Effective Temporal Dependence Discovery in Time Series Data

Qingchao Cai[†], Zhongle Xie[†], Meihui Zhang[‡*], Gang Chen[§], H. V. Jagadish[¶], Beng Chin Ooi[†]
[†]National University of Singapore, [‡]Beijing Institute of Technology, [§]Zhejiang University, [¶]University of Michigan
[†]{caiqc, zhongle, ooibc}@comp.nus.edu.sg
[‡]meihui_zhang@bit.edu.cn [§]cg@zju.edu.cn [¶]jag@umich.edu

ABSTRACT

To analyze user behavior over time, it is useful to group users into cohorts, giving rise to *cohort analysis*. We identify several crucial limitations of current cohort analysis, motivated by the unmet need for temporal dependence discovery. To address these limitations, we propose a generalization that we call *recurrent cohort analysis*. We introduce a set of operators for recurrent cohort analysis and design access methods specific to these operators in both single-node and distributed environments. Through extensive experiments, we show that recurrent cohort analysis when implemented using the proposed access methods is up to six orders faster than one implemented as a layer on top of a database in a single-node setting, and two orders faster than one implemented using Spark SQL in a distributed setting.

PVLDB Reference Format:
Qingchao Cai, Zhongle Xie, Meihui Zhang, Gang Chen, H. V. Jagadish, and Beng Chin Ooi. Effective Temporal Dependence Discovery in Time Series Data. *PVLDB*, 11 (8): 893-905, 2018.
DOI: https://doi.org/10.14778/3204028.3204033

1. INTRODUCTION

Like temporal locality in the data access of processes, there is a strong temporal dependence in user behavior in many contexts. One of the most well-known examples is stock market, where the price of a stock is widely believed to depend highly on its recent behavior in terms of many factors, such as trading volume, price change as well as many other technical metrics. As another example, whether a user of an E-commerce company will be retained is often related to her previous experience, and finding the behaviors that can improve user experience is hence of much importance to the business.

Such temporal dependence in user behavior has many applications. Identifying the activities that impact users' future behavior, in either a positive or a negative way, can be helpful in improving user engagement, as mentioned in the above user retention example. It can also be used for user behavior prediction, and hence can generate momentary value in many applications such as stock trading. In particular, with an established temporal dependence, the current behavior of a user provides a hint as to how she will behave in near future, which can be further fed into a machine learning framework for user behavior prediction.

Example 1. *For a stock portfolio, find the relationship between the trading volume of each week and the average price change in the following weeks.*

Example 1 aims to discover the temporal dependence between the trading volume (the causative behavior) and the average price change (the dependent behavior) thereafter, which represents a popular category of technical analysis in stock market. While investors are eager to find such dependences, there are no existing tools that allow them to do so. Investors are limited to whatever they can discover visually from a graphical plot. *This example is even more interesting if we can only consider the trading volume of the weeks when a short-term (e.g., 5-week) moving average crosses above a long-term (e.g., 10-week) moving average, which is widely considered as a buy signal in technique analysis.* While modern stock tools [5] can help to locate the time of such crossovers, they cannot summarize the price movement thereafter, not to mention how it depends on the trading volume of the week when the crossover takes place.

Discovering and exploiting temporal dependence is highly desired in many other applications. For one more example, consider health care, an area in which we have ongoing research collaborations. Analysis of temporal dependence can be of value throughout the entire disease treatment process, from disease prevention (by finding potential behaviors leading to the disease [8]) to treatment (by choosing the treatment/medicines with the best effect or the most cost-effectiveness, for the patient profile provided [24]).

A temporal dependence represents a causal relationship. It contains at least two components: the *causative behavior*, which is the cause that impacts users' future behavior, and the *dependent behavior*, which is the effect caused by (and hence happening after) the causative behavior. A concept of *time window* is also needed to enable the measurement of causative and dependent behavior to be well-defined. *It should be noted that a certain behavior can be not only a causative behavior that affects the future dependent behavior, but also a dependent behavior caused by previous causative*

*corresponding author

Proceedings of the VLDB Endowment, Vol. 11, No. 8

DOI: https://doi.org/10.14778/3204028.3204033

behaviors. That is to say, *each causative behavior corresponds to multiple dependent behaviors, and vice versa.*

The above framework for finding temporal dependence in user behavior fits well into *cohort analysis* [1, 3, 4, 6, 19], which was originally introduced in social science, and has recently demonstrated value in web user behavior analysis because of its ability to uncover user behavior patterns that are difficult to surface with traditional OLAP (online analysis processing) [19]. Typically, cohort analysis explores user (people) behavior from two angles: social change and aging, which have been considered as the two key factors impacting user behavior [17]. To capture social change, it organizes the total population into cohorts, each of which *consists of users that share common characteristics*, e.g., time and location, in doing a certain event (termed as the birth event), and then measures the behavior for each cohort at different *ages* (time periods after birth) to reflect the effect of aging.

A correspondence can be easily established between cohort analysis and the proposed framework for discovering temporal dependence. The causative behavior corresponds to the birth of users. The cohort is in turn defined as *a group of users that share common causative behavior*. For each cohort (i.e., causative behavior), its dependent behaviors are measured over each calendar time window (e.g., week) following the causative behavior. An age is associated with each dependent behavior according to its chronological order within all dependent behaviors of the same user. Finally, the dependent behaviors measured for a same cohort and a same age are merged together to reflect the collective dependent behavior of that cohort at that age, and in turn establish the dependence between each causative behavior (i.e., cohort) and the respective dependent behaviors.

However, there exist several limitations which hinder the cohort analysis from being directly used for the discovery of temporal dependence. First, the birth of users is defined with respect to a single *birth event*, and hence significantly restricts the diversity of causative behavior. For instance, the causative behavior of Example 1 cannot be mapped into a single birth event, as it needs to be summarized over a whole week. Second, the dependent behaviors are measured over fixed time durations, which suffices in the standard social science context where most people go through different life stages at roughly the same rate. However, in general, we cannot assume such an even rate of progress across individuals. For example, in some applications, users may initially begin as trial users, then transition to loyal users and power users. Some users may quickly grow into power users, while others may remain occasional users forever. We may not be able to assign meaningful average time duration for these phases: the variance may be too great.

Finally, there exists another fundamental limitation in cohort analysis which is the largest obstacle to its application to the discovery of temporal dependence. Specifically, the birth of a user in cohort analysis corresponds to the first occurrence of a certain *birth event*, which uniquely determines the host cohort of that user. While this specification works well in the context of social science, where the birth event is usually defined as the physical human birth, and hence occurs exactly once, it can be misleading when applied to temporal dependence discovery. As mentioned above, along the lifetime, each user can continuously conduct causative behaviors which often vary significantly across different time windows, and each of them must be taken into account for a meaningful dependence. In Example 1, the causative behavior, i.e., the trading volume, of each user (stock) keeps changing across different weeks, and merely considering the first week would disqualify most causative behaviors, leading to a significant deviation in the discovered dependence.

1.1 Contributions

This paper presents *recurrent cohort analysis*, which lifts all the aforementioned limitations of traditional cohort analysis and is hence a powerful tool for temporal dependence discovery. The major contributions of this paper are summarized below.

- *Recurrent cohort analysis*: We gradually give a formal definition of recurrent cohort analysis by first introducing the core concept of *time window attribute*, and then using it to construct the three components of recurrent cohort analysis: *causative behavior measurement*, *dependent behavior measurement* and *age definition*. The flexible definition of time window attribute enables to explore a huge space of temporal dependence.

- *Cohort operators*: We present a framework of table transformation to address recurrent cohort analysis in a database context, and further propose a set of cohort operators that capture the essential table transforms within the framework, and can hence be combined to generate a concise query plan for evaluation.

- *Operator evaluation*: We present a non-intrusive strategy and an intrusive strategy to evaluate the proposed cohort operators. The non-intrusive strategy simply translates each operator into standard SQL statements. A materialization optimization for this strategy is also presented. For the intrusive strategy, we natively implement the cohort operators on top of the storage layer of COHANA [19]. We also propose a system architecture for distributed evaluation of this intrusive strategy.

- *A comprehensive performance study*: We implement the non-intrusive strategy atop two modern database systems, and conduct a comprehensive performance comparison study between the non-intrusive strategy and the intrusive strategy.

The rest of this paper is structured as follows. Section 2 introduces the concept of time window which unifies the measurement of user behavior. The recurrent cohort analysis is presented in Section 3. To address it, we propose a set of database operators in Section 4, and discuss their evaluation in Section 5. A system architecture for distributed operator evaluation is presented in Section 6. Section 7 gives a performance study on different evaluation strategies. We present the related work in Section 8, and conclude this paper in Section 9.

