Technical Disclosure Commons

Defensive Publications Series

September 22, 2017

# Incremental Evaluation of Complex Conditions

Madhu Kallazhi Vasu

Nagaraju Pothineni

Bin Liu

# Incremental Evaluation of Complex Conditions

The present disclosure describes methods that can save significant time and processing power when running analytics on large quantities of data, such as advertising and commerce data. Billions of people use the internet and each may visit a variety of websites on any given day. Attempting to sort through all of that data and evaluate which users or visits meet a complex condition, such as visiting a golf equipment shopping page in the last week, can be like trying to find a needle in petabytes of data. The methods described in the present disclosure use an incremental evaluation technique that breaks down complex conditions and data into smaller pieces in order to evaluate and store intermediate results. As a result, less time and processing power is needed to produce a final response to queries regarding the data.

A visitor condition can be used to search through a collection of visitor data and determine qualified visitors. The visitor condition may be composed of a variety of unique segments. Each segment may be used for a specific purpose and can be evaluated using shortcuts and intermediate results. Evaluation of the visitor condition as a whole simply requires a cross-visit AND of all segments. A more detailed description regarding how these segments can work together to form a simple visitor condition is presented below.

The visitor condition may include two visitor segments, include_criteria and exit_criteria, each of which is a UnifiedSegment (see Figure 1 below) with a visitor_segment but no visit_segment. When evaluating a visitor condition, the exit_criteria segment takes priority. If exit_criteria is met by a visitor, the visitor is no longer a member of the audience even if include_criteria is also met. A visitor's unit of data can be referred to as a visit, or session, which contains multiple activities in a unit of time. A visitor may have multiple visits on any given day. The visitor condition applies to all visits for each visitor during a lookback window.

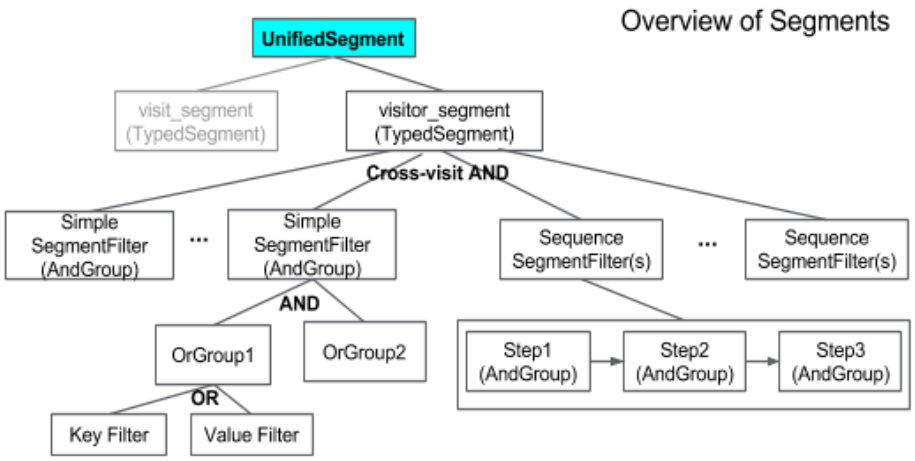


Figure 1: Overview of Segments

As shown in Figure 1, a UnifiedSegment can contain two TypedSegments: one visit_segment and one visitor_segment. For audience definition and evaluating the visitor condition only the visitor_segment is used. A TypedSegment can include multiple SegmentFilters of two different types: Simple and Sequence. UnifiedSegments can contain multiple SegmentFilters AND-ed together in the form: $Expr = F_0 \text{^} F_1 \text{^} F_2$ (where $F_n$ represents a SegmentFilter). A Simple SegmentFilter can be of the form:

$F = (Filter_1\ OR\ Filter_2)\ AND\ (Filter_3\ OR\ Filter_4)$, where each filter can be a key or value filter. Any of the value filters can be visitor-scoped value filters which cannot be evaluated on just a single visit. A Sequence SegmentFilter can contain multiple ordered Steps, each of which can be evaluated in a single visit, but, in some implementations, the whole Sequence may require more than one visit to be evaluated. Complements can be used with any filters requiring that the filter cannot be true for any visits. Multiple SegmentFilters contained in an overall UnifiedSegment may require multiple visits to evaluate. It can become prohibitively processor-intensive to use all visits to evaluate a visitor condition when the lookback window extends for multiple months.

