Finding the Plateau in an Aggregated Time Series

Min Wang[1] and X. Sean Wang[2]

[1] IBM T. J. Watson Research Center
Hawthorne, NY 10532, USA
min@us.ibm.com
[2] Department of Computer Science, The University of Vermont
Burlington, VT 05405, USA
xywang@cs.uvm.edu

Abstract. Given d input time series, an aggregated series can be formed by aggregating the d values at each time position. It is often useful to find the time positions whose aggregated values are the greatest. Instead of looking for individual top-k time positions, this paper gives two algorithms for finding the time interval (called the plateau) in which the aggregated values are close to each other (within a given threshold) and are all no smaller than the aggregated values outside of the interval. The first algorithm is a centralized one assuming that all data are available at a central location, and the other is a distributed search algorithm that does not require such a central location. The centralized algorithm has a linear time complexity with respect to the length of the time series, and the distributed algorithm employs the Threshold Algorithm by Fagin et al. and is quite efficient in reducing the communication cost as shown by the experiments reported in the paper.

1 Introduction

Given a set of d input time series, by aggregating the d values at each time position, we obtain an aggregated time series A. A top-k query is to determine the top k time positions on A, namely, the time positions with the k greatest aggregated values. The well-known threshold algorithm (TA) [2] may be used to answer this type of query.

Recently, there has been active research on data aggregation in sensor networks [5, 6, 7, 1] and the top-k query can be very useful. For example, in an environmental monitoring system, multiple sensors may be used in an interested area to measure the temperature, humility, etc., at every minute. The measured data are stored in these sensors, and the system may need to find, within a specific time period, the time positions with the k highest average temperatures [3].

Assume the aggregated time series contains the average temperature for each minute during the past week and $k = 3$. If Friday is the warmest day during the week and the highest temperature during the week is at 1:30pm on Friday, we may very likely get the following three time positions as the answer to our top-3 query: 1:29pm on Friday, 1:30pm on Friday, and 1:31pm on Friday.

J.X. Yu, M. Kitsuregawa, and H.V. Leong (Eds.): WAIM 2006, LNCS 4016, pp. 325–336, 2006.

We believe that a more interesting query is to find the *plateau* over the aggregated time series. The plateau is defined as the maximum interval such that all the values on the time positions in the interval are no less than all the values at the time positions outside of the interval. Compared to the top-k time positions, the plateau may give us more information. The plateau definition becomes more interesting and useful when we add another constraint: all the values in the plateau should be "close enough" to the top-1 value of the whole sequence. How close is "close enough" can be a value specified by the user.

In the example above, assume that the user considers two degrees as close enough, and asks for the plateau. The answer will be the interval [1:10pm on Friday, 1:45pm on Friday] if the temperature at each time position in this interval is at most two degrees lower than the highest temperature observed at 1:30pm on Friday, and all the time positions outside of this interval have temperature values no higher than the value of each time position in the interval. Obviously, the plateau carries more information about high-temperature time positions than that of the k time positions we get from a traditional top-k query.

In this paper, we formally define the plateau over time series and present efficient algorithms to find the plateau in both centralized and distributed settings. We show that the plateau can be found in linear time with respect to the length of time series in the centralized setting. For the distributed setting, we develop a distributed search algorithm and through experiments we show that it significantly outperforms a direct extension of the TA algorithm in terms of number of accesses to the distributed sources.

The rest of the paper is organized as follows. In the next section, we introduce some basic notions and formally define the key concept of *plateau*. Sections 3 and 4 describe our algorithms for finding the plateau in an aggregated time series in a centralized setting and a distributed setting, respectively. We present our experimental results in Section 5 and draw conclusions in Section 6.

2 Preliminary and Basic Assumptions

We first define time series. A *time series* is a finite sequence of real numbers and the number of values in the sequence is its *length*. We assume all time series are sampled at the fixed (discrete) time positions $t_1, \ldots, t_n$. A time series is denoted as s, possibly with subscripts, and its value at time t is denoted $s(t)$.

An *aggregated time series* is a time series whose value at time position t is from aggregating the values from multiple *input time series*. Specifically, given $s_1, \ldots, s_d$ and an aggregation function f, the aggregated time series is s_f with $s_f(t) = f(s_1(t), \ldots, s_d(t))$ for each t. We shall use A to denote aggregated time series, and omit the mentioning of function f when it is understood. A "normal" time series can be considered as a degenerated aggregated time series, and hence we shall use A to denote both "normal" time series and aggregated ones.