2. TIME WINDOW

At the core of recurrent cohort analysis is the measurement of causative behaviors and dependent behaviors. In order to make it well-defined, we first introduce a concept of *time slice*, which is the minimum time unit over which user behavior can be measured. For flexibility, we further

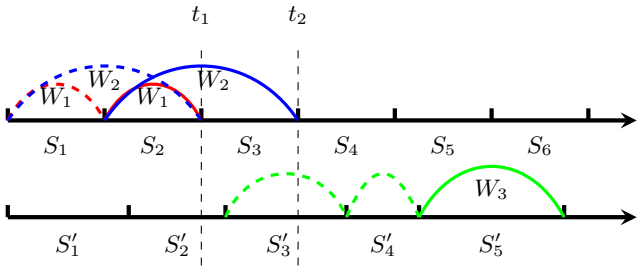


Figure 1: Sliding time windows in the measurement of causative behaviors (the upper line) and dependent behaviors (the lower line)

introduce another concept of *time window*, which consists of several consecutive time slices for user behavior measurement and is allowed to slide such that both the range and the number of the constituent time slices can vary during sliding. We use a similar data model to that of [19], which is also a relational table, **eventT**, that records all the information about each activity of each user. The schema of this table can be found in Table 1. It has two mandatory columns: user and time, which record who performed the activity at what time. Unlike the model of [19], **eventT** does not necessarily contain the event column.

2.1 Time slice

Both the causative and dependent behaviors are required to be measured along time dynamically. A common and natural way to do so is to partition the lifetime of each user into fixed-length time intervals, e.g., day/week/month, such that user behavior can be measured for each interval [20]. We also take this approach except that, instead of only based on calendar, user lifetime can be partitioned in multiple ways, as shown below.

Definition 1 (Lifetime partition)**.** *The partition of user lifetime, $\mathbb{P}$, is defined as*

$$\mathbb{P} \triangleq \begin{cases} (\mathsf{A}, \mathcal{C}), & \text{if } \mathsf{A} = \mathsf{event} \\ (\mathsf{A}, U), & \text{if } \mathsf{A} = \mathsf{time} \\ (\mathsf{A}), & \text{otherwise} \end{cases}$$

where A *is an attribute of the user activity relation* **eventT**, *$\mathcal{C}$ is an optional propositional formula on* **eventT**, *and U is a calendar time unit (e.g., day/week/month).*

Definition 2 (Time slice)**.** *Given a partition $\mathbb{P}$, the lifetime of a user u is divided into the following time slices:*

$$S_u = \{[t_i, t_{i+1}) | i \geq 0 \wedge t_i \in D \wedge t_{i+1} \leftarrow \min\{t | t \in D \wedge t > t_i\}\}$$

where D, as given below, is the delimiter set that partitions the lifetime of u,

$$D = \begin{cases} \{d_{u,i}[\mathsf{time}] | i = 1 \vee \mathbb{P}.\mathcal{C}(d_{u,i}) = true\}, & \text{if } \mathbb{P}.\mathsf{A} = \mathsf{event} \\ \{d_{u,i}[\mathsf{time}] | i = 1 \vee d_{u,i}[\mathbb{P}.\mathsf{A}] \neq d_{u,i-1}[\mathbb{P}.\mathsf{A}]\}, & \\ & \text{if } \mathbb{P}.\mathsf{A} \notin \{\mathsf{event}, \mathsf{time}\} \end{cases}$$

where $d_{u,i}$ represents the i-th activity that user u performed in chronological order, and $d_{u,i}[\mathsf{A}]$ is the value that the column A *takes in $d_{u,i}$.*[1] *For the sake of expression, the i-th time slice of S_u, i.e., $[t_{i-1}, t_i)$, is denoted by $S_{u,i}$.*

[1]We omit the definition of delimiters for the case of $\mathbb{P}.\mathsf{A} =$ time, as it is trivial to understand but cumbersome to express.

Roughly speaking, when the attribute of $\mathbb{P}$ is time, the user lifetime is partitioned such that each time slice corresponds to a time interval of $\mathbb{P}.U$. When the attribute of $\mathbb{P}$ is event, each delimiter corresponds to a time point at which the activity performed is qualified with respect to $\mathbb{P}.\mathcal{C}$. Otherwise, the delimiter set contains the time points at which the value of the attribute $\mathbb{P}.\mathsf{A}$ changes as compared to that of the previous activity. An example of time slices is shown in Figure 1, where two different $\mathbb{P}$ are used to partition the lifetime of a same user, t_i and t'_i are the delimiters defined by the two partitions, and S_i and S'_i are the time slices determined by the delimiters.

The first two cases, i.e., $\mathbb{P}.\mathsf{A} =$ time and $\mathbb{P}.\mathsf{A} =$ event, are adapted from traditional cohort analysis where an event-based partition is implicitly specified by the birth event, and a calendar partition is involved in the cohort behavior measurement. The last case where user lifetime is partitioned by an attribute other than time and event (termed as dimensional partition hereafter), is motivated by the fact that many attributes are step functions of time such that the value remains constant for an indeterminate time duration followed by a sudden jump. Representatives of such attributes can be E-commerce session, discretized stock price and roles of a game. This flexible definition of time slice significantly diversifies the measurement of causative and dependent behaviors, and hence enables to explore a large space of temporal dependence in user behavior.

2.2 Time window

Summarizing the behavior for each time slice enables us to measure user behavior dynamically along user lifetime. In addition, time slices can also differentiate the dependent behavior from causative behavior, since the chronological order between behaviors can be easily deduced from the time slice set defined in Definition 2. Therefore, in the simplest case, we can use time slices to specify the measurement of both causative behaviors and dependent behaviors. However, this restricts user behavior to be measured over a single time slice each time, and is insufficient in many cases. Consider Example 1 with the trading volume of each single week as the causative behavior. If we want to find how the recent 5-week volume affects future price, then time slice based specification no longer works.

A natural way to handle the above problem is to define a *time window* which contains a configurable length in terms of the number of consecutive time slices, and is allowed to slide over slices such that a time window with length l will consist of slices $(S_{u,i-l+1}, \ldots, S_{u,i})$ when it slides to slice $S_{u,i}$. However, there still exists limitation with this definition of time window: we cannot construct a time window that consists of time slices earlier than the one that it currently slides to, which, however, is very useful in many cases. Consider again Example 1. If we want to study the impact of trading volumes of those weeks that have a higher closing price than their previous weeks, then time windows with only a length field will not suffice.

Definition 3 (Time window)**.** *A time window $\mathbb{W}$ is defined as follows:*

$$\mathbb{W} \triangleq (\mathbb{P}, w^l, w^h)$$

where $\mathbb{P}$ is given in Definition 1, and for a time slice $S_{u,i}$ this time window slides to, w^l and w^h are two integers determining the set of the respective constituent time slices,

$T_{u,i}$, *which is given by*

$$T_{u,i} = \{S_{u,j} | f(i, w^l) \leq j \leq f(i, w^h)\} \quad (1)$$

where the function f *is given by*

$$f(i, w) = \begin{cases} w & \text{if } w > 0 \\ i + w & \text{otherwise} \end{cases} \quad (2)$$

and we term $S_{u,i}$ *as the* **current position** *of this time window.*

To address this limitation, we adopt the above definition of time windows, which allows us to configure both the number and position of the constituent time slices. The introduction of w^l and w^h significantly increases the flexibility in specifying time windows and hence also in user behavior measurement. Basically, each of them can be specified in either an absolute way by assigning it with a positive integer, or a relative way by assigning it with a non-positive integer. Consider w^l, which determines the earliest time slice for each position of the time window. As can be observed from Equation 2, if w^l is assigned with a positive value, then for user u, the earliest time slice composing the time window is always S_{u,w^l}, regardless of the position. In the other case, the referenced time slice of w^l keeps changing as the time window slides.

Figure 1 demonstrates the sliding traces of two time windows W_1 and W_2, which are respectively surrounded by blue and red curves. They share a same partition $\mathbb{P}$, but differ in the specification of w^l and w^h such that $W_1.w^l = W_1.w^h = W_2.w^l = -1$ and $W_2.w^h = 0$. The initial position of both time windows is S_2, and the corresponding constituent time slices are $\{S_1\}$ and $\{S_1, S_2\}$, as surrounded by the two dotted lines. When $_1$ and W_2 slide to S_3, their constituent time slices become $\{S_2\}$ and $\{S_2, S_3\}$, respectively. Therefore, we are now able to specify a time window whose constituent time slices locate away from its current position.

2.3 Time window attribute

A time window defined in Definition 3 enables us to dynamically measure behavior of a user over its lifetime by summarizing the activities over the constituent time slices for each position in the sliding path of this time window. This process can be modeled as the evaluation of a *time window attribute*, which is defined below.