Incremental Evaluation

Space can be traded for time by storing and reusing intermediate evaluation results of SegmentFilters. In some implementations, evaluation results can be stored for each SegmentFilter as soon as it can be evaluated to true and avoid evaluating it again. Simple SegmentFilters can be marked as true for some visits and Sequence SegmentFilters can be marked as true for visits up to a point. In some implementations, a result is stored for each SegmentFilter and for each day within a time window. For example, suppose visit data is gathered from day 0 to day 15 (starting at 12AM on day 0) and visitor conditions need to be evaluated on a lookback window of 5 days. There is enough data gathered to evaluate the visitor condition on the morning of day 5, where evaluation requires intermediate results for each day from day 0 to day 4 along with any new data after that. Intermediate evaluation results can be stored in a proto, in some implementations.

Protocol buffer definitions that can be used to implement the methods described in the present disclosure are shown below.

```
message SegmentEvalStatus {
 optional int64 true_time = 1;
 repeated int32 matched_filters = 2;

 message VisitorScopeFilters {
  required int32 or_group = 1;
  repeated int32 filter = 2;
 }

 message StepStatus {
  required int32 step_index = 1;
  repeated string reference = 2;
  repeated string value = 3;
  optional VisitorScopeFilters remaining_filters = 4;
 }

 message SequenceStatus {
  repeated StepStatus step_status = 1;
```

```
 }

 map<int32, SequenceStatus> sequence_progress_map = 3;
 map<int32, VisitorScopeFilters> remaining_filters = 4;
}

message AudienceEvalStatus {
 optional SegmentEvalStatus include_criteria_status = 1;
 optional SegmentEvalStatus exit_criteria_status = 2;
}
```

Simple SegmentFilters

The most simple, and also most common, UnifiedSegment includes only Simple SegmentFilters. In this case, each SegmentFilter can be evaluated once for each visit each day, and a bit can be stored if the SegmentFilter evaluates to true. Complements can be considered when the whole UnifiedSegment is evaluated. If a SegmentFilter is already true for a previous visit on the same day, there is no need to evaluate it for visits later in the same day. The intermediate result for each visitor can be written to a bigtable after processing all visits for a day. Audience evaluation results (whether a visitor is in an audience) can be written to the user attribute store. No intermediate results are stored for a visitor if there is no need to update the result from the previous day. All of the SegmentFilters can be AND-ed together over the lookback window and complements considered when the UnifiedSegment is evaluated.

When complements are included anywhere in the evaluation of a UnifiedSegment that contains only Simple SegmentFilters, additional evaluation shortcuts can be taken to optimize the process. The process can be optimized at two places: storing intermediate results and evaluating the UnifiedSegment. Complements can be specified in three places when evaluating a UnifiedSegment: the UnifiedSegment itself, the TypedSegment (AND of SegmentFilters), and on each SegmentFilter. It is unlikely that complements would be included on both the

UnifiedSegment and TypedSegment for the purpose of defining an audience (since they would negate each other).

The first case to consider includes complements at the SegmentFilter level but not at the TypedSegment or UnifiedSegment levels. In the example shown in Table 1 below, a Simple SegmentFilter with a lookback window of three days is shown. $F_0$ is complemented and evaluates to true on day 0. This means any lookback window including day 0 will be false and no other SegmentFilters need to be evaluated for that day. On the other hand, if there is a complement at the TypedSegment level, just the opposite would be true. A complemented TypedSegment that includes day 0 in the lookback window can be evaluated to true the moment complemented SegmentFilter $F_0$ evaluates to false. Taking advantage of complements to shortcut the evaluation in this manner can save significant space and processing time.

