarding the truncation errors is beyond the scope of this paper. As already mentioned, when N is large and all of x-values are roughly of the same order of magnitude, rounding or truncation errors may occur when $f^{(l,1)}_t$ and/or $f^{(l,2)}_t$ for $1 \leq t \leq T$ are evaluated in the straightforward way [30]. A greater accuracy can be achieved by simply shifting some of the calculation to double precision, see [24], [25] for a discussion on rounding errors and the stability of presented algorithms. Barlow presents an *one-pass-through* algorithm [37], which is numerically stable and which is also suitable for parallel computing.

The scope of the presentation in this paper is merely to illustrate the technology. Of course, if a function does not allow an one-pass algorithm, it cannot be used directly for continuous computation. A classical example in this direction is the average absolute deviation, as mentioned before, in some cases there are approximate one-pass implementation of the algorithms.

D. General case

So far, the standard deviation was considered as exemplification. Now, let us formalise the continuous aggregation strategy and consider the general case:

Proposition 2 (Reassembling the partial functions)

Let $1 \leq l \leq l_F$, let $j, q \geq 1$, such that j is the index and q is the length of the chunks. Let

$$F^{(l)} : B^{(l)} \to \mathbb{R}$$

be an aggregation function such that:

a) there exists l_f real valued functions

$$f^{(l,1)}, f^{(l,2)}, \ldots, f^{(l,l_f)} \text{ defined on } B^{(l)}$$

such that for each chunk $C_j^{(l)}$, the values of

$$f^{(l,i)}_{(j+1)q} \ (1 \leq i \leq l_f)$$

can be calculated out of the values of $f^{(l,i)}_{jq}$ and $C_j^{(l)}$,

b) $F^{(l)}$ is a function of $f^{(l,i)}$ for all $1 \leq i \leq l_f$.

Then intermediary results, such as the value of $F^{(l)}_{(j+1)q}$ can be calculated out of $f^{(l,i)}_{(j+1)q}$ $(1 \leq i \leq l_f)$. ■

Assumption 6 *Let j_f be the index of the final chunk to be processed. Obviously, the composition algorithms should ensure that the value $F^{(l)}_{j_f \cdot q}$ does not depend on the size of the chunks.* ■

E. Reprocessing

In practical systems, in general, there should be a technology in place that allows recalculation of the aggregated values. This is necessary if for any reason whatsoever, some stream values are erroneous. Sometimes, it takes time to correct them, since not all wrong values can be detected and corrected automatically. Regarding the standard deviation, two new functions $df^{(l,1)}_t$ and $df^{(l,2)}_t$ can be introduced, such that

$$df^{(l,1)}_{jq}(x) := \sum_{i=1}^{q} x^2_{jq+i}$$

and

$$df^{(l,2)}_{jq}(x) := \sum_{i=1}^{q} x_{jq+i}.$$

Thus, correct and updated computed values can be achieved, for example by adding to $f^{(l,3)}_T$ the new value of $df^{(l,1)}_{jq}$ and subtracting the corresponding old value $df^{(l,1)}_{jq}$, similar considerations are valid for $df^{(l,2)}_{jq}$. This means especially, that the corresponding values for the initial chunk and the corrected chunk have to be (re)calculated. In the end, the value $F^{(l)}_{j_f \cdot q}$ has to be recalculated. As already mentioned, the above considerations are included in order to illustrate the methodology. In practice, better suited algorithms can or should be used instead.

F. Pseudo-code algorithm exemplification

In the following, a simplified algorithm to exemplify our continuous aggregation strategy is sketched. It is based on disassembling the standard deviation $F_t^{(l)}$ using the functions $f_t^{(l,1)}, f_t^{(l,2)}, f_t^{(l,3)}$, see equation (5) and (6). In order to keep the representation of the algorithm simple, it is supposed that the chunks have the same length equal to l_{chunk}, see Assumption 3. The corresponding algorithm is presented in Figure 4. In real-world systems, the data collection may involve also time limits t_{Max}, such that these time limits restrict the length of the chunks.

1) Ad-hoc reporting including request for up-to-date data: Granularity defines the level of detail of information. The finer (higher) the level of granularity, i.e., smaller grains, the deeper the level of detail. In time-series data, for example, the granularity of the data might be at a millisecond, second, minute or hour level, the higher the level of granularity, the more information is available for analysis. The lowest level of granularity is the granularity of the raw data, i.e., of the data collected by the streams. For analysing purposes, in real-word systems, the lowest level of granularity may be to fine grained, for example, the streams deliver data within milliseconds and with the highest precision, but this low level of detail is pointless for reporting. In this case, a *pre-aggregation* is performed by calculating the average or similar of the numeric values. We suppose that the streams do not contain calculated or aggregated data. Coarser level of granularity can be achieved by aggregation.

2) Term ad-hoc: According to Merriam-Webster entry 1, the term ad-hoc means “for the particular end or case at hand without consideration of wider application”. Ad-hoc reporting is a model of Business Intelligence (BI), in which reports are build and distributed by non-technical BI-users to meet individual and department information requirements. The perception in the industry is that ad-hoc reporting refers to the capability to develop reports by dragging and dropping query items from the meta-model onto a design surface [38]. This means especially that ad-hoc reporting is not an “improvised” or unplanned reporting, as the definition entry 2 of Merriam-Webster for the term ad-hoc would suggest, but a very well thought out reporting environment set up by BI experts, such that the corresponding tools can be also used by non-specialists.

3) Example from the industry: Generally speaking, in the semiconductor industry, the lowest level of granularity, which characterises the production item (wafer or integrated circuits), is the *product*. The next level of hierarchy is the *product-group*, succeeded by the *product-class*, by the *technology*, and in the end by the *production-line*. The usual/standard reporting summarises the aggregated values on product, product-group, product-class, technology, or production-line level.