Definition 4 (Time window attribute)**.** *A time window attribute* $\mathbb{A}$ *is defined as follows:*

$$\mathbb{A} \triangleq (f, \mathsf{A}, \mathbb{W}) \quad (3)$$

where f *is an aggregate function,* A *is an attribute of user activity table* **eventT***, and* $\mathbb{W}$ *is a time window.*

However, a time window attribute may not be able to capture user behavior in many cases. One such example can be found in Example 1, where for each trading volume, we need to measure the price change in each week thereafter. This requires us to compute the price change in each week, which is the difference between the closing price of the current week and that of the previous week. Hence, we need two time window attributes $\mathbb{A}_1$ and $\mathbb{A}_2$ which respectively compute the closing price for the current week and the previous week, and hence need be configured as $\mathbb{A}_1.\mathbb{W}.w^l = \mathbb{A}_1.\mathbb{W}.w^h = 0$ and $\mathbb{A}_2.\mathbb{W}.w^l = \mathbb{A}_2.\mathbb{W}.w^h = -1$. Given $\mathbb{A}_1$ and $\mathbb{A}_2$, we can now express the price change simply as $\mathbb{A}_1 - \mathbb{A}_2$. In addition, as we shall see in Section 3.3, for each trading volume, the price changes of multiple weeks have to be averaged in order to measure the dependent behavior of stocks of a same cohort. In order to support such cases, we further define the *composite time window attribute* and the *compound time window attribute.*

Definition 5 (Composite time window attribute)**.** *A composite time window attribute is an arithmetic expression including one or more time window attributes which share same underlying lifetime partition.*

Definition 6 (Compound time window attribute)**.** *A compound time window attribute is defined in the same way as in Equation 3, except that* A *is no longer an attribute of* **eventT***, but a time window attribute with the same lifetime partition as* $\mathbb{W}$*.*

It should be noted that each time window attribute involved in Definition 5 and 6 is not restricted to be the one given in Definition 4, but can itself be a composite or a compound time window attribute. In the rest of paper, when we mention a time window attribute, unless explicitly specified, its definition can be any one of Definition 4, 5 and 6.

As we shall see, time window attribute plays a central role in recurrent cohort analysis. It presents an elegant abstract for dynamic user behavior measurement, and hence enables a concise definition of recurrent cohort analysis where user behavior need be measured variously.

3. RECURRENT COHORT ANALYSIS

In this section, we first give a definition of user behavior measurement, and then show how it can be used to compose recurrent cohort analysis. As before, we assume **eventT** as our data model.

3.1 User behavior measurement

In the simplest case, a single time window attribute defined in Definition 4 suffices for the purpose of defining user behavior measurement, as it provides a clear specification on how to measure the behavior of each user along its lifetime, which is to aggregate the given attribute over the constituent time slices for each time window position of each user.

There are two problems with this form of user behavior measurement. First, it does not support the selection of user activities for behavior measurement. This support is analogous to the selection predicate that precedes the aggregation in conventional database analytics, and is indispensable in the cases where not all user activities are of interest. Consider Example 1. It is often the case that we are only interested in the stocks of a certain sector, or that we want to exclude the prices earlier than a certain date. The second problem is that this form lacks a mechanism to prune uninteresting positions in the sliding path of the time window $\mathbb{A}.\mathbb{W}$. As we have mentioned in Section 1, for Example 1, *we might be only interested in the weeks when the 5-week moving average crosses above the 10-week moving average.*

Definition 7 (User behavior measurement)**.** *User behavior measurement* $\mathbb{M}$ *is defined as*

$$\mathbb{M} \triangleq (\mathcal{C}^e, \mathcal{C}^w, \mathbb{A})$$

where $\mathcal{C}^e$ *is a propositional formula on the user activity relation* **eventT***,* $\mathcal{C}^w$ *is a propositional formula on a set of*

time window attributes, and $\mathbb{A}$ is a time window attribute representing the user behavior. The time window attributes involved in $\mathcal{C}^w$ share the same lifetime partition as $\mathbb{A}$.

The above definition gives a precise description on how to measure user behavior. The time window slides over the entire lifetime of each user. For each position during the sliding, if $\mathcal{C}^w$ evaluates to true, then the behavior will be measured by aggregating $\mathbb{A}.f$ over the attribute $\mathbb{A}.\mathsf{A}$ of the activities that are qualified with respect to $\mathcal{C}^e$.

3.2 Recurrent cohort analysis

Definition 8 (Recurrent cohort analysis)**.** *Recurrent cohort analysis is defined as*

$$(\mathbb{M}^c, \mathbb{M}^d, [\mathbb{A}^g])$$

where $\mathbb{M}^c$ and $\mathbb{M}^d$, as defined in Definition 7, respectively represent the measurement of causative behavior and dependent behavior, and $\mathbb{A}^g$ is an optional time window attribute which shares the underlying partition with $\mathbb{M}^d.\mathbb{A}$, and determines the age of each dependent behavior relative to a given causative behavior.

In the above definition of recurrent cohort analysis, the causative behavior measurement $\mathbb{M}^c$ determines the cohorts that each user entered during its lifetime: at each time window position where $\mathbb{M}^c.\mathcal{C}^w$ evaluates to true, this user enters a cohort represented by the causative behavior measured over respective time slices.

The dependent behavior measurement $\mathbb{M}^d$ does slightly differently from $\mathbb{M}^c$ in that it measures the collective behavior of users belonging to a same cohort for each age that those users experienced after entering the cohort. Specifically, for each cohort c that a user enters, the time window $\mathbb{M}.\mathbb{A}.\mathbb{W}$ slides from the first position that locates after the cohort entrance. Thereafter, for each position p in the sliding path where $\mathbb{M}^d.\mathcal{C}^w$ evaluates to true, the activities within the constituent time slices of p, which can be derived from Equation 1, are considered to take place at an age to which $\mathbb{A}^g$ evaluates for p, and hence will be taken into consideration during measuring the cohort c's behavior at that age. By default, $\mathbb{A}^g$ is the number of positions that the time window has already slid until p (inclusive), which is consistent with the age definition in traditional cohort analysis [19].

Consider Figure 1, which gives the lifetime of a single user. Suppose the time windows for measuring causative behavior and dependent behavior, i.e., $\mathbb{M}^c.\mathbb{A}.\mathbb{W}$ and $\mathbb{M}^d.\mathbb{A}.\mathbb{W}$, are W_2 and W_3, respectively, where $W_2.w^l = -1$ and $W_2.w^h = W_3.w^l = W_3.w^h = 0$. For simplicity, we assume the age $\mathbb{A}^g$ is set by default, and both of $\mathbb{M}^c.\mathcal{C}^w$ and $\mathbb{M}^d.\mathcal{C}^w$ are empty and hence always evaluate to true. In this case, initially W_2 is positioned at S_2, and the first causative behavior of this user, denoted by c_1, is thus aggregated over the activities within time slices $\{S_1, S_2\}$. When W_2 slides to S_3, the respective causative behavior c_2 is then summarized over the activities within $\{S_2, S_3\}$. Since each causative behavior also represents a cohort, after performing the two causative behaviors, this user enters cohort c_1 and c_2 at time t_1 and t_2, respectively.

Now we consider the dependent behavior measurement of this example. After entering cohort c_1, the time window W_3 starts to slide from slice S'_3, which is the first time slice after t_1. For this cohort, the age of the user is 1 when W_3 is at the initial position S'_3, and is incremented by 1 each time W_3 slides to the next slice. Similarly, for cohort c_2, this user is at age 1 when W_3 is initially positioned at S'_4, and grows up as W_3 moves forward. As a result, to measure the behavior of cohort c_1 at age 1 and 2, the activities within S'_3 and S'_4 should be taken into consideration, respectively; for cohort c_2, the activities within S'_4 and S'_5 should be respectively included in order to measure the behavior of age 1 and 2.

3.3 Examples

We consider the refined version of Example 1 which only considers the trading volume of the weeks when the 5-week moving average crosses above the 10-week moving average. Hence, we need to express the propositional formula $\mathcal{C}^w$ of the causative behavior measurement. To that end, we need to construct two time window attributes, denoted by $\mathbb{A}_1$ and $\mathbb{A}_2$, which calculate the average closing price for the most recent 5 weeks and for the most recent 10 weeks, and hence need be specified as $\mathbb{A}_1.w^h = \mathbb{A}_2.w^h = 0$, $\mathbb{A}_1.w^l = -4$ and $\mathbb{A}_2.w^h = -9$. In addition, we need another two time window attributes $\mathbb{A}_3$ and $\mathbb{A}_4$ which calculate the previous 5-week and 10-week average closing prices, and hence are specified as $\mathbb{A}_3.w^h = \mathbb{A}_4.w^h = -1$, $\mathbb{A}_3.w^l = -5$ and $\mathbb{A}_4.w^h = -10$. Then, we can formulate the crossover condition as $\mathbb{A}_1 > \mathbb{A}_2 \ \wedge \ \mathbb{A}_3 \leq \mathbb{A}_4$, and define the causative behavior measurement $\mathbb{M}^c$ as $(\emptyset, \mathbb{A}_1 > \mathbb{A}_2 \wedge \mathbb{A}_3 \leq \mathbb{A}_4, \mathbb{A}^c)$, where $\mathbb{A}^c$ computes the trading volume for the current week, and hence has a time window $\mathbb{W}$ with $w^l = w^h = 0$.

For the dependent behavior measurement $\mathbb{M}^d$, we need to average, for each cohort, the price changes over weeks of a same age. Therefore, a compound time window attribute is required, since as mentioned in Section 2.3, the price change is itself a composite time window attribute: $\mathbb{A}_5 - \mathbb{A}_6$, where $\mathbb{A}_5$ and $\mathbb{A}_6$ respectively compute the closing prices of the current week and the previous week, and hence are configured with $\mathbb{A}_5.\mathbb{W}.w^l = \mathbb{A}_5.\mathbb{W}.w^h = 0$ and $\mathbb{A}_5.\mathbb{W}.w^l = \mathbb{A}_6.\mathbb{W}.w^h = -1$. The compound time window attribute $\mathbb{A}^d$ can then be composed as $(f, \mathbb{A}_5 - \mathbb{A}_6, \mathbb{W})$, where f is the average function, and $\mathbb{W}$ is configured with $w^l = w^h = 0$. Finally, since there is no $\mathcal{C}^c$ or $\mathcal{C}^w$ specified for $\mathbb{M}^d$, we can now express $\mathbb{M}^d$ as $(\emptyset, \emptyset, \mathbb{A}^d)$.