Definition 1. *Given a time series A and a real value ε, a time interval $[t_l, t_r]$ is said to be an* ε-plateau *of A if for each time position $t \in [t_l, t_r]$, we have (1) $|A(t) - A(t')| \leq \varepsilon$ for all $t' \in [t_l, t_r]$, and (2) $A(t) \geq A(t'')$ for all $t'' \notin [t_l, t_r]$.*

Intuitively, an ε-plateau in a time series is the largest time interval that has values no less than the value of any time position outside of the interval, and the difference between the values within the time interval is at most ε. An ε-plateau must contain a time position with the greatest value in the time series.

We abbreviate ε-plateau to simply plateau when ε is implicit or irrelevant. A *maximum ε-plateau* is an ε-plateau that is not a proper subinterval of another ε-plateau. In the sequel, when not explicitly stated and clear from the context, we will use the term plateau to mean the maximum plateau.

When there are more than one top-1 time position in the aggregated time series, two cases arise: all the top-1 time positions are either contiguous or not. In the former case, we will have only one maximum plateau. For the latter, the only (maximum) plateaux we will find are formed by top-1 time positions, regardless of the ε value. This is rather trivial algorithmically since it is equivalent to finding all top-1 time positions (and possibly combine these positions that are contiguous with each other). We do not pursue this case any further. Since the former case is equivalent to having a unique top-1 position, we will in the sequel assume that the top-1 position is unique in each aggregated time series, and hence we will have a unique maximum plateau for each ε value.

Example. Consider the maximum plateau in the (aggregated) time series shown in Fig. 1. The top-1 time position is $t_d = t_{10}$ with value 12. If $\varepsilon = 2$, then the plateau is $[t_9, t_{10}]$. If $\varepsilon = 10$, then the plateau is $[t_8, t_{11}]$. □

An equivalent way of defining a plateau is by a minimum value threshold τ. That is, instead of condition (1) in the definition, we would insist that $A(t) \geq \tau$ for all $t \in [t_l, t_r]$. Obviously, this is equivalent to the original definition if we take $\tau = A(t_m) - \varepsilon$, where t_m is a time position with the greatest value. In the sequel, we may use plateau to mean an ε-plateau or equivalently a plateau with a minimum value threshold.

We may also define the ε-plateau via the notion of rank as follows.

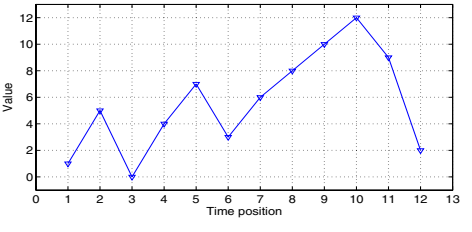

Fig. 1. Example time series

Definition 2. *Given a time series, the* top-rank*, or simply* rank*, of a time position t, denoted $R(t)$, is defined as* 1 *plus the number of time positions that have greater values, that is,* $R(t) = 1 + |\{t' | A(t') > A(t)\}|$.

If $R(t) \leq k$, we will also say that t is a *top-k time position*. Hence, a top-1 time position has a value that is no less than that of any other time positions.

Given a time series and real value ε, if $[t_l, t_r]$ is an *ε-plateau*, then for each time position $t \in [t_l, t_r]$, all the time positions with ranks higher (or $R()$ values smaller) than $R(t)$ must be in $[t_l, t_r]$. For example, if the plateau includes a rank 3 time position then all the rank 1 and rank 2 time positions must also be in the plateau.

Much has appeared in the literature for algorithms that look for top-k items from multiple data sources (e.g., [4]). Many algorithms use a variant of Fagin et al.'s threshold algorithm (TA), which has been shown to be optimal [2]. In TA, the aggregation function f is assumed to be *monotonic*, i.e., $x_1 \leq y_1, \ldots, x_d \leq y_d$ implies $f(x_1, \ldots, x_d) \leq f(y_1, \ldots, y_d)$. Many practical aggregation functions, like sum, average, maximum, are monotonic.

We now briefly review TA, as applied to look for top-k time positions in the aggregated time series. Assume we have d input time series $s_1, \ldots, s_d$. We sort each time series based on the values (from large to small), and keep the time position information with the values. Thus, we have d such sorted lists: $L_1, \ldots, L_d$. In TA, we proceed as follows.