Let us suppose that an equipment owner, who processes a new product on his equipment, would like to report the

Algorithm 2 Sample code exemplifying the continuous aggregation strategy

```
Input: Stream $x$
Output: $F_t^{(l)}(x)$ at each retrieval point in time $t$
Result: Aggregated value of the standard deviation ready for reporting at any retrieval point in time $t$
double precision $f^{(l,1)}(x) = 0$ ;          // component function
double precision $f^{(l,2)}(x) = 0$ ;          // component function
double precision $F^{(l)}(x) = 0$ ;      // value of the standard deviation corresponding to the state of collection
int $l_{chunk} = 10,000$ ;          // number of items of a chunk
float $[l_{chunk}]$ $c$ ;          // retrieved values of a chunk
float $[l_{chunk}]$ $c_{prev}$ ;          // data of the previous chunk
int $L_{col} = 0$ ;          // length of the collection
// ----------------------------------------------------
/* data of the length of a chunk is collected     */
procedure collection() {
int $l_{cur} = 0$ ;          // length of the collected data
repeat
    collect data into $c$ ;     // collect data into the chunk
    $l_{cur}++$;          // until the chunk is full
until $l_{cur} = l_{chunk}$ ;
for $i = 1$ to $l_{chunk}$ by $1$ do $c_{prev}[i] := c[i]$ ; // copy $c$ to $c_{prev}$
} // ----------------------------------------------------

/* data of the length of a chunk is collected     */
procedure aggregation() {
float$[l_{chunk}]$ $x$ ; // contains data of the previous chunk
for $i = 1$ to $l_{chunk}$ by $1$ do $x[i] := c_{prev}[i]$; // copy $c_{prev}$ to $x$
/* calculation of the functions composing the standard deviation     */
$f^{(l,1)}(x) := f^{(l,1)}(x) + \sum_{i=1}^{l_{chunk}} (x[i])^2$;
$f^{(l,2)}(x) := f^{(l,2)}(x) + \sum_{i=1}^{l_{chunk}} x[i]$;
$L_{col} := L_{col} + l_{chunk}$;          // number of items collected
$F^{(l)}(x) := \frac{1}{L_{col}} \sqrt{|L_{col} \cdot f^{(l,1)}(x) - (f^{(l,2)})^2(x)|}$;// only if required
} // ----------------------------------------------------

/* final calculation of the standard deviation     */
procedure post-aggregation() {
$F^{(l)}(x) := \frac{1}{L_{col}} \sqrt{|L_{col} \cdot f^{(l,1)}(x) - (f^{(l,2)})^2(x)|}$;
} // ----------------------------------------------------

/* start aggregation in parallel to data collection phase     */
procedure void main() {
collection();
repeat
    start in parallel the threads: collection() & aggregation()
    wait until both threads have finished
until collection phase is over ;
aggregation();          // due to the phase shift
post-aggregation();          // final aggregated values
}
```

Figure 4: Pseudo-code based algorithm using standard deviation exemplifying the continuous aggregation strategy.

standard deviation for the cycle time or waiting time for the last week. The standard reporting system does not cover this case, since for example, the first three days of the week the old product is processed, then the equipment is adjusted for the new product, which is processed afterwards. The standard reporting covers the first three days of the week for the old product and the remaining days for the new product. Hence, mixed reporting, covering more products should be set up. The task is not trivial, since the standard deviation corresponding to the mixed week cannot be calculated out of corresponding values of the standard deviation of each product. The example as above illustrates also the inherent difficulties of ad-hoc reporting, it may imply substantial effort on the BI side.

G. Model extension

Next, we extend our formal model in order to cover the difficulties as above by exemplifying the methodology on the standard deviation, see algorithm Figure 5. Similar to the continuous aggregation strategy:

Proposition 3 (Calculation of compound streams)
- *let t be a point in time when streams are collected,*
- *let x and y be streams such that x contains data for the product p and y contains data for the product q,*
- *let n_t^x be the number of items of the stream x corresponding to time t,*
- *similarly, let n_t^y be the number of items of the stream y corresponding to the same point in time t,*
- *let $f_t^{(p,1)}(x)$ and let $f_t^{(q,1)}(y)$ be the corresponding values for the stream x and y respectively.*

Set

$$f_t^{((p+q),1)}(x+y) := f_t^{(p,1)}(x) + f_t^{(q,1)}(y)$$
and
$$f_t^{((p+q),2)}(x+y) := f_t^{(p,2)}(x) + f_t^{(q,2)}(y).$$

Let

$$f_t^{((p+q),3)}(x+y) := (n_t^x + n_t^y) \cdot f_t^{((p+q),1)}(x+y) - \left(f_t^{((p+q),2)}(x+y)\right)^2$$
and
$$F_t^{(p+q)}(x+y) := \frac{1}{(n_t^x + n_t^y)} \cdot \sqrt{\left|f_t^{((p+q),3)}(x+y)\right|}. \quad (7)$$

Then $F_t^{(p+q)}(x+y)$ determines the values of the standard deviation corresponding to the compound stream $x+y$ at time $t \in \{1, 2, ..., N\}$. ■

The calculation of the standard deviation relying on the formula (7) should be applied by a reporting tool within the ad-hoc reporting strategy. A closer look at the relation (7) reveals that all components like $f_t^{(p,1)}(x)$, $f_t^{(q,1)}(y)$, etc., are precalculated during the continuous aggregation period.