4. OPERATORS FOR RECURRENT COHORT ANALYSIS

In this section, we address the recurrent cohort analysis in database context by demonstrating a series of table transformation that starts from the original user activity table **eventT**, and ends at the final table **resultT**, which establishes the temporal dependence between the causative and dependent behaviors. Since $\mathbb{M}^c.\mathcal{C}^e$ and $\mathbb{M}^d.\mathcal{C}^e$ are simple database predicates and trivial to evaluate, we assume they are both empty. For the simplicity of expression, we further assume the time window attributes involved in $\mathbb{M}^c$ and $\mathbb{M}^d$ share the same time window, denoted by $\mathbb{W}$. Albeit leading to additional tables generated, relaxing this assumption does not affect the transformation process, and is hence omitted.

4.1 Table transformation

As mentioned, to measure the causative behavior of each user, we need to summarize the activities within the constituent time slices for each qualified (wrt $\mathbb{M}^c.\mathcal{C}^w$) position in the sliding path of the time window $\mathbb{W}$. Hence, it is natural to first generate a table (**sliceT**) for time slices defined by

Table 1: The schemas of tables that are used for recurrent cohort analysis, where ✓ and ◯ respectively represent a mandatory column and an optional column.

tables \ columns	user	time	event	cohort	wpos	wlow	whigh	age	metric
eventT	✓	✓	◯						
sliceT	✓				✓	✓	✓		
windowT	✓				✓	✓	✓		
cohortT	✓			✓	✓	✓	✓		
winAgeT	✓				✓	✓	✓	✓	
cohortAgeT	✓			✓	✓	✓	✓	✓	
resultT				✓				✓	✓

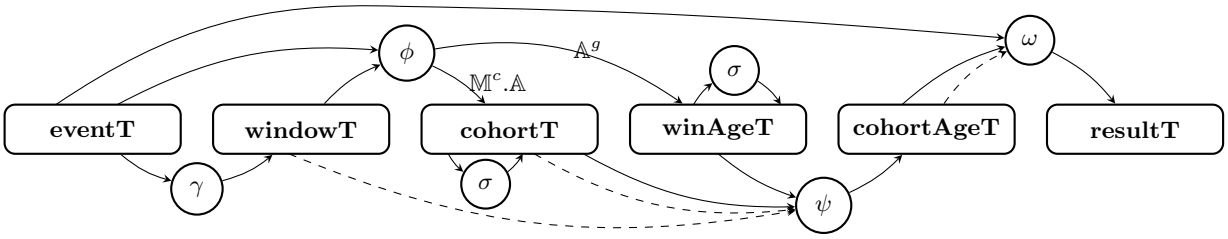


Figure 2: Table transformation

$\mathbb{W}.\mathbb{P}$, and then generate a table (**windowT**) which contains for each user the constituent time slices for each position in the sliding path of the time window. From **windowT**, we can identify the activities in **eventT** that need be included for the measurement of each causative behavior, and aggregate over them to generate a table (**cohortT**) which records for each user the cohorts (i.e., causative behaviors) that this user enters during the sliding of the time window W.

Likewise, the first two tables that need be generated for dependent behavior measurement $\mathbb{M}^d$ are the counterparts of **sliceT** and **windowT**. Since all time windows involved in $\mathbb{M}^c$ and $\mathbb{M}^d$ are same, **sliceT** and **windowT** can be reused in this stage. The next step is to determine the age of each user for each position in the respective sliding path, which leads to table **winAgeT**. Then, for each user and each cohort that this user entered (recorded in table **cohortT**), we collect the age associated with each sliding position after the cohort entrance, and record it in table **cohortAgeT**. Finally, we are able to locate the activities for each age of each cohort, and we aggregate over them to measure the respective dependent behavior and in turn generate the final table **resultT**.

The schemas of the aforementioned tables are shown in Table 1, where user, time, event, cohort and age have a meaning suggested by their names, wpos is the position of the time window, and wlow and whigh are respectively the lower and the upper time points of the constituent time slices of the position wpos.

The transformations between the tables are shown in Figure 2, where each table transformation is accomplished by an operator. There are in total five different operators involved, namely, *window slicing operator* γ, *window aggregation operator* ϕ, standard database selection operator σ, *age-by operator* ψ and *recurrent aggregation operator* ω. The window slicing operator γ completes the transformation between **eventT** and **windowT**, and hence conceals the intermediary table **sliceT** in Figure 2. The selection operator σ is used to enforce the conditions specified in $\mathbb{M}^c.\mathcal{C}^w$ and $\mathbb{M}^d.\mathcal{C}^w$. For ψ and ω, there can be alternative input tables, as shown by dashed lines, which respectively represent the case where the age-defining time window attribute is specified by default, and the case where the time window attribute $\mathbb{M}^d.\mathbb{A}$ for measuring the dependent behavior is compound (i.e., defined in Definition 6).

4.2 Operators

In this section, we give the definition of the operators that are introduced in Figure 2, except the standard database selection operator σ.

4.2.1 Window slicing operator γ

Definition 9 (Window slicing operator)**.** *The window slicing operator is defined as follow:*

$$\gamma = \{(u,p,l,h) | d \in \textbf{sliceT} \wedge u \leftarrow d[\mathsf{user}] \wedge p \leftarrow d[\mathsf{wpos}] \wedge l \leftarrow \min\{t | t \in S_{u,f(p,w_1)}\} \wedge h \leftarrow \max\{t | t \in S_{u,f(p,w_2)}\}\}$$

where f is given in Definition 3, $w_1 = \mathbb{W}.w^l$, $w_2 = \mathbb{W}.w^h$, and **sliceT** *is given by*

$$\{(u,p,l,h) | d \in \textbf{eventT} \wedge u \leftarrow d[\mathsf{user}] \wedge p \leq |\{S_u\}| \wedge l \leftarrow \min\{t | t \in S_{u,p}\} \wedge h \leftarrow \max\{t | t \in S_{u,p}\}\}$$

where S_u and $S_{u,p}$ are given in Definition 2.

The window slicing operator first generates **sliceT** from **eventT** to record the time slices of each user, and then produces **windowT** which contains the constituent time slices for each position in the sliding path of each user. The output of the window slicing operator is **windowT**, which records, for each user, the start time, wlow, and the end time, whigh, of the constituent time slices of each position, wpos.

4.2.2 Window aggregation operator ϕ

Definition 10 (Window aggregation operator)**.** *The window aggregation operator is defined as follow:*

$$\phi = \{(u,c,p,l,h) | d \in \textbf{windowT} \wedge u \leftarrow d[\mathsf{user}] \wedge p \leftarrow d[\mathsf{wpos}] \wedge l \leftarrow d[\mathsf{wlow}] \wedge h \leftarrow d[\mathsf{whigh}] \wedge c \leftarrow \mathbb{A}.f(\pi_{\mathbb{A}.\mathsf{A}}(\sigma_{\mathsf{user}=u \wedge l \leq \mathsf{time} < h}(\textbf{eventT})))\}$$

where $\mathbb{A}$ is the time window attribute representing the behavior to measure.

The window aggregation operator measures the behavior of each user along the sliding path of the time window. As shown in Figure 2, this operator is used for two purposes. First, it measures the causative behaviors of each user, and generates Table **cohortT**, where the measured behavior is recorded in the column cohort to represent the cohorts this user enters. Second, it outputs the age for each user at each position into table **winAgeT**, where the measured behavior is recorded in column age.

It should be noted that the above definition only applies to the case where $\mathbb{A}$ is defined by Definition 4. For the composite or the compound time window attribute, we need to do a little more work by further summarizing the behavior measured by ϕ. We omit this case since it is trivial to deduce.

4.2.3 Age-by operator ψ

Definition 11 (Age-by operator)**.** *The age-by operator is defined as follow:*

$$\begin{aligned}\psi = \{(u, c, p,\, l, h, a) | d \in \mathbf{cohortT} \wedge d' \in \mathbf{winAgeT} \\ \wedge\, d[\mathsf{user}] = d'[\mathsf{user}] \wedge d[\mathsf{wpos}] < d'[\mathsf{wpos}] \\ \wedge\, u \leftarrow d[\mathsf{user}] \wedge c \leftarrow d[\mathsf{cohort}] \wedge p \leftarrow d'[\mathsf{wpos}] \\ \wedge\, l \leftarrow d[\mathsf{wlow}] \wedge h \leftarrow d[\mathsf{whigh}] \wedge a \leftarrow d'[\mathsf{age}]\}\end{aligned}$$

For each user and each cohort it enters, the age-by operator simply associates each following position in the sliding path with an age that is determined by $\mathbb{A}^g$. For the case with a default $\mathbb{A}^g$ where the age is the number of positions that the time window has slid, we can replace **winAgeT** and $a \leftarrow d'[\mathsf{age}]$ with **windowT** and $a \leftarrow d'[\mathsf{wpos}] - d[\mathsf{wpos}]$, respectively. This case corresponds to the dashed lines associated with the age-by operator ψ in Figure 2.