1. Do sorted access in parallel (or using a round-robin schedule) to each of the d sorted lists. As a value v is retrieved under the sorted access from one list (assuming the associated time position is t), do random access to the other time series to find the values $s_i(t)$ for all i, and compute $A(t) = f(s_1(t), \ldots, s_d(t))$. We say time position t has been "seen" and $A(t)$ is kept for each "seen" time position t.
2. For each list L_i, let v_i be the last value returned under the sorted access. Define the *threshold value* θ to be $f(v_1, \ldots, v_d)$. Stop as soon as there are at least k distinct $A(t)$ values on the "seen" time positions that are greater than θ, and then output the top-k time positions among all the "seen" time positions.

3 Centralized Algorithm

In this section, we discuss how to find the plateau for an aggregated time series when all the input time series are available at a central point. For example, we can imagine each sensor in a sensor network sends its measurement data to the control center every hour. In such a setting, the central point can calculate the aggregated time series A based on the input time series and the given aggregation function. We present a linear time algorithm for finding the plateau on A.

We first define a *left ε-plateau* of the time series A to be an ε-plateau when we only consider the time series on the left of (and including) the top-1 time position. That is, it is an ε-plateau we find in $A(t_1), \ldots, A(t_m)$, where t_m is the top-1 time position. Right ε-plateaux are defined analogously. We define the maximum left and right ε-plateaux in a similar way as we defined the maximum ε-plateau, and use the term *left* and *right plateau* to mean the maximum left and right plateau, respectively, when the context is clear. Note, however, the union of a left and a right ε-plateaux does not necessarily form an ε-plateau as will be shown in the example at the end of this section.

The following result directly follows the definitions.

Theorem 1. *Denote* $min\_right(i) = min\{A(t_j) \mid i \leq j \leq m\}$, *and* $max\_left(i) = max\{A(t_j) \mid 1 \leq j < i\}$. *Interval* $[t_l, t_m]$ *($l \leq m$) is a left ε-plateau if and only if* $min\_right(l) \geq A(t_m) - \varepsilon$ *and* $min\_right(l) \geq max\_left(l)$ *where t_m is the top-1 time position.*

In the above theorem, we assume $max\_left(1)=-\infty$. We have an analogous theorem for the right ε-plateaux. These theorems give the basis for our linear time algorithm in finding the maximum left and right ε-plateaux. It is obvious that *min_right* and *max_left* for the time positions before t_m (and *min_left* and *max_right* for the positions after t_m) can be computed in an incremental fashion with two sequential scans, using for example the recurrence relation $min\_right(i) = min\{min\_right(i+1), A(i)\}$. Assume these values are available and assume $\tau = A(t_m) - \varepsilon$. Then we can easily design the procedure:

$$find\_left\_plateau\left([t_L, t_m], \tau\right) : [t_l, t_m], \tau_l$$

The input parameters are $[t_L, t_m]$ and τ, where t_L is the left boundary of the time series to be considered, t_m is the right boundary of the time series to be considered (t_m is also the top-1 position in $[t_L, t_m]$), and τ is the required minimum value threshold. The output parameters are $[t_l, t_m]$ and τ_l, where $[t_l, t_m]$ is the maximum left plateau and $\tau_l = max\left\{\tau, A(t_i) \mid i = l, \ldots, l-1\right\}$. The procedure simply scans from t_L towards t_m and finds the first time position t_l such that $min\_right(l) \geq max\_left(l)$ and $min\_right(l) \geq \tau$.

The correctness of this procedure follows Theorem 1 directly. It is also clear that the time complexity of the procedure is $O(l - L + 1)$.

The question now is how to get the global ε-plateau. Assume *find_left_plateau* and *find_right_plateau* return $[t_l, t_m]$ and τ_l, and $[t_m, t_r]$ and τ_r, respectively. By Theorem 1, all the positions in $[t_l, t_m]$ have values no smaller than τ_l while all the positions in $[t_1, t_l)$ have values no greater than τ_l. We have similar conclusions for $[t_m, t_r]$ and τ_r. If $\tau_l = \tau_r$, we can merge the left and right ε-plateaux to obtain the maximum ε-plateau. Otherwise, we should shrink the side with the smaller τ using the greater τ. This shrinking process is repeated until $\tau_l = \tau_r$ and we then merge the left and right ε-plateaux into the ε-plateau. The whole process is summarized in Fig. 2. The algorithm finds the maximum ε-plateau $[t_l, t_r]$ of time series A. It also returns a real value τ such that all the values in $[t_l, t_r]$ are no smaller than τ while all the values not in $[t_l, t_r]$ are no greater than τ.