Remark 4 (Calculation time is stream size independent) *Hence, the calculation time for the standard deviation does not depend on the size of the stream or the point in time when the calculation is performed.*

Conclusion 1 (Real-time capability of ad-hoc reporting) *Thus, ad-hoc reporting set up as in the example above in a real-time environment has real-time capability.* ■

Algorithm 3 Sample code exemplifying the ad-hoc reporting strategy

```
Input: Stream $x$; Stream $y$;
Output: $F_t^{(l)}(x+y)$ at each retrieval point in time $t$;
Result: Compound aggregated value of the standard deviation ready
        for reporting at the last point in time $t \in \{1, 2, \ldots, T\}$ for
        which data has been collected;
double precision $f_t^{(l,1)}(x) = 0$ ;          // component function
double precision $f_t^{(l,1)}(y) = 0$ ;          // component function
double precision $f_t^{(l,2)}(x) = 0$ ;          // component function
double precision $f_t^{(l,2)}(y) = 0$ ;          // component function
double precision $f_t^{(l,1)}(x+y) = 0$ ;              // calculated
double precision $f_t^{(l,2)}(x+y) = 0$ ;              // calculated
double precision $F_t^{(l)}(x+y) = 0$ ; // value of the standard
deviation
int $n_t(x) = 0$ ;               // number of items belonging
to Stream $x$ collected till the point in time of the
ad-hoc reporting; corresponds to the last time-stamp
of the collected data
int $n_t(y) = 0$ ;
int $n_t(x+y) = 0$ ;
ad-hoc-retrieval() {
/* calculation of the functions composing the
standard deviation                                        */
$n_t(x) \Leftarrow retrieve\ the\ value\ from\ database$
$n_t(y) \Leftarrow retrieve\ the\ value\ from\ database$
$f_t^{(l,1)}(x) \Leftarrow retrieve\ the\ value\ from\ database$;
$f_t^{(l,1)}(y) \Leftarrow retrieve\ the\ value\ from\ database$;
$f_t^{(l,2)}(x) \Leftarrow retrieve\ the\ value\ from\ database$;
$f_t^{(l,2)}(y) \Leftarrow retrieve\ the\ value\ from\ database$;
$f_t^{(l,1)}(x+y) \leftarrow f_t^{(l,1)}(x) + f_t^{(l,1)}(y)$;
$f_t^{(l,2)}(x+y) \leftarrow f_t^{(l,2)}(x) + f_t^{(l,2)}(y)$;
$n_t(x+y) \leftarrow n_t(x) + n_t(y)$;
$F_t^{(l)}(x+y) :=$
    $\frac{1}{n_t(x+y)} \sqrt{\left| n_t(x+y) \cdot f_t^{(l,1)}(x+y) - (f_t^{(l,2)})^2(x+y) \right|}$;
}
```

Figure 5: Pseudo-code based algorithm using standard deviation exemplifying the principle of the ad-hoc reporting strategy and enabling real-time capability.

H. Parallel execution of the continuous aggregation routines

So far, in order to keep the algorithms simple and transparent, simplified algorithms were presented. The aim was to exemplify the principle, such that they could be easily adapted to solve real-world situations. Next, load balancing techniques are presented. The advantage of these techniques is that it can bypass Assumption (4), which limits the aggregation time and it supposes that it does not exceed the retrieval time of the chunks.

Let j be the number of parallel instances of the aggregation routine that calculates components of the standard deviation, see procedure "aggregation" in Figure 4. This procedure has to be adapted such that it calculates partial values corresponding to the function $f^{(l,1)}$ and $f^{(l,2)}$. The implementation is quite straightforward, hence only some hints are given. Define the header as: "procedure aggregation (float[l_{chunk}] x)" Define the corresponding local variable $\delta f^{(l,1)}(x)$ and set

$$\delta f^{(l,1)}(x) := \sum_{i=1}^{l_{chunk}} (x[i])^2; \qquad (8)$$

Add the statement

$$f^{(l,1)} := f^{(l,1)} + \delta f^{(l,1)}(x) \qquad (9)$$

for the global variable $f^{(l,1)}$. Thus, by calling j parallel instances of the procedure "aggregation", j chunks are processed in parallel. We assumed in this example that statement (8) is the time consuming one.

I. Benefits in the software development process

1) Transparent software development: One of the outstanding advantages of the continuous aggregation strategy is the possibility to simplify and align/harmonise the set-up process of aggregation, thus leading to faster, modularised and more effective and transparent software development. This involves improved maintenance possibilities due to its conceptual unity. Moreover, people can be trained much easier on maintenance, since the software developed is not the outcome of individual abilities and unique skills, but of very well specified methodologies.

2) Paradigm shift: Lewis [39], [40] stated that *software construction is an intrinsically creative and subjective activity and as such has inherent risks*. Lewis added: *the software industry should value human experience, intuition, and wisdom rather than claiming false objectivity and promoting entirely impersonal "processes"*.

Our contribution is a "step" in setting up objective criteria regarding software developing processes, such that it *can be a science, not just an art*, paraphrasing Roetzheim's statement [41] regarding software estimate. This way, our approach facilitates the *paradigm shift* from a subjective software construction activity, towards objectively verifiable straightforward strategies. Our approach does not claim that the overall effort of the transition from large scale aggregation to small scale aggregation is diminishing, the complexity of converting multi-pass algorithms to one-pass algorithms should not be underestimated. It does require *intrinsically creative* and *subjective activity* as formulated by J.P. Lewis, but merely on the algorithmic side.

J. Real-time capability

1) Real-time systems: The term *continuous information processing* involves incessant data collection and steady aggregation, such that preliminary aggregated results corresponding to the current status of the collected data are available for evaluation purposes. Continuous processing of large amounts of data is primarily an algorithmic problem [42].

Real-time systems are subject to time constrains, i.e., their actions must be fulfilled within fixed bounds. The perception of the industry of real-time is first of all fast computation [43]. Moreover, TimeSys [44] requires the following features for a real-time system:

a) *predictably fast response to urgent events,*
b) *high degree of schedulability*: the timing requirements of the system must be satisfied at high degrees of resource usage,
c) *stability under transient overload*: when the system is over-loaded by events and it is impossible to meet all the deadlines, the deadlines of selected critical tasks must still be guaranteed.

The characterisation above exemplifies the different requirements in some fields of the industry. A real-time system requires real-time capability of the underlying components, including the operating system, etc. These considerations show the immanent difficulties of the industry to cope with the complexity of real-time requirements of opaque and incomprehensible systems.

