Continuous Information Processing Enabling Real-Time Capabilities: An Energy Efficient Big Data Approach

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

***Abstract*—A considerable part of data aggregation during information processing in industry is still carried out in nightly batch mode. In contrast, using our method termed Continuous Information Processing Methodology (CIPM), the aggregation can be started as soon as the data collection is initiated. Our method was motivated by a real-world application scenario at a semiconductor company. During the data collection process, partial aggregated values are determined, such that, after the data collection phase has been completed, the final aggregated values are available for evaluation. In order to benefit from the rigour of a formal approach, a mathematical model is introduced and the conversion from batch mode to CIPM is exemplified. The most common aggregation functions used in various field of industry and business can be easily adapted and used within CIPM. The major additional benefits of the CIPM are reduced and spread aggregational effort over the whole collection period as well as 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 capabilities; Data Analytics; Data processing; Stream processing; Batch processing; Business Intelligence; Big Data.**

I. Introduction

At first, the core of our aggregation theory in a nutshell is succinctly addressed, some definitions such as Big Data are tightened up and the principal motivation of our paper, i.e., increased real-time requirements in the industry, is presented. Cisco, in a white paper, identified the deficiencies of the classical nightly batch jobs aggregation strategy and summarised them in five Pain Points. All except for Pain Point 3 regarding ad-hoc reporting, will be addressed within this paper. The statistical function *standard deviation* is used to illustrate our 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 methodology 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 capabilities 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.

According to [1] "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 [1], involving the company's entire population, its members to "efficiently manage data properly and incorporate them into decision making processes" [2].

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 [3]. Real-time information processing has become a significant requirement for the optimal functioning of the manufacturing plants [4]. Worldwide by 2022, over 50 billion *Internet of Things* (IoT) devices including sensors and actuators are predicted to be installed in machines, humans, vehicles, buildings, and environments.

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 [5]. 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 [6]. In a real-time system deadlines must always be met, regardless of the system load. A system not specified as real-time cannot usually 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.

 ISBN: 978-1-61208-894-5

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 [7].

B. Aim

Cisco [8] 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 will address all points except Pain Point 3), which is subject for future research.

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 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 how the batch aggregation can be transformed into continuous aggregation. The presentation of the main results and discussions based upon these results constitute the content of Section V, whereas Section VI concludes this paper and sketches the future work.

II. Related work

The focus of this Section is primarily on algorithmic approaches regarding the previous art. The analysis of different one-pass algorithms [9] 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 [10], [11]. Zhang [11] outlines the key challenges of spatio-temporal aggregate computation on geo-spatial image data, focusing primarily on data having the form of raster images. She gives a very detailed overview of the state of the art regarding efficient aggregate computation. Her 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 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 [11], [12].

B. Scotty

Scotty [13] 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 [14]. 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 [15]. 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. 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 [16] 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. 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. [17] 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 is the middle number in an ordered list of items. The median is a holistic function, i.e., its result has to rely on the entire input set, so that there is no constant bound on the

 ISBN: 978-1-61208-894-5

size of the storage needed for the computation. An algorithm suitable for continuous aggregation based on heap technology can be found in [7]. For the sake of completeness, the main idea is presented below. 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 total number of items is even and if the case arises, the heaps are balanced against each other, such that the two heaps contain the same number of items, etc. Hence, holistic aggregation functions can be used with continuous aggregation techniques, they should however satisfy the foreseen time constraints.

D. Quantile

A survey of approximate quantile computation on large-scale data is given by Chen [18]. In streaming models, where data elements arrive one by one in a streaming way, 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 [19], [20], 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 [16] 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 [7] can be used with CIPM.

III. Problem Description

The term information function and aggregation function [21] are used synonymously within this paper. 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 [22]. 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.