4.2.4 Recurrent aggregation operator ω

Definition 12 (Recurrent aggregation operator)**.** *The recurrent aggregation operator is defined as follow:*

$$\begin{aligned}\phi = \{(c, a, s, m) | \mathbf{T} \leftarrow \{(c, a, b) | d \in \mathbf{cohortAgeT} \\ \wedge\, d' \in \mathbf{eventT} \wedge d[\mathsf{user}] = d'[\mathsf{user}] \\ \wedge\, d[\mathsf{wlow}] \leq d'[\mathsf{time}] < d[\mathsf{whigh}] \\ \wedge\, c \leftarrow d[\mathsf{cohort}] \wedge a \leftarrow d[\mathsf{age}] \wedge b \leftarrow d'[\mathsf{A}]\} \\ \wedge\, d \in \pi_{\mathsf{cohort},\mathsf{age}}(\mathbf{T}) \wedge c \leftarrow d[\mathsf{cohort}] \wedge a \leftarrow d[\mathsf{age}] \\ \wedge\, s \leftarrow |\pi_{\mathsf{user}}(\sigma_{\mathsf{cohort}=c}(\mathbf{T}))| \\ \wedge\, m \leftarrow f(\pi_{\mathsf{A}}(\sigma_{\mathsf{cohort}=c \wedge \mathsf{age}=a}(\mathbf{T})))\}\end{aligned}$$

where $\mathsf{A} = \mathbb{M}^d.\mathbb{A}.\mathsf{A}$ is the attribute of **eventT** *over which the dependent behavior shall be measured, and* **T** *is a temporary table with a schema of* (cohort, age, A) *which contains all the values of* A *that need be included for measuring the dependent behavior of each* cohort *at each* age*.*

Basically, the recurrent aggregation operator first identifies for each cohort at each age, all the activities that are performed by users of that cohort at that age, and then summarizes over those activities to show how the dependent behavior is related to the cohort and age, thereby establishing the desired temporal dependence.

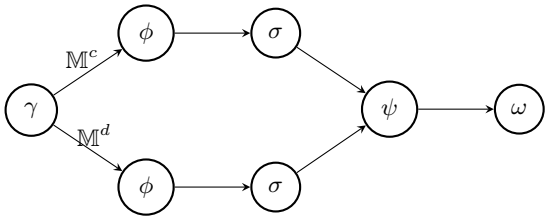


Figure 3: Query plan. The upper branch of γ is for causative behavior measurement, and the lower branch is for dependent behavior measurement.

The above definition applies to the case where A is an attribute of **eventT**. When it is another time window attribute, we can first use a window aggregation operator ϕ to evaluate it for each position in the sliding path of each user, and append the result to **cohortAgeT**. Then, **cohortAgeT** itself can be used to measure the dependent behavior in the similar way as **T** of Definition 12 does, which corresponds to the dashed line associated with the recurrent aggregation operator ω in Figure 2.

4.3 Query plan

From Figure 2, it is intuitive to derive the query plan for recurrent cohort analysis. As shown in Figure 3, the query plan is actually a sequential execution of the five operators involved in Figure 2. The window aggregation operator ϕ and the selection operator σ are both executed twice. In addition to $\mathbb{M}^c.\mathbb{A}$ and $\mathbb{A}^g$, the two window aggregation operators ϕ also measure the time window attributes that are respectively included in $\mathbb{M}^c.\mathcal{C}^w$ and $\mathbb{M}^d.\mathcal{C}^w$, in order to evaluate the two selection operators σ. This is possible because we assume all time window attributes involved in $\mathbb{M}^c$ and $\mathbb{M}^d$ share the same underlying time window configuration. However, in normal cases where there can be multiple configurations in the time slice range of time windows, we need execute ϕ the same number of times.

5. OPERATOR EVALUATION

In this section, we present two evaluation strategies for the operators devised in the previous section. The first strategy is to translate these operators into standard SQL statements (SQL-based strategy), and for the second one (intrusive strategy), we natively implement the proposed operators atop the storage layer of COHANA [19].

5.1 SQL-based strategy

We shall only discuss how to generate **sliceT** that is used in the window slicing operator γ, since the generation of all other tables can be straightforwardly translated to SQL queries from the definitions of respective operators.

As shown in Definition 9, **sliceT** records the time slices of each user partitioned by a given $\mathbb{P}$. Hence, it suffices to compute the delimiters defined by $\mathbb{P}$, which can be found in Definition 2. In the case of $\mathbb{P}.\mathsf{A} \in \{\mathsf{time}, \mathsf{event}\}$, the delimiters can be trivially computed by using built-in time functions or simply applying a standard database selection operator on the user activity table **eventT**.

In the case of dimensional partition where $\mathbb{P}.\mathsf{A} \notin \{\mathsf{time}, \mathsf{event}\}$, each delimiter corresponds to a change in the value of $\mathbb{P}.\mathsf{A}$. We can associate each activity with its rank in the chronologically sorted activities performed by the same user.

The delimiters can then be obtained by joining this augmented activity table with itself on the following condition θ:

$$\theta \triangleq \mathbf{T}_1.\mathsf{user} = \mathbf{T}_2.\mathsf{user} \wedge \mathbf{T}_1.\mathsf{A} \neq \mathbf{T}_2.\mathsf{A} \wedge \mathbf{T}_1.\mathsf{rank} = \mathbf{T}_2.\mathsf{rank}+1$$

where $\mathbf{T}_1$ and $\mathbf{T}_2$ are both aliases of the augmented **eventT**, and A is given in the partition $\mathbb{P}$.

5.1.1 Optimization

During the translation of window generation operator, we can optimize for the case of temporal and dimensional time window by materializing the delimiters of each time slice along with each activity record belonging to that slice, since they are static in these two cases. However, for the event-based case, such materialization optimization cannot be applied as the time slices depends on the propositional formula specified, which can vary from time to time.

The materialization, however, does not come at no cost. It trades storage space for processing time. For it to work, one may need to do the materialization for each time unit and for each possible dimension. Each materialization means two additional columns added to the activity relation, and hence can significantly offset the gain in processing time, especially in the cases of limited runtime memory, where additional IOs may occur due to the increase in the data size. An alternative way of materialization is to persist **sliceT** separately. This way, the storage cost is small, but we need to join this table with the original activity table for the evaluation of each time window attribute.

Unfortunately, the most serious problem of the SQL-based evaluation strategy is that we need to join the cohort table **cohortT** with **eventT** for the recurrent aggregation operator, as can be seen from Definition 12. The processing time of this join operation grows quadratically with the number of activities, since the size of both tables grow linearly as users perform more activities. As such, the SQL-based evaluation strategy does not scale well.

5.2 Intrusive strategy

It can be seen from Section 4 that except the recurrent aggregation operator ω, the other operators involved in the query plan manipulate the input tables in user granularity. For each user, the window slicing operator γ computes the constituent time slices of each position in the sliding path of the time window; the window aggregation operator ϕ measures the behavior for each position; the selection operator σ prunes the positions that are unqualified with respect to $\mathcal{C}^w$, and for each cohort this user enters, the age-by operator ψ determines the age for each position following the cohort entrance. Although the recurrent aggregation operator aggregates over activities for multiples users of a same cohort, it can still execute in a user-by-user manner, since for most aggregation functions, such as SUM, MIN, MAX, COUNT and AVG, the final result can be calculated from the values aggregated for each user. In addition, the database selection operator used to evaluate $\mathbb{M}^c.\mathcal{C}^e$ and $\mathbb{M}^d.\mathcal{C}^e$ operates in the granularity of user activity, and is hence even finer than the other operators. Therefore, to perform recurrent cohort analysis, we can first evaluate the query plan for each user and combine the respective results together to establish the desired dependency between the causative behavior and the dependent behavior.

To that end, we present another evaluation strategy for the proposed operators by implementing the proposed operators on top of the storage layer of COHANA [19], which we developed for traditional cohort analysis. This system organizes the user activity table **eventT** by clustering the activities of the same user in chronological order, and employs a hierarchical storage layout by horizontally partitioning table into multiple equal-size chunks, and applying multiple compression techniques for the storage of each chunk column. Each chunk column provides both sequential access and random access through the interface of `getNextTuple` and `getTuple`, and a `skipTo` interface is also provided to adjust the internal offset for chunk column access. We refer interested readers to [19] for more details.

5.2.1 Implementation

We model the time window attribute as an object with five interfaces, namely, `getSlices`, `getMaxTime`, `getMinTime` and `getValue`, which naturally reflect the operations on time window attributes that are required by the cohort operators. The `getSlices` returns the number of time slices of the current user, and the `getMaxTime` and `getMinTime` interfaces respectively returns the upper and the lower delimiters of the constituent time slices of a given sliding position, as recorded in **windowT**. The `getValue` interface returns the value taken by the time window attribute at the giving position. To speed up `getValue`, we introduce an `initWindow` interface to pre-compute the aggregated value for each time slice, and implement the `getValue` interface by combining together the values of the constituent time slices. In this way, we remove the duplicate evaluations in the cases where there are multiple constituent time slices for each position during the sliding path.