2) Real-time capability of CIPM: In order to point out the real-time capability of a continuous information processing system, its behaviour is analysed and it is shown that it satisfies the given time limits. In real-world systems, it is supposed that the *maximum size of the streaming data* and the *streaming speed* are known and these thresholds are not exceeded.

With the aim to keep the argumentation simple and straightforward, it is assumed that the streaming speed is constant, i.e., the same amount and type of data is collected within equal time intervals. Hence, it is appropriate to setup chunks of data of the same size collected within equal time spans, such that the aggregation time of different chunks is equal. The aggregation time t_{agg} of a particular chunk should not exceed its retrieval time t_{ret}, i.e., $t_{agg} \leq t_{ret}$, else data to be aggregated will accumulate, see Assumption 4.

The strategy to achieve real-time behaviour based on continuous stream computing is straightforward.

Remark 5 (Condition for achieving real-time capability) *Let t_C be the time constraint such that within the time interval specified accordingly, aggregated data should be available. In order to have real-time capability, the condition*

$$t_{ret} + t_{agg} \leq t_C$$

should be satisfied.

Obviously, to achieve this goal, some fine tuning should be performed by choosing the appropriate size of the chunks. Hence, continuous computation including small scale aggregation, pave the way for real-time capability.

In conclusion, within this section a formal model has been introduced in order to best describe the concepts of the continuous information processing strategy. The focus is on the terms of one-pass algorithm, small scale aggregation, continuous computing, and real-time capability. One-pass algorithms enable small scale aggregation, which can pave the way for real-time capability, on the condition that the timely constrains can be satisfied by the underlying computing environment. Actually, the one-pass requirement of the algorithms is not necessary, it suffices that the partial results of the computation of the chunks can be merged such that the expected aggregated values can be calculated.

V. Outline of the results; Discussions

Our objective has been to work towards developing practical solution to overcome the difficulties related to batch jobs, identified by Cisco in a white paper [9] as Pain Points. The pros and cons of the newly developed continuous information processing strategy versus the traditional batch jobs approach are outlined in this section and additional weak points of each technique are identified.

A. Cisco's Pain Points

1) Toughest challenge: The main challenge – which led to the outcome of this paper – was to investigate, whether it is possible to give satisfactory answers to the Pain Points raised by Cisco [9] concerning batch aggregation on data streams. Except Pain Point No. 3 regarding ad hoc reporting, to all other Pain Points, such as batch window time constraints, painful recovery, service-level agreements, etc., methods of resolution have been previously established [1]. In order to be able to properly present our methodology, a formal model has been set up and it has been shown that under some circumstances (for example, if the aggregation functions can be processed efficiently in one-step) the data collection and data aggregation can be performed continuously, and thus comprise real-time capability.

2) Sticking point – additional implementation effort: The one-pass implementation (or alternatively using small scale aggregation technology) of aggregation functions, can be meticulous, and may require additional effort. Most of the aggregation functions, also termed *measures*, used in the industry, permit such implementations; one of the well-known counterexample is the average absolute deviation. Since the computation is continuous and final results are available soon after the data collection has been completed, the Pain Point No. 1 regarding the question of batch window time constraints is obsolete.

3) Energy efficiency due to simplified recovery and to load distribution: Painful recovery (Paint Point No. 2) is less painful if there is a well thought-through recovery algorithm in place, such that only the erroneous parts are recalculated. Since there is a much better control of the computation/aggregation flow, a better service-level and resource conflict management can be achieved by using the continuous aggregation strategy.

It is true, that usually, batch jobs are performed during nighttime hours, when the workload on the computer is lower than during working hours. Unfortunately, due to computation errors or erroneous raw data, the batch aggregation has to be restarted also during normal working hours. Hence, the computer capacity should support the extended load due to recomputing the batch jobs during working hours. On the contrary, by using continuous computation, the load is distributed uniformly over the whole duration of the data collection and as a result, peak loads remain manageable. Moreover, due to our aggregation strategy – such that calculation is performed during the collection phase as early as possible, best when the data is still in memory – reloading the persisted data into memory is reduced to a minimum. Besides, the small scale aggregation can be optimised by identifying the optimal size of the chunks, such that the time constraints are met with minimal computational effort. This way, smaller computers can be used, especially since the energy efficiency of the batch aggregation is in general significantly worse than the correspondent computation regarding small scale aggregation.

B. Continuous aggregation versus batch jobs

1) Our fundamental computational strategy in a nutshell: According to the long time experience of the first author, the best performance in the field of Business Intelligence/Data Warehousing is obtained if the data is processed/transformed/precalculated as soon as possible; best, *as soon as the data is known to the system*. This includes also multiple storage strategies of the same raw/transformed data. Sometimes, it is advantageous to pursuit a *dual strategy*. On the one hand, try to follow the continuous computation strategy as long as possible i.e., as long as the implementation of the corresponding aggregation functions is possible with reasonable effort and run-time performance, and on the other hand, precalculate as much as possible by maintaining the batch jobs strategy.

2) Executions plans as the weak point of the batch jobs strategy: The main challenge of the batch jobs strategy, when using general purpose database management systems, is a technical one and it relates to the optimisation through *execution plans*. In highly simplified terms, the execution plans attempt to establish the most efficient execution of statements (queries) out of a summary of pre-calculated statistics. Unfortunately, the execution plans do not always generate the optimal (fastest, most efficient) query; performance can also degrade if the execution plans are updated. Hence, if the streams are not steady, performance degradation of the batch jobs may occur. There are methods to overcome the automatic generation of the execution plans, but the problem in principle remains.

On the contrary, by using small scale aggregation, the size of data sets on which computation is performed is more or less constant, and data is in memory, hence less prone to fluctuations due to the executions plans. It is therefore reasonable to assume some upper bounds, enabling real-time capability of the system.