1) Overview of the CIPM : Following, the fundamental issues of the continuous aggregation strategy are outlined by using two simple flow diagrams, Figure (1) presenting the classical nightly jobs aggregation strategy, whereas Figure (2) describing very succinctly the continuous aggregation strategy. It is assumed, that reporting is based on daily aggregated data. Data collection starts at 00:00:00 for the current day and it ends, retrieving data generated till 23:59:59 of the same day. Whenever applying the classical batch jobs strategy, the transformation/aggregation is started only after the data is fully retrieved/collected for the current day, which we are referring to. In this case, the transformation/aggregation can be started only after midnight.

On the other hand, the CIPM is carried out on small chunks of data, 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. While the second chunk C_2 is retrieved, the first chunk C_1 is transformed/aggregated, and so on and so forth. At the end of the current day, most of the collected data is aggregated. 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, the aggregated values are ready for reporting soon after midnight.

In order to keep the presentation simple and accessible and to avoid technical complications, it is required 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 phase of aggregation 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.

In order to keep the presentation simple, it is assumed within this paper, that the chunks are of the same size and the time to retrieve them does not change. Of course, this assumption in not necessary in real-world systems.

2) Exemplification using Standard Deviation (SD): Next, the complexity of our approach is illustrated by exemplifying the technology on the *standard deviation of the sample*. The standard deviation shows how much variation (dispersion, spread) from the mean exists. It is easy to present 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

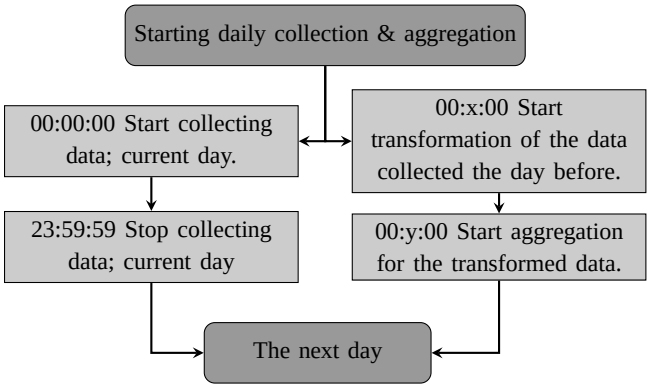

Figure 1: Simplified flow diagram exemplifying the batch job strategy(x is the time gap due to collection delay; y is the time gap due merely to transformation).