Algorithm. *Find_Plateau*
Input: Time series A of length n, and ε.
Output: $[t_l, t_r]$: maximum ε- plateau
$\tau = max\{A(t_m) - \varepsilon, A(t_j) | t_j \notin [t_l, t_r]\}$, where t_m is the top-1 time position
(1) Find the top-1 time position t_m.Set $\tau = A(t_m) - \varepsilon$, and compute *min_right*, *max_left*, *min_left* and *max_right* as described earlier.
(2) Call *find_left_plateau*$([t_1, t_m], \tau)$. Return $[t_l, t_m]$, τ_l.
(3) Call *find_right_plateau*$([t_m, t_n], \tau)$. Return $[t_m, t_r]$, τ_r.
(4) Let $t_L = t_l$ and $t_R = t_r$. If $\tau_l = \tau_r$ then $\tau = \tau_l$. Return $[t_L, t_R], \tau$. Done.
(5) If $\tau_l > \tau_r$ then call *find_right_plateau*$([t_m, t_R], \tau_l)$. Return $[t_m, t_r]$, τ_r. Goto Step 4.
(6) If $\tau_l < \tau_r$ then call *find_left_plateau*$([t_L, t_m], \tau_r)$. Return $[t_l, t_m]$, τ_l. Goto Step 4.

Fig. 2. The *Find_Plateau* algorithm

Theorem 2. *Algorithm Find_Plateau correctly finds the ε-plateau of time series A in linear time and space.*

Proof. The space complexity of the algorithm is obvious since we only need to store two numbers for each time position. Now we analyze its time complexity. Steps 1-3 take linear time as mentioned earlier. The nontrivial part of the proof is that *find_left_plateau* and *find_right_plateau* may be called multiple times due to Steps 5 and 6. However, each repeated call to *find_left_plateau* will start the scan from the stopping position of the previous call. That is, even in the worst case, the multiple calls to *find_left_plateau* will scan up to m positions and thus the complexity of all calls is $O(m)$. Similarly, the complexity of all possible multiple calls to *find_right_plateau* is $O(n - m + 1)$. Hence, the time complexity of Algorithm *Find_Plateau* is $O(n)$.

The correctness follows the correctness of the procedures *find_left_plateau* and *find_right_plateau*. Indeed, with Steps 2 and 3, we find the respective maximum plateaux with $A(t_m) - \varepsilon$ as the minimum value threshold for the plateaux. It is clear that Steps 5 and 6 will both still return ε-plateaux. The question is whether the final result is the maximum ε-plateau. The answer is positive since each time we used smallest τ_l and τ_r value that is necessarily to maintain the combined interval to be a plateau. □

Example. We want to find the 10-plateau in the time series shown in Fig. 1. The top-1 time position is $t_m = t_{10}$ with value 12. Given $\varepsilon = 10$, we have threshold $\tau = 12 - 10 = 2$ initially. The call to *find_left_plateau*$([t_1, t_{10}], 2)$ returns with the maximum left plateau $[t_8, t_{10}]$ and $\tau_l = 7$, and *find_right_plateau*$([t_{10}, t_{12}], 2)$ returns with the maximum right plateau $[t_{10}, t_{12}]$ and $\tau_r = 2$. Note that we cannot combine the left and right plateaux into one yet since $\tau_l \neq \tau_r$ (actually, $[t_9, t_{12}]$ is not a plateau). Since $\tau_l > \tau_r$, so we call *find_right_plateau*$([t_{10}, t_{12}], 7)$. This time, it returns a new right plateau $[t_{10}, t_{11}]$ and a new $\tau_r = 7$. Now we can combine the left and right plateaux into $P = [t_8, t_{11}]$. We also output $\tau = 7$. □

4 Distributed Algorithm

In this section, we discuss how to find the plateau for an aggregated time series without bringing all the data into a centralized server. The reason for this may include the large size of the time series from the data sources, and the high communication costs. In this setting, we would like to calculate the ε-plateau with a minimum amount of communication. To do this, we assume that data sources have some computation power to support the local processing as required by the Threshold Algorithm (TA) of [2].