C. Enhanced system modelling

One of the most important side benefits of the continuous information processing strategy is the straightforward system modelling. In this way, the design of the architecture, data flow, aggregation strategy, database schema design, etc., is given by the structure of the streaming data, the aggregation functions and the algorithms of their implementation. Thus, the more individualistic design, heavily based on the experience of the application developer is converted into a predefined set of well founded modelling strategies, sustaining a paradigm switch from more or less subjectively individualistic conceptions in software design and development towards objectively established optimal solutions. Quantitative estimations show that many Data Warehouse projects fail at a concerning rate, wasting all the time, money, and effort spent on them [45].

D. Broadening the tasks of the classical reporting strategy

1) Supporting production control: The essence of the continuous information processing strategy is that it enables the calculation of the aggregation functions during the collection phase. For example, for reporting purposes, the data for a full day is collected. The classical batch jobs strategy envisaged the generation of the data pool for reporting only after the data has been fully collected. Hence, calculated/aggregated values for reporting were available on the next day, depending on the execution time of the batch jobs. Thus, the scope of classical reporting strategy was to capture, survey and review the production status of the previous day. On the contrary, based on the data already collected, the continuous aggregation strategy enables the calculation/generation of preliminary reports at various points in time. This way, for example, soon after 12:00, the daily reports show the production figures corresponding to the time frame [0:00, 12:00]. Therefore, if these figures are not optimal, corresponding measures to boost production could be taken. Thus, modern reporting based on our technology enables *production control*.

2) Reducing ramp-up time: In some cases, optimisation can be substantial, saving time and costs. For example, in the semiconductor manufacturing, there are optional production steps, where the material is measured. The number of measurements can be in the range of hundreds, and the measurement time can last for several hours. The common aggregation technology assumes that all measurement data is collected before starting the aggregation. By adopting our continuous computation technology, preliminary measurement results can be calculated. This way, faulty processed material can be identified earlier, since there is no need for full calculation in order to identify faulty processing, and hence, the ramp-up time of a new product can be substantially reduced, thus giving the company decisive advantage over his competitors.

E. Additional issues

1) Hardware upgrade vs. performance improvement: Next, two issues are addressed, which are decisive from technical point of view:

1) absence of optimal Data Warehouse design methodology,
2) performance problems due to the high complexity, requirements on expandability, and the low scalability of the existing complex solutions.

According to the experience of the first author at Qimonda in the Business Intelligence and Data Warehouse environment, increasing the processing capability of the computers does not always lead to improved performance of the Data Warehouse applications. By doubling the computing capacity, roughly 20% in performance improvement has been achieved. Using high performance racks produced the best results. In the end, when the effort for performance improvement is greater than the effort to redesign the Data Warehouse, appropriate measures should be taken.

2) Improving data quality: Furthermore, due to our modular straightforward design strategy, the flow of data can be much closely monitored, hence superior *data quality* can be achieved. Moreover, enhanced *maintenance* and straightforward *implementation* can be reached due to the harmonised approach.

In conclusion, the price for achieving continuous aggregation may be high, the build in functions like standard deviation cannot be used any more, and as the case may be, new one-pass or similar algorithms for the aggregation functions have to be set up, hence algorithmic and programming effort may increase. The benefits are obvious, a straightforward design strategy, up-to-date aggregated values during data collection, a uniform computational effort over the data collection period and an efficient recalculation strategy, which lead in the end to a much efficient utilisation of computational resources. Improving the performance of batch jobs is tedious, if the redesign strategy is not an option, sophisticated data base technologies or costly high performance racks can overcome the problem.

VI. Conclusion and Future Work

In the following, the advantages of the CIPM are summarised and the future work, we are concerned with, is sketched.

A. Conclusion

Satisfactory solutions to the problems caused by the nightly batch aggregation – as pointed out by Cisco [9] – are given. To ensure an accurate presentation of our methodology, a formal model has been set up and it has been shown that for a specific type of aggregation functions – including those that support efficient one-pass implementation – the data aggregation can be performed continuously and thus allows real-time capability.

1) Advantages CIPM - Résumé: Continuous aggregation strategy:

1) supports *real-time capability*; if time constraints can be met,
2) supports *aggregated values corresponding to the captured data*; i.e., reporting capability at any point in time during data collection,
3) supports *enhanced production control* due to up-to-date aggregated data at any point in time during data collection,
4) supports *straightforward design strategies* due to clear, easy understandable architectural and implementation principles,

5) supports *easy maintenance* due to transparent and straightforward software development process,
6) supports *higher quality of aggregated data* due to the simplified architectural and implementation principles,
7) supports *uniform load of the underlying database system* due to the continuous aggregation principles,
8) supports *ad-hoc real-time reporting capability* due to the principles of the continuous aggregation as elaborated by us,
9) supports *higher degree of reporting flexibility than the MOLAP technology* due to the fact that the action of slicing and dicing is not bound to a hierarchical tree model,
10) avoids *complicated SQL-queries for data retrieval or data aggregation using large cartesian products* due to the new architectural principles,
11) supports more accurate *"Knowledge Discovery in Databases" capability* due to the much detailed data analysis capacity,
12) supports *early detection of erroneous data sets* due to the continuous aggregation principles, avoiding panic situations as in the case of nightly batch jobs,
13) supports *straightforward performance improvement strategies* due to the simplified architectural principles,
14) *avoids or reduces "hot working phases" at night for the IT personnel* due to the absence of nightly jobs,
15) reduces *the risk of incomplete or missing standard reporting* – "race against time", as termed by Cisco – due to the enlarged aggregation window,
16) reduces *the complexity of the database administration* due to the new streamlined environment versus a database pushed to its limits, as in case of batch jobs,
17) supports improved *load balancing* due the modularised aggregation strategy,
18) supports *lower memory requirements*, since the data to be aggregated is already uploaded into memory during the data collection phase,
19) supports *streamlined SQL-statements*, since the data to be aggregated is preserved in small chunks,
20) avoids *regular performance improvement tasks* due to the streamlined architecture of the aggregation strategy,
21) avoids *performance bottlenecks* due to the recalculation of the nightly aggregation during business hours,
22) ensures *very good scalability, both for the small scale aggregation and for the ad-hoc reporting* due to much better performance control,
23) avoids *performance fluctuation* due to imperfect execution plans,
24) supports *efficient recalculation of aggregated values*, in case erroneous data is collected,
25) supports *parallelisation* beyond the built-in facilities of the underlying database system,
26) supports *independency of the underlying database system*, such as Relational, No-SQL, etc., and last but not least:
27) ensures *energy efficiency*, since smaller hardware can be used due to the fact that aggregation is performed during the whole data collection period.