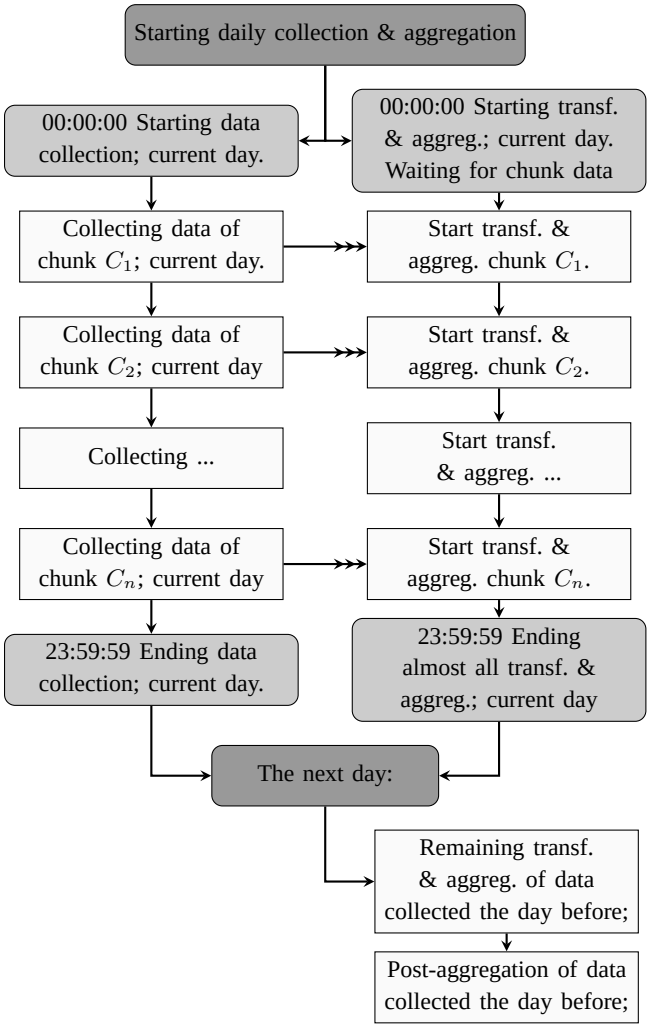

Figure 2: Simplified flow diagram exemplifying the continuous aggregation strategy. The arrow with three heads signifies that the aggregation phase waits till the respective chunk data has been collected.

sample) standard deviation is:

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

At first glimpse, the above representation of the standard deviation cannot be applied to 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. [23], [24] call 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:

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

Let $1 \leq n \leq N$. Let $S_n = n\sum_{i=1}^{n}x_i^2 - \Big(\sum_{i=1}^{n}x_i\Big)^2$, let $A_n := \sum_{i=1}^{n}x_i^2$, let $B_n := \sum_{i=1}^{n}x_i$. The alternative representation (2) of the standard deviation is suitable to be used within the continuous computation approach.

During the data collection phase, the functions A_n and B_n are updated, either after each item x_n, as soon as it in 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 S_n, A_n, B_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 data collection.

For example, almost all *Key Performance Indicators* (KPIs) used in the semiconductor industry can be adapted to be applied within CIPM [25]–[28]. The same is true in other areas of the industry or business.

3) Reason 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 [29]. Please note that the representation (2) using absolute values, has been adapted in order to avoid negative values under the square root. Using double precision arithmetic can possibly avoid the occurrence of anomalies as above.

 ISBN: 978-1-61208-894-5

4) 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 [30]–[32].

IV. The Formal Model

The description of our methodology is formalised, we introduce a mathematical model in order to use the advantages of the rigour of a formal approach over the inaccuracies and the incompleteness of natural languages. It is assumed that the streams are *finite*, i.e., there are two points in time, t_s, the initiating and t_e, the termination point, such that within this time interval, the data is collected and aggregated.

If the aggregation occurs after the entire raw data have been previously collected, involving technologies that process all of the collected data at once, then the process is termed *batch aggregation mode* or *large scale aggregation*. On the contrary, if the collected data $[t_s, t_e]$ can be split into $k \geq 2$ smaller (equal) units of length l, such that

$$U_1 := [t_s, t_{s+l}],\ U_2 := [t_{s+l+1}, t_{s+2l}], \ldots,$$
$$U_k := [t_{s+(k-1)\cdot l+1}, t_e]$$

also termed chunks and *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*.

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

A. Notation

1) General notations: 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. In order to keep our model simple, it is supposed that each stream delivers data at the same point in time, let $\{1, 2, \ldots, T\}$ be the points in time when the data is collected and known by the system.

Let $1 \leq t \leq T$. The value of the stream $X^{(l)}$ collected at time t is denoted by $x_t^{(l)}$. The streamed values can be represented as a matrix

$$(x_{tl})_{1\leq t\leq T; 1\leq l\leq l_X}.$$

2) Grouping: Let l_F 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

$$G := \{G^{(1)}, G^{(2)}, \ldots, G^{(l_G)}\}$$

is defined on the space of the streams, such that

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

and $l_F = l_G$. Accordingly:

$$g_t^{(l)} := x_t^{(l_1)} \times x_t^{(l_2)} \times \cdots \times x_t^{(l_l)}$$

is the value of the grouping $G^{(l)}$ at time t. This way, new compound streams are created. 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.

3) Aggregation functions: Let

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

be the set of the aggregation functions, such that

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

for $1 \leq l \leq l_F$.