In the distributed setting, as required by the TA, we assume the aggregation functions are *monotonic*.

4.1 A Naive Algorithm

A straightforward way of finding the plateau in a distributed setting is to find the top-1 time position t_m in the aggregated time series, and then to find all the time positions whose aggregated values are no smaller than $A(t_m) - \varepsilon$.

The top-1 time position t_m and its aggregated value $A(t_m)$ can be found by a direct use of TA. We may trivially extend the TA algorithm to proceed, after finding top-1 time position, to repeatedly find the next top time positions and their associated aggregated values until the threshold θ is smaller than $A(t_m)-\varepsilon$. In this way, we find all the time positions with values no smaller than $A(t_m)-\varepsilon$. With these time positions, we can use our linear algorithm to find the maximum ε-plateau. Indeed, a little deeper analysis of the linear algorithm indicates that if we change all the values smaller than $A(t_m)-\varepsilon$ to a very small number (smaller than all possible values), then the plateau found by the linear algorithm is the same as the one found with all the data available.

4.2 A Distributed Search Algorithm

In some situations, the above naive algorithm performs very poorly. Indeed, consider the following aggregated time series of length n:

$$2, 2, \ldots, 2, 1, 3$$

and consider 1-plateau (i.e., $\varepsilon = 1$). Clearly, the top-1 time position is t_n, and the 1-plateau is $[t_n, t_n]$. However, the above naive algorithm will need to retrieve all the time positions t_1 through t_{n-2}, in addition to t_n. The run time and the communication cost will be proportional to n. A simple observation will yield that if we find out that the time position t_{n-1} has a value 1 that is lower than $A(t_n) - \varepsilon = 3 - 1 = 2$ and the greatest value between t_1 and t_{n-1} is 2, then we can safely conclude that $[t_n, t_n]$ is the plateau we are seeking.

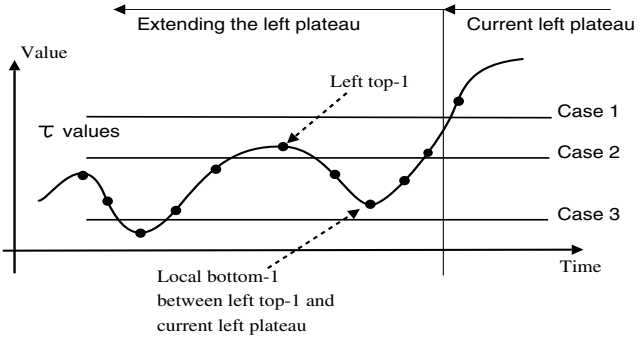

Fig. 3. Three cases for the distributed algorithm

Similar to the linear centralized algorithm in Section 3, we first concentrate on finding the left and right plateaux, separately, and then combine them into a single plateau. The above example is for the left plateau. Let us examine it a little closer with the help of the diagram in Fig. 3. In this diagram, the current (not necessarily maximum) left plateau is the one we have already found (e.g., $[t_m, t_m]$ where t_m is the top-1 point in the whole series), and we would like to see if we can extend the current left plateau towards the left in order to find the maximum left plateau. For this purpose, we find the top-1 time position

(called "left top-1" in the digram) on the left of the current left plateau, and then we find the bottom-1 time position (called "local bottom-1" in the diagram) between the left top-1 and the current left plateau.

Three cases arise based on the τ value as depicted in Fig. 3. (Recall that τ gives the restriction that all the values in the plateau must be no less than τ). Consider Case 2 first as this is the case for the above example. In this case, the τ value is between the left top-1 value and the local bottom-1 value. The following are two useful observations for this case:

(1) Any value in the maximum plateau must be no less than the value of this left top-1. This gives us a new τ value for the left plateau.
(2) The left plateau cannot be extended to the time position of the local bottom-1. This gives us a new boundary when extending the left plateau.

By using these observations, we can extend the left plateau by using the new τ value and the boundary. This can be done with a recursive call to the extending procedure itself. One condition for the recursion to stop is if the new boundary is actually the current plateau. Going back to the above example, the procedure stops after we find the local bottom-1 is at position t_{n-1}, which is at the immediate left of the current left plateau (i.e., $[t_n, t_n]$).

Now consider Case 1. Since the left top-1 value is below τ, we know no time positions on the left of the current left plateau can be in the maximum left plateau. In this case, the current left plateau is the maximum left plateau.