2) Difficulties CIPM - Résumé:

1) some *predefined formulas, e.g., STDEV, cannot be used directly*, since the function above is not cumulative,
2) *difficult architectural set-up*, i.e., new algorithms have to be designed and implemented,
3) *longer development times* due to the new architectural design strategy,
4) *IT staff has to be additionally trained* due to unconventional architectural and maintenance strategies,
5) *heterogeneous team including mathematicians and data scientist* need to be used, i.e., the algorithmic part of the development may be sophisticated,
6) *increased development costs* due to the unconventional development strategies, and last but not least:
7) *strong management commitment to overcome the difficulties* due to the anticipating challenges.

We have not experienced major difficulties in implementing the CIPM approach in database applications, implementation from the scratch is pretty straightforward, porting to CIPM an existing legacy database application using batch jobs, can be quite cumbersome. For sophisticated legacy applications, the most efficient method is to try to improve performance as long as possible by applying database technologies, and/or by using high performance racks, etc.

3) Final considerations: The price of the advantages as above depends on the structure of the aggregation function. Most of the key performance indicators used in the industry permit an incremental representation, i.e., as functions of different representation of sum(), avg(), count(), similar to the standard deviation, as presented in this article. The effort in this cases is manageable. Generally speaking, the aggregation functions should permit efficient one-pass calculation, or in the case of holistic functions, an algorithm, such that the time constraints can be satisfied.

In conclusion, for small applications, where real-time constraints are not an issue, batch jobs (large scale aggregation) will deliver satisfactory results, whereas for large applications, even if real-time capabilities are not required, the advantages of CIPM may prevail.

B. Future Work

The Pain Point No. 3 of Cisco's white paper [9]: "ad hoc reporting; managing unplanned reports in a plan-based environment", has been handled in this paper. The question still remains, as to what extent "unforeseen" reports can be meaningfully set up. Furthermore, one asks oneself, what is the optimal strategy regarding volatile versus persistent aggregation, i.e., aggregation within a query set up by a visualisation tool versus aggregation persisted in a data storage. From an algorithmic perspective, persistent aggregation offers more advantages if the query is often invoked. Moreover, the results can be much better validated if the data is persisted. On the other hand, sporadic queries should remain volatile, i.e., the result of the queries should not be persisted for further reuse.

Tangwongsan [17] points out that "much less is known for nontrivial scenarios", i.e., "functions that are not associative and do not support FIFO windows". Within this paper satisfactory answers are given for those aggregation functions that can be reduced to additive functions, and/or allow one-pass algorithm. For example, it has been shown that the median [8] can successfully be used within CIPM.

Generally speaking, the integration of the holistic functions within CIPM remains an outstanding challenge. Commonly, for data warehouse applications, the raw data is always stored for a certain period of time in order to be able to retrace the computation. Hence, improving the speed of the computation by using distributed algorithms, etc., can ensure real-time capabilities. On the contrary, when dealing with long time and high amount of streaming data, storing the data even temporarily is not possible. In such cases, developing suitable algorithms for approximate calculations could help. Each holistic function should be handled on a case-by-case basis. A general strategy seems improbable for the time being. In conclusion, in general, for non-holistic functions (including the usual KPIs used in business and industry) satisfactory results can be given. The problem regarding the holistic functions is still open, some of them can be used under some circumstances in real-time environment.

Acknowledgment

We appreciate the information and insightful comments provided by the anonymous reviewers, which significantly improved the quality of this paper. Last, but not least, we acknowledge the assistance and helpful comments provided by Prof. Dini.

References

[1] M. Zinner, K. Feldhoff, and W. E. Nagel, "Continuous Information Processing Enabling Real-Time Capabilities: An Energy Efficient Big Data Approach," *ICSEA 2021 : The Sixteenth International Conference on Software Engineering Advances*, pp. 155–165, 2021, Retrieved: June 2022. [Online]. Available: https://www.thinkmind.org/articles/icsea_2021_2_180_10095.pdf

[2] A. De Mauro, M. Greco, and M. Grimaldi, "A formal definition of big data based on its essential features," *Library Review*, 2016, Retrieved: June 2022. [Online]. Available: https://www.researchgate.net/publication/299379163_A_formal_definition_of_Big_Data_based_on_its_essential_features

[3] H. U. Buhl, M. Röglinger, F. Moser, and J. Heidemann, "Big Data," *Business & Information Systems Engineering*, vol. 5, no. 2, pp. 65–69, 2013, Retrieved: June 2022. [Online]. Available: https://doi.org/10.1007/s12599-013-0249-5

[4] Statista, "Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2025," 2021, Retrieved: June 2022. [Online]. Available: https://www.statista.com/statistics/871513/worldwide-data-created/

[5] R. Sousa, R. Miranda, A. Moreira, C. Alves, N. Lori, and J. Machado, "Software tools for conducting real-time information processing and visualization in industry: An up-to-date review," *Applied Sciences*, vol. 11, no. 11, p. 4800, 2021, Retrieved: June 2022. [Online]. Available: https://www.mdpi.com/2076-3417/11/11/4800

[6] K. Yasumoto, H. Yamaguchi, and H. Shigeno, "Survey of real-time processing technologies of iot data streams," *Journal of Information Processing*, vol. 24, no. 2, pp. 195–202, 2016, Retrieved: June 2022. [Online]. Available: https://doi.org/10.2197/ipsjjip.24.195

[7] I. Sommerville, "Software engineering 9th edition," *ISBN-10*, vol. 137035152, p. 18, 2011.