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 computed/aggregated in small batches. In order to be able to continuously compute – i.e., retrieve/collect, transform, aggregate – the time to aggregate the small batch should not exceed the collection time of the same batch. Obviously, if this is not the case, the aggregation cannot be performed during the data collection phase. During the transformation phase, the data is verified for accuracy, consolidated/aligned (i.e., data from multiple sources is harmonised), grouped and adapted such that it is best finalised for aggregation. During the transformation phase data is not altered and has the level of granularity of the original raw data.

4) Standard deviation as exemplification: Let $1 \leq l \leq l_F$. Let us suppose that $F^{(l)} : G^{(l)} \to \mathbb{R}$ is the standard deviation, see representation (2) and let $x \in G^{(l)}$ a particular stream. Let $1 \leq t \leq T$ and let:

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

Let $F_t^{(l)}(x)$ be the value of the function $F^{(l)}$ applied on the values subscripted by $\{1, 2, \ldots, t\}$.

Then $F_t^{(l)}(x)$ 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:

$$F_t^{(l)}(x) = \frac{1}{t}\sqrt{\left|f_t^{(l,3)}(x)\right|}. \quad (4)$$

B. Information processing

1) Chunk-wise processing: Let $j, q \geq 1$ 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 $C_j^{(l)}$ consists of partial streams

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

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. As already mentioned, in order to assure continuous computation, the time to perform the aggregation on the chunks should not exceed the retrieval time of a chunk, else the aggregation cannot be performed during the retrieval period. The values of $f_{(j+1)q}^{(l,1)}$ and $f_{(j+1)q}^{(l,2)}$ can be easily calculated out of $f_{jq}^{(l,1)}$ and $f_{jq}^{(l,2)}$, for example:

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

The value $F_{jq}^{(l)}$ can be calculated at each step, or alternatively, after having reached the end of the collection phase. This phase is termed *post aggregation phase*, since calculations are not done during the small scale aggregation phase (i.e., chunk aggregation), but after all chunks have been retrieved and aggregated. Since the small scale aggregation should be as fast and effective as possible, the functions $f_{jq}^{(l,3)}, F_{jq}^{(l)}$ must not be necessary calculated for each chunk, – 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.

2) 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 $f_t^{(l,1)}$ and/or $f_t^{(l,2)}$ for $1 \leq t \leq T$ are evaluated in the straightforward way [29]. A greater accuracy can be achieved by simply shifting some of the calculation to double precision, see [23], [24] for a discussion on rounding errors and the stability of presented algorithms. Barlow presents an *one-pass-through* algorithm [33], which is numerically stable and which is also suitable for parallel computing.

The scope of the presentation above 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 approximative one-pass implementation of the algorithms.

3) General case: Now, let us consider the general case. 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

$F^{(l)} : G^{(l)} \to \mathbb{R}$ such that:

a) there exists l_f real valued functions

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

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

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

can be calculated out of the values of $f_{jq}^{(l,i)}$,

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 $F_{(j+1)q}^{(l)}$ can be calculated out of $f_{(j+1)q}^{(l,i)}$.

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

4) Reprocessing: In practical systems, in general, there should be a technology in place that allows recalculation. This is necessary, if for what 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 $df_t^{(l,1)}, df_t^{(l,2)}$ can be introduced, such that

$$df_{jq}^{(l,1)}(x) := \sum_{i=j\cdot q}^{(j+1)\cdot q} x_i^2$$

and

$$df_{jq}^{(l,2)}(x) := \sum_{i=j\cdot q}^{(j+1)\cdot q} x_i.$$

Then, correct and updated computed values can be achieved, for example by adding to $f_T^{(l,3)}$ the new value of $df_{jq}^{(l,1)}$ and subtracting the corresponding old value $df_{jq}^{(l,1)}$, similar considerations for $df_{jq}^{(l,2)}$. 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 $F_{j_f \cdot q}^{(l)}$ 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.