The native implementation of the operators can be found in Algorithm 1. The main entrance of the algorithm is processUser. For causative behavior measurement, we first apply $\mathbb{M}^c.\mathcal{C}^e$ to select the qualified activities of the current user (line 2–5), and then call the `initWindow` interface to evaluate $\mathbb{M}^c.\mathbb{A}$ and each time window attribute involved in $\mathbb{M}^c.\mathcal{C}^w$ for each time slice. Hence, the window slicing operator γ is also implemented inside `initWindow`. Thereafter, for each qualified sliding position of the time window $\mathbb{M}^c.\mathbb{A}.\mathbb{W}$, we compute the causative behavior and allocate the current user to the respective cohort (line 12–14). For dependent behavior measurement, we also first evaluate all the involved time window attributes for each time slice (line 16). After that, $\mathbb{M}^d.\mathcal{C}^w$ is applied to prune unqualified sliding positions of the time window $\mathbb{M}^d.\mathbb{A}.\mathbb{W}$ (line 17–20). Finally, for each cohort that the current user enters, the behavior measured for each following (qualified) sliding position is then used to update the respective dependent behavior (line 21–25).

5.3 Discussion

The intrusive evaluation strategy benefits a lot from the customized layout of the user activity table **eventT** where the activities of each user are clustered together in chronological order. This layout enables the `initWindow` interface to be fulfilled within two or three scans from the first activity to the last activity of the current user. First, the window slicing operator scans user activities to find the delimiters of time slices. In the case of dimensional lifetime partition, the delimiters can be found by comparing the values between the current and the previous activity during scanning, and

Algorithm 1: Operator evaluation

Input : A data chunk D, and two positions start and end which respectively represent the first activity of the current user and the next user

```
1  initUser(D, start, end, C^e, A)
       // selection operator for C^e
2      D.skipTo (start), T ← ∅
3      for i ← start to end do
4          if C^e(D.getNextTuple()) then
5              T ← T ∪ i
       // window slicing operator
6      for A ∈ A do
7          A.initWindow (T, D)
8  processUser(D, start, end, M^c, M^d, A^g)
       // causative behavior measurement
9      A^c ← {A|A is involved in M^c.C^w}
10     initUser(D, start, end, M^c.C^e, {M^c.A} ∪ A^c)
11     cohorts ← ∅
12     for i ← 1 to M^c.A.getSlices() do
           // selection operator for M^c.C^w
13         if M^c.C^w({A.getValue(i)|A ∈ A^c}) then
14             cohorts ←
                 cohorts ∪ (M^c.A.getValue(i), M^c.A.getMaxTime(i))
       // dependent behavior measurement
15     A^d ← {A|A is involved in M^d.C^w}
16     initUser(D, start, end, M^d.C^e, {M^d.A, A^g} ∪ A^d)
17     slices ← ∅
18     for i ← 1 to M^d.A.getSlices() do
           // selection operator for M^d.C^w
19         if M^d.C^w({A.getValue(i)|A ∈ A^d}) then
20             slices ← slices ∪ i
21     for c ∈ cohorts do
22         for i ∈ slices do
23             if c.time < M^d.A.getMinTime(i) then
                   // age-by operator
24                 a ← A^g.getValue(i)
                   // recurrent aggregation operator
25                 combine metrics[c, a] with M^d.A.getValue(i)
```

hence the database join required in the SQL-based approach is no longer needed. Depending on whether $\mathcal{C}^e$ is empty on not, another scan may be needed to select activities of interest (line 3–5 of Algorithm 1). The final scan is to aggregate the selected activities of each time slice.

As such, it can be easily drawn from Algorithm 1 that its time complexity is $O(N_u + |S_u|^2)$, where N_u is the number of activities user u performed, and S_u is the set of time slices as defined in Definition 2. In contrast, the time complexity of the SQL-based approach is $(N_u \times |S_u|)$, as each activity of u should be taken into consideration to measure the dependent behavior of each cohort that u previously entered, as shown in Definition 12. Since $N_u \gg |S_u|$, the performance of the intrusive strategy is likely much better than that of the SQL-based approach, as demonstrated in Section 7. Moreover, thanks to the columnar chunk storage and the compression techniques employed, the scanning of activities is very efficient, which further improves the performance of the intrusive strategy.

6. DISTRIBUTED PROCESSING

In the intrusive strategy, all activities of each user are clustered within a single chunk of the hierarchical storage layout, and can be processed independently of the activities

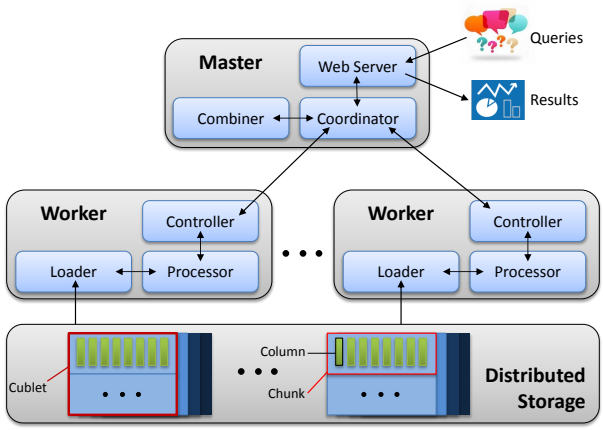


Figure 4: The architecture for distributed recurrent cohort analysis queries

of other users as a result of user-oriented cohort algorithms proposed in Section 5.2. Therefore, different chunks can be processed in parallel without any synchronization between them, which renders the intrusive strategy extremely suitable for distributed processing.

There have been many frameworks for distributed computation [13, 14, 15, 30, 31]. Such frameworks, however, cannot be used for our purpose as the cohort operators cannot be readily mapped into their computation models. Also, these frameworks employ complicated mechanisms to coordinate computation and data flow among computing nodes. Such mechanisms, however, are not necessary in the distributed evaluation of the intrusive strategy, as there is no need for synchronization or communication during the evaluation. As a result, using existing frameworks for distributed evaluation is too heavy-weight, which motivates us to design a system architecture, as shown in Figure 4.

There are two layers in this architecture. The bottom layer is a distributed storage to persist the cublets, each composed of roughly the same number of compressed chunks. The upper layer is a cluster of multiple workers and a master, managed by ZooKeeper [2]. The master is elected from workers, and is responsible for allocating cublets and coordinating query processing among workers. For cublet allocation, the master employs a round-robin policy so that the load on each worker is balanced due to the similar number of chunks in each cublet. Upon receiving a query (task), the master simply broadcasts it to all workers for processing and then waits for replies. Once it collects all the replies for a query, it then combines them and sends the combined result to the query client. Currently, we only allow distributive and algebraic aggregate functions to be used in time window attributes, so the combination of query replies is trivial.

The functionalities of workers are rather simple. Each worker loads the cublets allocated to it from underlying distributed storage into local memory in advance, and uses the in-memory copies for the processing of received queries by executing the algorithm proposed in Section 5.2 against user activities contained in its allocated cublets.

When the result set is large, the combination of query replies may make the master become bottleneck. This can be addressed by distributing the combination among workers such that each worker is responsible for the combination

of a disjoint subset of cohorts. The master pulls together the combined results and returns them to the query client without further processing.

Once a failure of a worker is detected by ZooKeeper, the master will get notified and then handle this failure. It does so by first reallocating the cublets that were allocated to the failed worker to several alive workers, and then re-broadcasting the pending queries that were received before detection of the failure to all workers. This can lead to multiple replies for a same query being received by the master. We handle this problem by assigning a unique query id to each query, and a message id to each message that a worker sends to the master. Upon detecting two replies from the same worker for a same query, the master simply chooses the reply with the larger message id, and discards the other one.

When the master crashes, the ZooKeeper service will elect a new master from alive workers. The newly elected master then contacts each worker to establish the correspondence between workers and cublets, and reallocates the cubelets that were allocated to the previous master among the alive workers. Thereafter, it starts the web service to accept queries. For now, we do not resume the queries that were pending when the previous master crashed, and rely on the query clients to resubmit those queries upon timeout.

7. PERFORMANCE STUDY

In this section, we conduct a set of experiments to study the performance of the two evaluation strategies. For the SQL-based strategy, we use two state-of-the-art database systems, MonetDB and PostgreSQL, which respectively represent the two major database genres, i.e., column-oriented databases and row-oriented databases. We implement the proposed operators by following the table transformation shown in Figure 2 and translating the respective definitions into standard SQL statements. The tables that are involved in the transformation are created as database views so that the query optimizer of the two databases can have more opportunities to optimize the physical query plan.

To compare the two strategies, we evaluate the single-node performance of the three systems, i.e., MonetDB, PostgreSQL and COHANA[2], by running four queries against datasets of different sizes. Furthermore, we conduct an additional experiment in distributed settings to investigate the scalability of the intrusive approach, in comparison with Spark SQL [10]. All servers included in both the single-node and distributed experiments are equipped with 8GB RAM and one quad-core Intel Xeon E3-1220 3.1GHz processor and run CentOS 6.6.