| | $F_0$ | $F_1$ | $F_2$ | $F_3$ |
|---|---|---|---|---|
| day 0 | True | True | True | |
| day 1 | | True | True | |
| day 2 | | | True | |
| day 3 | | | True | |

Table 1: $F = \overline{F_0}\text{^}F_1\text{^}F_2\text{^}\overline{F_3}$

Sequence SegmentFilters

A Sequence SegmentFilter can have multiple Steps, each of which is similar to a Simple SegmentFilter. A formula for this filter can take the form $F = S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_3$ where $S_i$ represents a Step in the Sequence. During evaluation, all visits for a visitor in the lookback window can be turned into a group of hit data and each Step can then be evaluated at hit level. Visit level metrics can be copied to each group of hit data for consideration. Different Steps need to be true at different hits. As an example, Step $S_i$ needs to be true for a hit that happened later

than a hit where Step $S_{i-1}$ is true. A Sequence can be evaluated to true for a visitor when all Steps evaluate to true for different hits satisfying the time order.

In order to use a sliding lookback window, the evaluation results for each Step need to be stored in time order every day their results change. Referring to the example in Table 2, a Sequence SegmentFilter of five Steps with a lookback window of 4 days is shown.

| | day 0 | day 1 | day 2 | day 3 | day 4 |
|---|---|---|---|---|---|
| Index of Steps that are true in time order | 0, 2, 1 | 0, 3 | 4, 1, 2 | 4, 3 | 4 |

Table 2: $F = S_0 \rightarrow S_1 \rightarrow \cdots \rightarrow S_4$

In simple cases, Table 2 can be stored as an intermediate result. The second row shows which Steps were evaluated true and the order in which they were evaluated as true. At the time of evaluating the whole Sequence SegmentFilter, the stored indices can be scanned from left to right in order to determine if the Steps are true in order across the whole lookback window. For example, consider the lookback window from day 0 to day 3 shown in Table 2. $S_0$ and $S_1$ are true, in order, on day 0. $S_2$ is true again on day 2, and $S_3$ is true again on day 3. $S_4$ is not true again until day 4, so the Sequence SegmentFilter as a whole is false from day 0 to day 3.

It is possible that Steps are true multiple times in a single visit. An example where this is true is shown in Table 3:

| | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ | $t_8$ | $t_9$ | $t_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $S_1$ | | | T | | | | T | | | |
| $S_2$ | | T | | T | | | | T | | |
| $S_3$ | T | | T | | T | T | T | | T | |

Table 3: Hit Data for a Visit

The columns in Table 3 represent different timestamps for a single visit. The timestamps where each Step is evaluated to true are shown in the rows. It appears that, in general, for a visit such as the one shown in Table 3, at most $n$ occurrences for the $n^{th}$ Step of the Sequence need to be stored. For example, $S_1$ at $t_3$ is always needed in case this is the first visit in the lookback window. $S_1$ at $t_7$ will never be needed since $S_1$ is already true at an earlier timestamp for the same visit. $S_2$ at $t_2$ may be needed even though it comes before the first $S_1$ during this visit because the lookback window may extend earlier than this visit where an $S_1$ could be used to start forming the Sequence. $S_2$ at $t_4$ will always be needed in case the lookback window begins with this visit, as it comes after the first $S_1$. $S_2$ at $t_7$ will never be needed. $S_3$ at $t_1$ may be needed, in case the lookback window includes $S_1$ and $S_2$ in order for previous visits. $S_3$ at $t_3$ may be needed, as it comes after the first $S_2$ during this visit, and may be used in conjunction with an $S_1$ from a previous visit in the lookback window. $S_3$ at $t_5$ will always be needed in order to complete the Sequence in this time window, and $S_3$ at $t_6$, $t_7$, and $t_9$ will never be needed. An occurrence for a Step in a sequence SegmentFilter is useful when and only when it is used to construct a Sequence in the lookback window, and the earliest occurrence is always stored. Most Sequence filters contain four Steps or fewer, so on each day at most ten occurrences need to be stored. The same logic can be applied to different visits across the same day.