C. Pseudo-code algorithm exemplification

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

 ISBN: 978-1-61208-894-5

```
/*
* Sample code to exemplify the continuous aggregation
     strategy
*/
double precision f^(l,1)= 0;//component function
double precision f^(l,2)= 0;//component function
double precision F^(l) = 0;//intermediary value of the
     standard deviation corresponding to the state of
     collection
int l_chunk = 10,000;//number of the items of a chunk
float[l_chunk] c;//contains the retrieved values of the chunk
float[l_chunk] c_prev;//contains the data of previous chunk
int L_col = 0;//length of the collection
//-------------------------------------------------------
/*
* data corresponding to the length of a chunk is collected
*/
procedure data_collection(){
   int l_cur = 0;//number of the collected items
   repeat
      collect data into c;
      l_cur + +;
   until (l_cur = l_chunk)
   for (int i; i < l_chunk; i++){
      c_prev[i] = c[i];//copy the values of c into c_prev
   }
};
//-------------------------------------------------------
/*
* data of the previous chunk is aggregated
*/
procedure data aggregation(){
   float[l_chunk] x; // contains data of the previous chunk
   for (int i; i < l_chunk; i++){
      x[i] = c_prev[i];//copy the values of c_prev into x
   }
   // calculation of the functions composing the standard
        deviation
   f^(l,1) := f^(l,1) + sum_{i=1}^{l_chunk} (x[i])^2;
   f^(l,2) := f^(l,2) + sum_{i=1}^{l_chunk} x[i];
   L_col := L_col + l_chunk;//number of items already collected
   //
   F^(l) := 1/L_col sqrt(|L_col · f^(l,1) − (f^(l,2))^2|); // only if required
};
//-------------------------------------------------------
/*
* final calculation of the standard deviation
*/
procedure data_post_aggregation(){

   F^(l) := 1/L_col sqrt(|L_col · f^(l,1) − (f^(l,2))^2|);

};
//-------------------------------------------------------
/*
* start aggregation in parallel to data collection
*/
void main(){
   data_collection();
   repeat
      start: in parallel
         thread: data_collection();
         thread: data aggregation();
      wait until both threads finished;
   until collection_phase_has_ended;
   data aggregation();
   data_post_aggregation();
}
```

Figure 3: Pseudo-code based algorithm using standard deviation exemplifying the continuous aggregation strategy.

D. 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 [34], [35] 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 [36] 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.

E. Real-time capabilities

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 algorithmics problem [37].

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 [38]. Moreover, TimeSys [39] 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 capabilities 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 extent of the streams, i.e., 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.

The strategy to achieve real-time behaviour based on continuous stream computing is straightforward. 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 capabilities, 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 for the chunks. Hence, continuous computation including small scale aggregation, pave the way for real-time capabilities.

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 capabilities, 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 [8] 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 [8] 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 established. In order to be able to properly present our methodology, a formal model is set up and it is 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 capabilities.

2) Sticking point – additional implementation effort: The one-pass implementation (alternatively using small scale 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 computational/aggregational flow, a better service-level and resource conflict management can be achieved by using continuous aggregation. 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 recomputed 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 due to 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 Warehouse 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 optimization 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 capabilities of the system.

C. Enhanced system modeling

One of the most important side benefits of the continuous information processing strategy is the straightforward system modeling. 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 modeling 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 [40].

D. Performance advantages

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

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

According to the experience of the first author at Qimonda in the Business Intelligence and Data Warehouse environment, increasing the processing capabilities of the computers does not always lead to improved performance of the Data Warehouse application. 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. Furthermore, due to our modular straightforward design strategy, the flow of data can be much closely monitored, hence superior *data quality* can be achieved.

2) Broadening the tasks of the classical reporting strategy: 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 can be taken. Thus, modern reporting based on our technology enables *production control*.

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 computation. By adopting our continuous computation technology, preliminary measurement results can be calculated. This way, faulty processed material can be identified earlier and the ramp-up time of a new product can be substantially reduced, thus giving the company decisive advantage over his competitors.

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 rack high performance can help.