7.1 Dataset and Queries

The original dataset used in our experiments contains roughly 30M activities which were performed by 57K game users between 2013-05-21 and 2013-06-26. Figure 5 characterizes this dataset in terms of distribution in user lifetime and number of activities. As can be observed, 60% of users were active for only one day, and users performing less than 100 activities account for the same percentage.

We further use this dataset to generate three more datasets by either removing users and their activities or adding new users which have the same activities as original users. In total, there are four datasets which respectively have roughly 7.5M, 15M, 30M and 60M user activity records. All four datasets possess the same characteristics as shown in Figure 5, and are termed as the tiny, small, medium and large datasets.

[2]We use COHANA to represent the intrusive evaluation of the proposed cohort operators.

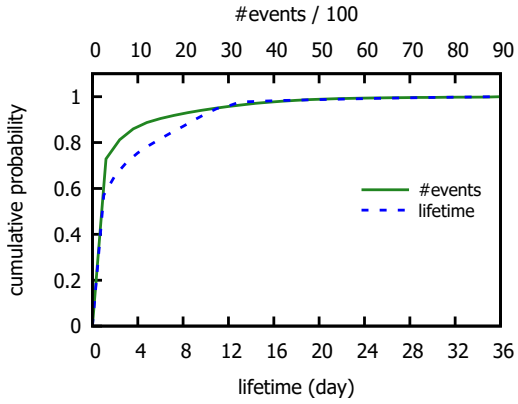


Figure 5: The characterization of the dataset in terms of user lifetime and the number of activities

We design the following four recurrent cohort analysis tasks for our performance study.

Query 1 Group users into cohorts based on the money spent for each shop event, and show for each cohort the 7-day retention thereafter.

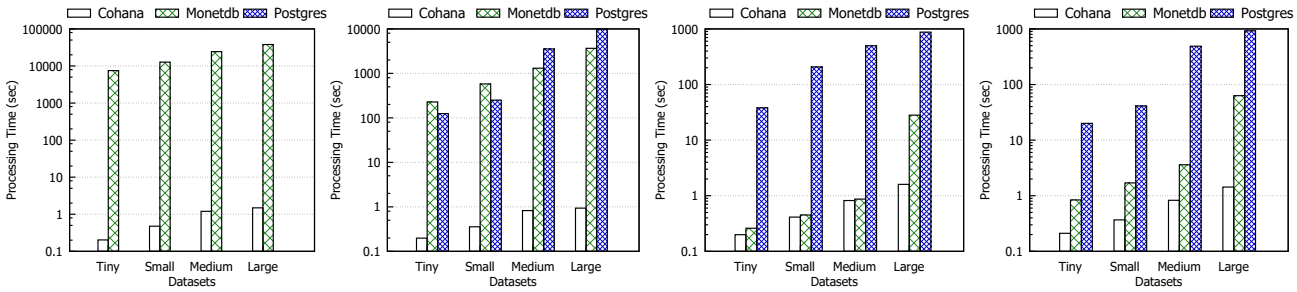
Query 2 Group users into cohorts based on the number of events performed each day, and show for each cohort the average 7-day spending thereafter.

Query 3 For each user, select each week during which she performed at least one shop event, and allocate her to the cohort based on the average spending up to that week. Then, show the per-day retention for each cohort.

Query 4 For each role, show the distribution of roles that would be played within 3 role changes. Only the roles with at lease one shop event performed are taken into consideration during the computation of distribution.

For Query 1, we want to find the relationship between user expense and retention rate. For Query 2, we want to explore how user expense is affected by daily activeness. For Query 3, we want to validate whether users with more spending are more likely to retain. The time window attribute for causative behavior measurement has a time slice range of $[1, 0]$, which includes all time slices up to the current one. For Query 4, we want to mine the pattern among role changes. A dimensional lifetime partition is used in the measurement of both the causative and the dependent behaviors: each change in the role attribute terminates the current time slice and launches a new one as well. Unlike the other three queries, the propositional formula $\mathcal{C}^w$ to select qualified time slice positions for dependent behavior measurement is not empty, and the age of this query is defined with respect to a time window attribute which extracts the value of the role played in the current time slice.

To speed up the SQL approach, we implement the optimization that we present in Section 5.1 by materializing the



(a) Q1 (b) Q2 (c) Q3 (d) Q4

Figure 6: Single-node evaluation

Table 2: SQL Optimization Speedup

Dataset	MonetDB		PostgreSQL	
	Q3	Q4	Q3	Q4
Tiny	38.46	19.05	1.29	2.50
Small	42.22	27.06	1.20	4.51
Medium	37.93	61.11	1.27	2.13
Large	2.74	11.49	1.27	3.01

respective time slice boundaries for Q3 and Q4. For Q1, this optimization does not apply, as the time slices are partitioned based on a selection predicate, i.e., the event-based partition. In addition, we do not need to materialize for Q2, since the time attribute of the four datasets is in date granularity, and already serves the purpose of materialization. Table 2 shows the effectiveness of this materialization optimization. As can be observed, the performance gain resulted from the materialization can acheieve 60×, and is hence very promising.

7.2 Single Node Evaluation

For this experiment, we study the performance of the three systems by running the four queries against the four datasets. As can be observed from Figure 6, the intrusive approach can process different queries very efficiently. For example, it can process all four queries in no more than one second for all datasets other than the large one. In addition, the performance of the intrusive approach is very steady and does not vary much across different queries. In contrast, the database approach needs a much longer (up to 6 orders) time than the intrusive approach, and depending on queries, their performance can vary significantly. This is because, as given in Section 5.1.1, the SQL-based approach has to join the cohort table (i.e., **cohortT**) generated with the activity table, which incurs a much higher time complexity than the intrusive approach which processes each user individually and hence avoids the expensive joining process.

Since the cohort tables generated for different queries have significantly different numbers of tuples, the time spent in joining them with the activity table also varies, which explains the varying performance of the database approach when evaluated for different queries. For Query 1[3], as a result of a simple event selection condition, its cohort table is much lager than the table generated for other queries, which makes the joining operation run out of memory, and

[3]The result of PostgreSQL for Q1 is absent, as it takes such a long time that we are not able to wait for its completion.

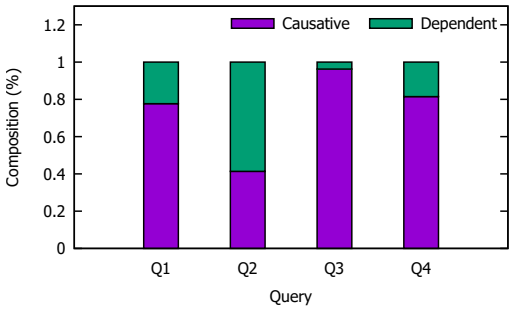


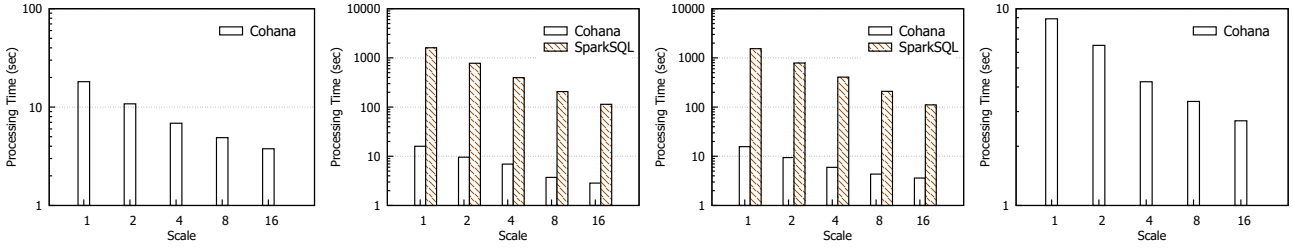
Figure 7: Time breakdown

hence incurs high IO cost. For Q3, MonetDB can achieve a similar performance to the intrusive approach for tiny-to-medium datasets. This can also be attributed to the cohort table generated. In this case, as the number of cohort entries of each user equals the number of weeks during the life time of that user, which is at most 6, the cohort table hence is of a very small size.

Figure 7 shows for each query the time breakdown between the causative behavior measurement and the dependent behavior measurement. The medium dataset is used for this experiment. The results of the other datasets are similar, and hence are omitted. It can be shown from this figure that the measurement of causative behaviors takes a longer time in all queries, which seems counter intuitive as the dependent behavior of each sliding position should be measured for each causative behavior (i.e., cohort), that took place earlier. This is attributed to our `initWindow` implementation which pre-computes all involved time window attributes, and hence eliminates the duplicate measurements of dependent behaviors. In addition, in the measurement of dependent behaviors, only the activities performed after the first causative behavior need be taken into consideration. This explains the negligible time spent in the dependent behavior measurement of Q3, since, as shown in Figure 5, only 20% of users are active after the first week.