Finally consider Case 3. In this case, we may be tempted to conclude that the left plateau can be extended to the time position of left top-1. However, this would be wrong if going to further left (left of the left top-1), we would meet a time position with a value lower than τ and then another time position with a value higher than the value of the local bottom-1. See Fig. 3 for this situation. What we need to do in this case is to find out if such a situation actually occurs. To do this, we recursively consider the time series on the left of (and including) the time position for the left top-1. Now local top-1 forms a left plateau by itself since it is a top-1 value in this subseries, and we try to extend the "local" plateau to the left. This (recursive) procedure will return a "local" left plateau starting from left top-1, and returns the actual τ value used by this "local" left plateau. If this returned τ value is still lower than the value of the local bottom-1, then we can conclude that all the positions on the right of the left top-1 are indeed in the left plateau (together with all the time positions in the "local" left plateau). Otherwise (i.e., the returned τ value is greater than the value of the local bottom-1), then we can conclude that the left plateau cannot be extended to the time position of left top-1, and the new τ value to use is the returned τ value from the "local" left plateau procedure.

We can now summarize our search algorithm in Fig. 4. In this algorithm, we refine the TA algorithm to look for top-1 and bottom-1 time positions (in terms of aggregated values) in an interval of $[left, right]$ of the time series. We assume TA will return the aggregated values associated with the top-1 and bottom-1 time positions. This extension can be obtained straightforwardly without requiring the data sources maintain separate sorted lists for each different time interval.

Procedure: *find_left_plateau*
Input: $[t_l, t_m]$ where t_l is the left boundary of the time interval to be considered and t_m is the top-1 position in $[t_l, t_m]$ and the right boundary of the time series to be considered, τ: the minimum value threshold for the left plateau
Output: $[t_{l'}, t_m]$: maximum left plateau $\tau' = max\{\tau, A(t_i) | i = l, \ldots, l' - 1\}$.

(0) If $l = m$, then return $[t_m, t_m]$ and τ.
(1) Let $t_t = top[t_l, t_{m-1}]$, and $t_b = bot[t_t, t_{m-1}]$.
(2) Three cases.
  (2.1) if $A(t_t) < \tau$, then return $[t_m, t_m]$ and τ.
  (2.2) if $A(t_t) \geq \tau$ and $A(t_b) < \tau$, then
    if $b = m - 1$, then return $[t_m, t_m]$ and $A(t_t)$;
    else recursively call *find_left_plateau* with $[t_{b+1}, t_m]$ and $\tau = A(t_t)$, and return what's returned from recursive call
  (2.3) if $A(t_b) \geq \tau$, then
    if $t_t = t_l$, then return $[t_l, t_m]$ and τ;
    else recursively call *find_left_plateau* with $[t_l, t_t]$ and τ
      assume the returned values are $[t_{l'}, t_t]$ and τ'
      (2.3.1) if $\tau' \leq A(t_b)$, then return $[t_{l'}, t_m]$ and τ
      (2.3.2) if $\tau' > A(t_b)$, then set $\tau = \tau'$ and goto Step (2).

Fig. 4. The *find_left_plateau* procedure for the distributed setting

We will use the notation top[*left*, *right*] and bot[*left*, *right*], where *left* and *right* are time positions, to denote the top-1 and bottom-1 time positions found by TA within the interval [*left*, *right*], respectively.

Theorem 3. *The algorithm in Fig. 4 correctly finds the maximum left plateau.*

The procedure to find the right plateau is similar. The complete algorithm that finds the plateau is the same as for the centralized algorithm, but will use TA to find the top-1 time position (Step 1, without computing the four arrays) and the search algorithms to find the left/right plateaux (Steps 2-6). It is easily seen that this complete procedure will find the correct plateau.