An alternative to the approach described above with regard to evaluating Sequence SegmentFilters can be to store qualifying sub-Sequences for each day (or visit). This approach is more complex and seems less intuitive, however, the evaluation of Sequences becomes slightly simpler. Referring to the example in Table 2, the sub-Sequence [0,1] can be stored as an intermediate result for day 0. Similarly, the sub-Sequence [1,0] can be stored for day 1.

In cases where Steps in a Sequence refer to key or value filters used in a previous Step, the above methods described for evaluating the Sequence will no longer hold true. These cross-Step references require the storage of more information in order to be supported. Referring again to the example in Table 2, suppose that Step $S_3$ indicates that its visit_number needs to be larger than the visit_number in Step $S_1$, and that Step $S_4$ requires the page visited is different than the one used in Step $S_2$. Table 4 shown below details the additional information needed in these cases.

| | day 0 | day 1 |
|---|---|---|
| Index of Steps that are true, in time order, together with <key, value> pairs that are referenced in the future. | 0, [<visit_number, 0>,]<br>2, [<request_uri, index.html>,]<br>1, [] | 0, [<visit_number, 2>,]<br>3, [] |

Table 4: Cross-Step References

Visitor-scoped Value Filters

Recall that a SegmentFilter consists of OR groups AND-ed together such as $F = (Filter_1\ OR\ Filter_2)\ AND\ (Filter_3\ OR\ Filter_4)$. Each filter in the OR groups can be key or value filters, and value filters can include visitor-scoped metrics such as total number of hits in a lookback window. In some implementations, these visitor-scoped value filters cannot be evaluated on a single visit, however processing time can still be saved by making use of intermediate results. If any of the OR groups in the SegmentFilter do not contain visitor-scoped value filters, and the OR group can be evaluated as false, then no other information is needed and the whole SegmentFilter can be evaluated as false. If an OR group does contain at least one visitor-scoped value filter, but it also includes at least one regular (non-visitor-scoped) filter, and a regular filter can be evaluated to true, then the whole OR group can be evaluated to true and no other information is needed. If the OR group contains no regular filters, or none of them can be

evaluated to true, then the OR group relies upon at least one visitor-scoped value filter. When this is the case, an AND group is stored containing all visitor-scoped value filters needed to evaluate the SegmentFilter with all other filters are removed. Indices of OR groups and visitor-scoped value filters can be stored in the remaining_filters field for StepStatus and SegmentEvalStatus, for Sequence and Simple SegmentFilters, respectively. The SegmentFilter or Step can then be evaluated solely based on remaining_filters, which can be updated in intermediate storage after new visits are processed.

When evaluating the whole TypedSegment the SegmentEvalStatus proto can be checked. The remaining_filters should already be up to date from the previous day, and those filters can be decided together with data from the latest day. A filter may be on lifetime metrics such as number of transactions, for which for which the latest visit data can be combined with data from the user attribute store. The user attribute store can be populated at the end of each day with the latest computed lifetime metrics used to evaluate any visitor-scoped value filters. After all visitor-scoped value filters are evaluated, the truth of all Simple and Sequence SegmentFilters can be evaluated along with the entire audience.

Storage and Reuse of Intermediate Results

Information needed for incremental evaluation can be captured from the SegmentEvalStatus proto. A proto for each audience for each visitor can be stored in a bigtable referred to as an Intermediate State Table (IST). The proto does not need to be stored for a visitor on a day when there are no visits. Intermediate results for an audience are only stored if they are useful for evaluation (some filter evaluates to true).

## Abstract

Evaluating a complex condition such as users who have visited a certain type of website in the last week requires a great deal of processing power. The present disclosure describes an incremental evaluation technique that makes use of shortcuts and intermediate results in order to reduce the amount of time and processing needed to produce a final result. The main component of a visitor condition as described is a unified segment which can contain any number of simple or sequence type segment filters. Each segment filter can be evaluated using shortcuts and intermediate results. The evaluation of the entire unified segment simply requires a cross-visit AND of all segment filters. Making use of simpler evaluations on much smaller sets of data can be an effective approach to data analytics at a large scale.