VI. Conclusion and Future Work

In the following, the advantages of the CIPM are summarized 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 [8] are given, except for *Pain Point* No. 3. 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 supports efficient one-pass implementation – the data aggregation can be performed continuously and thus allows real-time capabilities.

1) Advantages CIPM - Résumé: Continuous aggregation strategy supports:

1) *real-time capabilities*; if time constraints can be met,
2) *aggregated values corresponding to the captured data*; i.e., reporting capabilities at any point in time during data collection,
3) *enhanced production control* due to up-to-date aggregated data at any point in time during data collection,
4) *straightforward design strategies* due to clear, easy understandable architectural and implementation principles,
5) *easy maintenance* due to transparent and straightforward software development process,
6) *higher quality of aggregated data* due to the simplified architectural and implementation principles,
7) *uniform load of the underlying database system* due to the continuous aggregation principles,
8) *efficient recalculation of aggregated values*; in case erroneous data is collected, and last but not least:
9) *energy efficiency*; smaller computers can be used due to the fact that aggregation is performed during the whole data collection period.

We have not experienced major difficulties in implementing the CIPM approach in database application, 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, by using high performance racks, etc.

2) 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 the standard deviation, as presented in this article. The effort in this cases is manageable. Generally speaking, the aggregation function should permit efficient one-pass calculation, or in the case of holistic functions, an algorithm, such that the time constraints can be satisfied.

B. Future Work

The Pain Point No. 3 of Cisco's white paper [8]: "ad hoc reporting; managing unplanned reports in a plan-based environment", remains still unhandled and it is subject of future research. The question arise, as to what extent such unplanned reports can be meaningfully foreseen and 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.

ACKNOWLEDGMENT

We acknowledge the assistance and helpful comments of the anonymous referees.

REFERENCES

[1] A. De Mauro, M. Greco, and M. Grimaldi, "A formal definition of big data based on its essential features," *Library Review*, 2016, Retrieved: September 2021. [Online]. Available: https://www.researchgate.net/publication/299379163_A_formal_definition_of_Big_Data_based_on_its_essential_features

[2] 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: September 2021. [Online]. Available: https://doi.org/10.1007/s12599-013-0249-5

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

[4] 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: September 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/11/4800

[5] 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: September 2021. [Online]. Available: https://doi.org/10.2197/ipsjjip.24.195

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

[7] Zinner *et al.*, "Real-time information systems and methodology based on continuous homomorphic processing in linear information spaces," 2015, Retrieved: September 2021. [Online]. Available: https://patentimages.storage.googleapis.com/ed/fa/37/6069417bdcc3eb/US20170032016A1.pdf

[8] Cisco, "BI and ETL Process Management Pain Points," *White Paper*, pp. 1–9, 2010, Retrieved: September 2021. [Online]. Available: https://www.cisco.com/c/dam/en/us/products/collateral/cloud-systems-management/tidal-enterprise-scheduler/whitepaper_c11-633329.pdf

[9] N. Schweikardt, "One-pass algorithm." 2009, Retrieved: September 2021. [Online]. Available: http://www.tks.informatik.uni-frankfurt.de/schweika/downloads/EncycDBS_OnePassAlgos.pdf

[10] 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: September 2021. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.55.9482&rep=rep1&type=pdf

[11] 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: September 2021. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.8746&rep=rep1&type=pdf

[12] J. Yang and J. Widom, "Incremental computation and maintenance of temporal aggregates," *The VLDB Journal*, vol. 12, no. 3, pp. 262–283, 2003, Retrieved: September 2021. [Online]. Available: http://ilpubs.stanford.edu:8090/482/1/2000-6.pdf

[13] TU-Berlin-DIMA, "Scotty: Efficient window aggregation for out-of-order stream processing," *Generated by GitHub Pages*, 2021, Retrieved: September 2021. [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/