Example. Consider the time series in Fig. 5. We will only show how to find the left 8-plateau with $t_m = t_6$ and $\tau = 2$. During the initial call (*C-1*) with $[t_1, t_6]$, we find left top-1 is at $t_t = t_4$, and local bottom-1 is at $t_b = t_5$. Since $A(t_b) = A(t_5) = 3 > \tau = 2$, we are in Case 3 (Step 2.3), and we make a recursive call (*C-2*) with interval $[t_1, t_4]$ and $\tau = 2$. In *C-2*, we have $t_t = t_2$ and $t_b = t_3$, and we are looking at Case 2. Since $b = 3 = m - 1 = 4 - 1$ in this case, we return

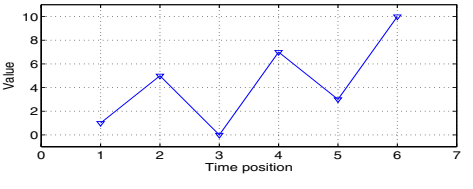

Fig. 5. Another example time series

to *C-1* with $[t_4, t_4]$ and a new $\tau = A(t_2) = 5$. In *C-1*, we were in Case 3 with returned $\tau' = 5$, and since $\tau' = 5 > A(t_b) = A(t_5) = 3$, we set $\tau = 5$ and go back to Step 2. Now we are looking at Case 2 since $A(t_t) = A(t_4) = 7 > \tau = 5 > A(t_b) = A(t_5) = 3$. Since $5 = b = m - 1 = 6 - 1$, we return $[t_6, t_6]$. Hence, we have found the maximum left 8-plateau to be $[t_6, t_6]$ and the return $\tau = A(t_t) = A(t_4) = 7$. □

4.3 Optimizing the Distributed Search Algorithm

There are many optimization techniques to add to the search algorithm. Here we only mention three of them that are used in our implementation. Other opportunities are abundant but are not pursued in this paper.

To start with, for Step 1, we may want to find the leftmost $top[t_l, t_{r-1}]$ and rightmost $bot[t_t, t_{r-1}]$ if there are multiple left top-1 and local bottom-1 time positions. While the algorithm is still correct if we use an arbitrary $top[t_l, t_{r-1}]$ and $bot[t_t, t_{r-1}]$ time positions among the multiple possibilities, the use of the leftmost and rightmost time positions, respectively, generally gives us the advantage in obtaining the plateau faster.

For Step 2.3, if $t_t = t_b$, then we know that all the time positions between $[t_b, t_m]$ have values no less than $A(t_t)$ (also no less than τ), then we may immediately extend the left plateau to $[t_b, t_m]$ without any recursion (although recursion will eventually find this extension as well).

Since we repeatedly use TA to find $top[t_l, t_r]$ and $bot[t_l, t_r]$, it is possible to reuse of the results across the different runs. For example, we may need to find $top[t_l, t_r]$ and later $top[t_l, t_{r-k}]$. During the search for $top[t_l, t_r]$, the final threshold value θ for TA used may be on a time position within $[t_l, t_{r-k}]$. In this case, we have already obtained the $top[t_l, t_{r-k}]$.

5 Experimental Results

In this section, we report the experimental evaluation of our distributed search algorithm. For the purpose of comparison, we also implemented the naive algorithm as mentioned in Section 4.1.

In order to control the experiments, we used synthetically generated data sets. We are interested in the situation that all the distributed data sources are monitoring the same phenomenon and hence the data should be somewhat correlated. In order to simulate this, to generate one data set, we first use a random walk to generate a *core time series* s_c, and then generate each input time series by (1) adding to the core with a fixed "shift" value, and then (2) randomly perturbing the value at each time position. That is, $s(t) = s_c(t) + shift + randpert$, where *shift* is a fixed (randomly picked) value for the entire time series s, and *randpert* is a random number at each time position. The parameters we used in our experiments are as follows: each step of the random walk takes a random value between $[-0.5, +0.5]$, i.e., $s_c(i) = s_c(i-1) + rand[-0.5, 0.5]$, and the shift is a random value between $[-5, 5]$ and the *randpert* is a random number between $[-2.5, 2.5]$. We used the sum as our aggregation function.

To give a "trend" to the random walk data, we modified the above generation of s_c with a slight bias. For the first half of the core time series, we add a small value (0.01 is used in the experiments) to each step, i.e., add 0.01 to $s_c(i)$, and in the second half of the core time series, we subtract the same small bias. This way, it's more likely that the time series will peak when reaching the middle of the time series. Since the bias is rather small, the trend is not prominent in our data sets.

Basically, three parameters affect the performance: the length of time series, the number of time series, and the ε value used for the plateau. Therefore, we tested our distributed search algorithm in three different ways, each varying one parameter while keeping the other two constant. The performance of our algorithm is measured on the number of accesses needed to the data sources (i.e., the number of sorted and random accesses required by the TA). For each fixed set of parameters, we generated 10 different data sets as described above and report the average number of accesses.