[8] Zinner *et al.*, "Real-time information systems and methodology based on continuous homomorphic processing in linear information spaces," 2015, Retrieved: June 2022. [Online]. Available: https://patentimages.storage.googleapis.com/ed/fa/37/6069417bdcc3eb/US20170032016A1.pdf

[9] Cisco, "BI and ETL Process Management Pain Points," *White Paper*, pp. 1–9, 2010, Retrieved: June 2022. [Online]. Available: http://download.101com.com/tdwi/ww29/cisco_bi_etl_process_management_pain_points.pdf

[10] N. Schweikardt, "One-pass algorithm." 2009, Retrieved: June 2022. [Online]. Available: http://www.tks.informatik.uni-frankfurt.de/schweika/downloads/EncycDBS_OnePassAlgos.pdf

[11] P. E. O'Neil, "The sb-tree an index-sequential structure for high-performance sequential access," *Acta Informatica*, vol. 29, no. 3, pp. 241–265, 1992, Retrieved: June 2022. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.55.9482&rep=rep1&type=pdf

[12] J. Zhang, "Spatio-temporal aggregation over streaming geospatial data," in *Proceedings of the 10th International Conference on Extending Database Technology Ph. D. Workshop*. Citeseer, 2006, Retrieved: June 2022. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.8746&rep=rep1&type=pdf

[13] J. Yang and J. Widom, "Incremental computation and maintenance of temporal aggregates," *The VLDB Journal*, vol. 12, no. 3, pp. 262–283, 2003, Retrieved: June 2022. [Online]. Available: http://ilpubs.stanford.edu:8090/482/1/2000-6.pdf

[14] TU-Berlin-DIMA, "Scotty: Efficient window aggregation for out-of-order stream processing," *Generated by GitHub Pages*, 2021, Retrieved: June 2022. [Online]. Available: https:21:3103.09.202121:3103.09.202121:3103.09.202121:3103.09.202121:3103.09.2021//tu-berlin-dima.github.io/scotty-window-processor/

[15] J. Traub, P. M. Grulich, A. R. Cuellar, S. Breß, A. Katsifodimos, T. Rabl, and V. Markl, "Scotty: Efficient window aggregation for out-of-order stream processing," in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 2018, pp. 1300–1303, Retrieved: June 2022. [Online]. Available: https://hpi.de/fileadmin/user_upload/fachgebiete/rabl/publications/2018/ScottyICDE2018.pdf

[16] J. Traub, P. M. Grulich, A. R. Cuéllar, S. Breß, A. Katsifodimos, T. Rabl, and V. Markl, "Scotty: General and efficient open-source window aggregation for stream processing systems," *ACM Transactions on Database Systems (TODS)*, vol. 46, no. 1, pp. 1–46, 2021, Retrieved: June 2022. [Online]. Available: https://www.redaktion.tu-berlin.de/fileadmin/fg131/Publikation/Papers/Traub_TODS-21-Scotty_preprint.pdf

[17] K. Tangwongsan, M. Hirzel, and S. Schneider, "Sliding-window aggregation algorithms." 2019, Retrieved: June 2022. [Online]. Available: http://hirzels.com/martin/papers/encyc18-sliding-window.pdf

[18] ——, "Optimal and general out-of-order sliding-window aggregation," *Proceedings of the VLDB Endowment*, vol. 12, no. 10, pp. 1167–1180, 2019, Retrieved: June 2022. [Online]. Available: https://www.scott-a-s.com/files/vldb2019_fiba.pdf

[19] Z. Chen and A. Zhang, "A survey of approximate quantile computation on large-scale data," *IEEE Access*, vol. 8, pp. 34 585–34 597, 2020, Retrieved: June 2022. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9001104

[20] P. P. Pebay, T. Terriberry, H. Kolla, and J. C. Bennett, "Formulas for robust, parallel computation of arbitrary-order, arbitrary-variate, statistical moments with arbitrary weights and compounding." Sandia National Lab.(SNL-CA), Livermore, CA (United States); The Xiph. Org , Tech. Rep., 2015, Retrieved: June 2022. [Online]. Available: https://www.osti.gov/servlets/purl/1504207

[21] P. Pébay, T. B. Terriberry, H. Kolla, and J. Bennett, "Numerically stable, scalable formulas for parallel and online computation of higher-order multivariate central moments with arbitrary weights," *Computational Statistics*, vol. 31, no. 4, pp. 1305–1325, 2016, Retrieved: June 2022. [Online]. Available: https://www.osti.gov/servlets/purl/1426900

[22] C. Labreuche, "A formal justification of a simple aggregation function based on criteria and rank weights," in *Proc. DA2PL2018, From Multiple Criteria Decis. Aid Preference Learn.*, 2018, pp. 1–1, Retrieved: June 2022. [Online]. Available: http://da2pl.cs.put.poznan.pl/programme/detailed-programme/da2pl2018-abstract-14.pdf

[23] R. Eccles and G. Serafeim, "Corporate and integrated reporting: A functional perspective,[w:] corporate stewardship: Achieving sustainable effectiveness, red," *E. Lawler, S. Mohrman, J. OToole, Greenleaf*, Posted:

2 Feb 2014 Last revised: 24 May 2018, Retrieved: June 2022. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2388716

[24] T. F. Chan, G. H. Golub, and R. J. LeVeque, "Algorithms for computing the sample variance: Analysis and recommendations," *The American Statistician*, vol. 37, no. 3, pp. 242–247, 1983, Retrieved: June 2022. [Online]. Available: http://www.cs.yale.edu/publications/techreports/tr222.pdf

[25] ——, "Updating formulae and a pairwise algorithm for computing sample variances," in *COMPSTAT 1982 5th Symposium held at Toulouse 1982*, 1982, pp. 30–41, Retrieved: June 2022. [Online]. Available: https://apps.dtic.mil/sti/pdfs/ADA083170.pdfP

[26] W. Hopp and M. Spearman, *Factory Physics: Third Edition*. Waveland Press, 2011.