7.3 Distributed Evaluation

This section presents the performance result of the intrusive strategy in distributed environments. The number of workers is varied from 1 to 16 and the dataset used for this experiment consists of roughly 480M activity records, and has a size of 64GB. For comparison purpose, we also deploy a Spark cluster to run Spark SQL. The master and driver of the Spark cluster are deployed at two different nodes, and the executors which conduct the real analysis are deployed at



(a) Q1 (b) Q2 (c) Q3 (d) Q4

Figure 8: Distributed evaluation

other nodes which amount to the same number of the workers used for the intrusive approach. Each Spark component is allowed to occupy almost all free memory (∼7 GB) of the host node. For better performance, we convert the dataset into ORC (Optimized Row Columnar) format, which also employs various optimization techniques, such as compression of COHANA. The ORC file is then stored in a HDFS cluster that is deployed in the same nodes as executors to exploit data locality.

The results are given in Figure 8. For Q1 and Q4, Spark SQL is not able to run to the end due to out of memory, the results of which are hence not available. As shown in this figure, initially, the performance of the intrusive strategy is improved almost linearly with the number of workers, showcasing the scalability of the intrusive approach. However, from 8 workers onwards, the performance gain from more workers gradually decreases. This is because, as we mentioned in Section 6, the combination of the result sets returned by workers starts to dominate the whole processing in these cases, especially for Q1 which has the largest number of cohorts and hence requires a lot of time for combining. Fortunately, this can be simply addressed by distributing the combining process among all workers, as mentioned in Section 6. Spark SQL also has a good scalability in that its processing time halves when the number of executors doubles. However, its performance is much (two orders) worse than the proposed intrusive approach, and even its best performance achieved in the setting of 16 nodes is still 6x worse than the performance of the intrusive approach achieved in the single-node case.

8. RELATED WORK

The traditional cohort analysis [17] is used to find how behavior of users (typically human being) is affected by two factors: social change and aging. It has been widely applied to many areas, such as social research [25], demographic study [26] and health care [16, 18]. Statistical methods have also been used to estimate this effect [12, 21].

Recently, due to the increasing volume of web user behavioral data, cohort analysis has been introduced to find unusual user behavior from such data and improve user retention [1, 3, 4, 29]. These products directly follow the single birth event specification and temporal age definition, which significantly restrict the diversity and representativeness of the cohorts that can be generated, and rigidify the way for measuring user behavior. Furthermore, the requirement of an event attribute also restricts the application spectrum.

The database support for the traditional cohort analysis was presented in [19, 29]. Basically, this work proposes three cohort operators for cohort analysis and implements them in COHANA. However, this work also follows the single birth event specification and temporal age definition, and hence faces the same problems as those commercial tools [1, 3, 4].

The system where we implement the proposed operators adopts a columnar design [11, 23, 27] and various compression techniques [28, 32] for the storage of chunks. Hence, our intrusive implementation of the proposed operators also follows the spirit of query processing over compressed columns [7, 9, 22]. The intrusive approach is suitable for distributed evaluation. However, existing distributed processing systems [13, 14, 15, 30, 31] incur a lot of unnecessary overhead. We hence propose and implement a light-weight distributed architecture for this purpose.

9. CONCLUSION AND DISCUSSION

This paper presents recurrent cohort analysis, which is a powerful tool for temporal dependence exploration, and can be used in many application domains. To address such analysis problems, we then present a set of cohort operators which capture the essential operations involved in the analysis. We further present two approaches for operator evaluation by respectively translating the operators into standard SQL statements and implementing them natively inside COHANA. A system architecture is then proposed for distributed processing of the latter approach. Finally, a comprehensive experimental study is performed to compare the two evaluation approaches in both single-node and distributed settings.

There are many interesting problems worth researching for recurrent cohort analysis. First, since the non-intrusive SQL approach does not work well, it is desired to natively implement the proposed cohort operators inside existing database systems. Another work is to support other data models, such as star and snowflake schemas. In addition, there is a large space to explore within the framework that we present for recurrent cohort analysis in this paper. For example, it might be interesting to permit users to leave cohorts upon satisfying certain triggering criteria.

10. ACKNOWLEDGEMENTS

This research was in part supported by the National Research Foundation, Prime Ministers Office, Singapore, under its Competitive Research Programme (CRP Award No. NRF CRP8-2011-08). Zhongle Xie's work was partially supported by the National Research Foundation Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme (E2S2-SP2 project). Gang Chen's work was supported by the National Basic Research Program (973 Program, No.2015CB352400). H. V. Jagadish's work was supported in part by NSF grant IIS-1250880.

11. REFERENCES

[1] Amplitude. https://amplitude.com.

[2] Apache zookeeper. https://zookeeper.apache.org/.

[3] Retention. https://mixpanel.com/retention/.

[4] Rjmetrics. https://rjmetrics.com/.

[5] Top 10 best stock market analysis software review 2018. https://www.liberatedstocktrader.com/top-10-best-stock-market-analysis-software-review/.

[6] Use the cohort analysis report. https://support.google.com/analytics/answer/6074676?hl=en.

[7] D. Abadi, S. Madden, and M. Ferreira. Integrating compression and execution in column-oriented database systems. In *SIGMOD*, pages 671–682, 2006.

[8] K. F. Adams, G. C. Fonarow, C. L. Emerman, T. H. LeJemtel, M. R. Costanzo, W. T. Abraham, R. L. Berkowitz, M. Galvao, and D. P. Horton. Characteristics and outcomes of patients hospitalized for heart failure in the united states: rationale, design, and preliminary observations from the first 100,000 cases in the acute decompensated heart failure national registry (adhere). *American heart journal*, 149(2):209–216, 2005.

[9] S. Amer-Yahia and T. Johnson. Optimizing queries on compressed bitmaps. In *VLDB*, pages 329–338, 2000.

[10] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, et al. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1383–1394. ACM, 2015.

[11] P. A. Boncz, M. Zukowski, and N. Nes. Monetdb/x100: Hyper-pipelining query execution. In *CIDR*, pages 225–237, 2005.

[12] N. E. Breslow, J. Lubin, P. Marek, and B. Langholz. Multiplicative models and cohort analysis. *Journal of the American Statistical Association*, 78(381):1–12, 1983.

[13] Q. Cai, W. Guo, H. Zhang, D. Agrawal, G. Chen, B. C. Ooi, K.-L. Tan, Y. M. Teo, and S. Wang. Efficient distributed memory management with rdma and caching. Technical report, National University of Singapore, Department of Computer Science, 2018.

[14] Q. Cai, H. Zhang, W. Guo, G. Chen, B. C. Ooi, K. L. Tan, and W. F. Wong. Memepic: Towards a unified in-memory big data management system. *IEEE Transactions on Big Data*, 2018.

[15] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[16] D. Donnell et al. Heterosexual hiv-1 transmission after initiation of antiretroviral therapy: a prospective cohort analysis. *The Lancet*, 375(9731):2092–2098, 2010.

[17] N. D. Glenn. *Cohort Analysis*. Sage Publications, Inc., London, 2005.

[18] E. A. Hoste, G. Clermont, A. Kersten, R. Venkataraman, D. C. Angus, D. De Bacquer, and J. A. Kellum. Rifle criteria for acute kidney injury are associated with hospital mortality in critically ill patients: a cohort analysis. *Critical care*, 10(3):1, 2006.

[19] D. Jiang, Q. Cai, G. Chen, H. V. Jagadish, B. C. Ooi, K.-L. Tan, and A. K. H. Tung. Cohort query processing. *PVLDB*, 10(1):1–12, 2016.

[20] Y. Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 447–456, New York, NY, USA, 2009. ACM.

[21] L. L. Kupper, J. M. Janis, A. Karmous, and B. G. Greenberg. Statistical age-period-cohort analysis: a review and critique. *Journal of chronic diseases*, 38(10):811–830, 1985.

[22] Y. Li and J. M. Patel. Bitweaving: Fast scans for main memory data processing. In *SIGMOD*, pages 289–300, 2013.

[23] S. Manegold, P. A. Boncz, and M. L. Kersten. Optimizing database architecture for the new bottleneck: Memory access. *The VLDB Journal*, 9(3):231–246, 2000.

[24] R. M. Martin, P. N. Biswas, S. N. Freemantle, G. L. Pearce, and R. D. Mann. Age and sex distribution of suspected adverse drug reactions to newly marketed drugs in general practice in england: analysis of 48 cohort studies. *British journal of clinical pharmacology*, 46(5):505–511, 1998.

[25] W. M. Mason and S. Fienberg. *Cohort analysis in social research: Beyond the identification problem*. Springer Science & Business Media, 2012.

[26] J. G. Pope. An investigation of the accuracy of virtual population analysis using cohort analysis. *ICNAF Research Bulletin*, 9(10):65–74, 1972.

[27] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik. C-store: A column-oriented dbms. In *VLDB*, pages 553–564, 2005.

[28] T. Westmann, D. Kossmann, S. Helmer, and G. Moerkotte. The implementation and performance of compressed databases. *SIGMOD Record*, 29(3):55–67, 2000.

[29] Z. Xie, Q. Cai, F. He, G. Y. Ooi, W. Huang, and B. C. Ooi. Cohort analysis with ease. In *ACM SIGMOD Demo*, 2018.

[30] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. pages 2–2, 2012.

[31] H. Zhang, G. Chen, B. C. Ooi, K. L. Tan, and M. Zhang. In-memory big data management and processing: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1920–1948, July 2015.

[32] M. Zukowski, S. Heman, N. Nes, and P. Boncz. Super-scalar RAM-CPU cache compression. In *ICDE*, pages 59–70, 2006.