 ISBN: 978-1-61208-894-5

[14] 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: September 2021. [Online]. Available: https://hpi.de/fileadmin/user_upload/fachgebiete/rabl/publications/2018/ScottyICDE2018.pdf

[15] 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: September 2021. [Online]. Available: https://www.redaktion.tu-berlin.de/fileadmin/fg131/Publikation/Papers/Traub_TODS-21-Scotty_preprint.pdf

[16] K. Tangwongsan, M. Hirzel, and S. Schneider, "Sliding-window aggregation algorithms." 2019, Retrieved: September 2021. [Online]. Available: http://hirzels.com/martin/papers/encyc18-sliding-window.pdf

[17] ——, "Optimal and general out-of-order sliding-window aggregation," *Proceedings of the VLDB Endowment*, vol. 12, no. 10, pp. 1167–1180, 2019, Retrieved: September 2021. [Online]. Available: https://www.scott-a-s.com/files/vldb2019_fiba.pdf

[18] 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: September 2021. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9001104

[19] 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: September 2021. [Online]. Available: https://www.osti.gov/servlets/purl/1504207

[20] 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: September 2021. [Online]. Available: https://www.osti.gov/servlets/purl/1426900

[21] 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: September 2021. [Online]. Available: http://da2pl.cs.put.poznan.pl/programme/detailed-programme/da2pl2018-abstract-14.pdf

[22] 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: September 2021. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2388716

[23] 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: September 2021. [Online]. Available: http://www.cs.yale.edu/publications/techreports/tr222.pdf

[24] ——, "Updating formulae and a pairwise algorithm for computing sample variances," in *COMPSTAT 1982 5th Symposium held at Toulouse 1982*, 1982, pp. 30–41, Retrieved: September 2021. [Online]. Available: https://apps.dtic.mil/sti/pdfs/ADA083170.pdfP

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

[26] W. Hansch and T. Kubot, "Factory Dynamics Chapter 7 Lectures at the Universitaet der Bundeswehr Muenich," p. 68, Retrieved: September 2021. [Online]. Available: https://fac.ksu.edu.sa/sites/default/files/Factory%20Dynamics.pdf

[27] 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: September 2021. [Online]. Available: https://doi.org/10.1016/j.egypro.2015.07.474

[28] 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. [Online]. Available: http://www.thinkmind.org/articles/intsys_v13_n12_2020_2.pdf

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

[30] 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: September 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0895717701001091

[31] 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.

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

[33] 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: September 2021. [Online]. Available: https://de.booksc.eu/book/6543977/98912d

[34] J. Lewis and T. Disney, "Large limits to software estimation," *ACM Software Engineering Notes*, vol. 26, no. 4, pp. 54–59, 2001, Retrieved: September 2021. [Online]. Available: http://scribblethink.org/Work/Softestim/kcsest.pdf

[35] J. Lewis, "Mathematical limits to software estimation: Supplementary material," *Stanford University*, 2001, Retrieved: September 2021. [Online]. Available: http://scribblethink.org/Work/Softestim/softestim.html

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

[37] 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: September 2021. [Online]. Available: https://gajrc.com/media/articles/GAJEB_11_21-26_zOIbTWD.pdf

[38] E. A. Lee, "What is real time computing? a personal view." *IEEE Des. Test*, vol. 35, no. 2, pp. 64–72, 2018, Retrieved: September 2021. [Online]. Available: https://ptolemy.berkeley.edu/projects/chess/pubs/1192/Lee_WhatIsRealTime_Accepted.pdf

[39] TimeSys Corporation, "The concise handbook of real-time systems," *TimeSys Corporation Pittsburgh, PA, Version 1.3*, pp. 1–65, 2002, Retrieved: September 2021. [Online]. Available: https://course.ece.cmu.edu/~ece749/docs/RTSHandbook.pdf

[40] 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: September 2021. [Online]. Available: http://www.iosrjournals.org/iosr-jce/papers/Vol19-issue1/Version-2/E1901022938.pdf

 ISBN: 978-1-61208-894-5