[27] W. Hansch and T. Kubot, "Factory Dynamics Chapter 7 Lectures at the Universitaet der Bundeswehr Muenich," p. 68, Retrieved: June 2022. [Online]. Available: https://fac.ksu.edu.sa/sites/default/files/Factory%20Dynamics.pdf

[28] C.-F. Lindberg, S. Tan, J. Yan, and F. Starfelt, "Key performance indicators improve industrial performance," *Energy procedia*, vol. 75, pp. 1785–1790, 2015, Retrieved: June 2022. [Online]. Available: https://doi.org/10.1016/j.egypro.2015.07.474

[29] M. Zinner *et al.*, "Techniques and Methodologies for Measuring and Increasing the Quality of Services: a Case Study Based on Data Centers," *International Journal On Advances in Intelligent Systems, volume 13, numbers 1 and 2, 2020*, vol. 13, no. 1 & 2, pp. 19–35, 2020, Retrieved: June 2022. [Online]. Available: http://www.thinkmind.org/articles/intsys_v13_n12_2020_2.pdf

[30] W. Kahan, "Pracniques: further remarks on reducing truncation errors," *Communications of the ACM*, vol. 8, no. 1, p. 40, 1965.

[31] T. Pham-Gia and T. Hung, "The mean and median absolute deviations," *Mathematical and Computer Modelling*, vol. 34, no. 7-8, pp. 921–936, 2001, Retrieved: June 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0895717701001091

[32] R. C. Geary, "The ratio of the mean deviation to the standard deviation as a test of normality," *Biometrika*, vol. 27, no. 3/4, pp. 310–332, 1935.

[33] J. K. Patel and C. B. Read, *Handbook of the normal distribution*. CRC Press, 1996, vol. 150.

[34] G. Laipple, S. Dauzère-Pérès, T. Ponsignon, and P. Vialletelle, "Generic data model for semiconductor manufacturing supply chains," in *2018 Winter Simulation Conference (WSC)*. IEEE, 2018, pp. 3615–3626, retrieved: June 2022. [Online]. Available: http://simulation.su/uploads/files/default/2018-laipple-dauzere-peres-ponsignon-vialletelle.pdf

[35] F. Biebl, R. Glawar, A. Jalali, F. Ansari, B. Haslhofer, P. de Boer, and W. Sihn, "A conceptual model to enable prescriptive maintenance for etching equipment in semiconductor manufacturing," *Procedia CIRP*, vol. 88, pp. 64–69, 2020, retrieved: June 2022. [Online]. Available: https://doi.org/10.1016/j.procir.2020.05.012

[36] K. Hilsenbeck, "Optimierungsmodelle in der Halbleiterproduktions-technik," Ph.D. dissertation, Technische Universität München, 2005, retrieved: June 2022. [Online]. Available: http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss20050808-1721087898

[37] J. L. Barlow, "Error analysis of a pairwise summation algorithm to compute the sample variance," *Numerische Mathematik*, vol. 58, no. 1, pp. 583–590, 1990, Retrieved: June 2022. [Online]. Available: https://de.booksc.eu/book/6543977/98912d

[38] GSA, "Cognos Ad-Hoc Reporting (Basics)," *ePM Quick Reference Guide 75*, 2016, Retrieved: June 2022. [Online]. Available: https://www.gsa.gov/cdnstatic/QRG.075_Ad_Hoc_Reporting_6.0.pdf

[39] J. Lewis and T. Disney, "Large limits to software estimation," *ACM Software Engineering Notes*, vol. 26, no. 4, pp. 54–59, 2001, Retrieved: June 2022. [Online]. Available: http://scribblethink.org/Work/Softestim/kcsest.pdf

[40] J. Lewis, "Mathematical limits to software estimation: Supplementary material," *Stanford University*, 2001, Retrieved: June 2022. [Online]. Available: http://scribblethink.org/Work/Softestim/softestim.html

[41] W. H. Roetzheim and R. A. Beasley, *Software project cost schedule estimating: best practices*. Prentice-Hall, Inc., 1998.

[42] B. Evgeniy, "Supercomputer beg with artificial intelligence of optimal resource use and management by continuous processing of large programs," *Glob Acad J Econ Buss*, vol. 1, pp. 21–26, 2019, Retrieved: June 2022. [Online]. Available: https://gajrc.com/media/articles/GAJEB_11_21-26_zOIbTWD.pdf

[43] E. A. Lee, "What is real time computing? a personal view." *IEEE Des. Test*, vol. 35, no. 2, pp. 64–72, 2018, Retrieved: June 2022. [Online]. Available: https://ptolemy.berkeley.edu/projects/chess/pubs/1192/Lee_WhatIsRealTime_Accepted.pdf

[44] TimeSys Corporation, "The concise handbook of real-time systems," *TimeSys Corporation Pittsburgh, PA, Version 1.3*, pp. 1–65, 2002, Retrieved: June 2022. [Online]. Available: https://course.ece.cmu.edu/~ece749/docs/RTSHandbook.pdf

[45] D. Asrani, R. Jain, and U. Saxena, "Data Warehouse Development Standardization Framework (DWDSF): A Way to Handle Data Warehouse Failure," *IOSR Journal of Computer Engineering (IOSR-JCE)*, vol. 19, pp. 29–38, 2017, Retrieved: June 2022. [Online]. Available: http://www.iosrjournals.org/iosr-jce/papers/Vol19-issue1/Version-2/E1901022938.pdf