The result of the first experiment is reported in Fig. 6. In this experiment, we fixed number of series to 30, and ε to 90. As can be seen, the length of the series do not affect the performance too much on both algorithms, although our distributed algorithm performs better with one scale of magnitude. Intuitively, the naive algorithm would be affected by the series length because there may be more time positions with aggregated value above $A(t_m) - \varepsilon$. However, in our particular setting, due to the one "peak" nature of our time series, the performance of the naive algorithm does not degenerate as series length increases. As we observed (but not reported here), if we use a larger ε value, the performance of the naive algorithm generally goes poorer as the series length increases. In general, however, the performance of our distributed algorithm scales well with series length even in *multiple-peak* situations.

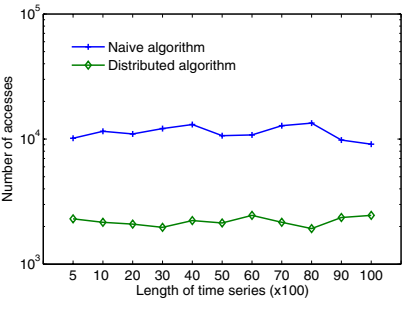

Fig. 6. Varying series length

The result of the second experiment is reported in Fig. 7. In this experiment, we fixed the time series length to 3,000, but varied the number of input time series from 1 to 100. Since we used sum as our aggregation, we varied the ε value in proportion to the number of time series. Specifically, ε is three times the number of time series (thus, if we have 30 time series, $\varepsilon = 90$). As can be seen that our distributed algorithm performs much better than the naive algorithm, with one scale of magnitude, consistently.

The result of the third experiment is reported in Fig. 8. In this experiment, we fixed the time series length to 3,000 and the number of time series to 30. Interestingly, when ε value is very small, the naive algorithm performs better than our distributed algorithm. In such cases, the naive algorithm retrieves almost

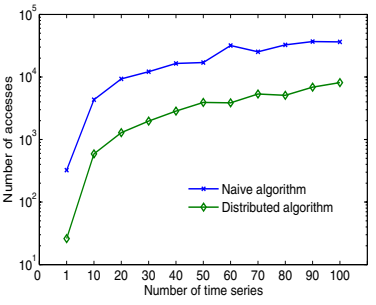

Fig. 7. Varying number of series

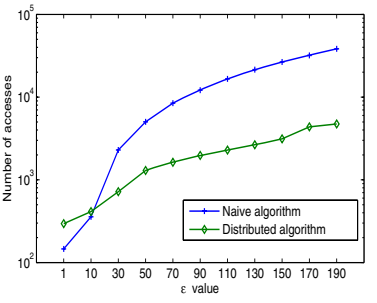

Fig. 8. Varying ε value

exactly all the time positions in the plateau. In general, if the plateau consists of all (or most of) the points that is above $A(t_m) - \varepsilon$, then the naive algorithm works very well. However, such cases should be rare in practice.

6 Conclusion

In this paper, we introduced the notion of the plateau in time series and presented two algorithms to find the plateau in aggregated time series. The first algorithm deals with the situation when all the data are available at a central location. In such a setting, we showed how the plateau can be found in linear time with respect to the length of the time series. The second algorithm is for distributed data sources in which we would like to reduce the communication cost. We presented a search algorithm that gives one scale of magnitude reduction in terms of communication cost over a straightforward use of the Threshold Algorithm [2].

As we observed, in some very special situations, the naive algorithm actually performs better than our more sophisticated search algorithm. It will be interesting to see how to merge the naive strategy into the search algorithm to take advantage of the special situations.

References

1. A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Hierarchical in-network data aggregation with quality guarantees. In *Proceedings of EDBT*, 2004.
2. R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66:614–656, 2003.
3. D. Gunopulos. Data storage and analysis in sensor networks with large memories, presentation at IBM Watson Research Center, 2005.
4. C. A. Lang, Y.-C. Chang, and J. R. Smith. Making the threshold algorithm access cost aware. *IEEE Trans. Knowl. Data Eng*, 16(10):1297–1301, 2004.
5. S. Madden, M. J. Franklin, and J. M. Hellerstein. TAG: A tiny aggregation service for ad-hoc sensor networks. In *Proceedings of OSDI*, 2002.
6. A. Manjhi, S. Nath, and P. B. Gibbons. Tributaries and deltas: Efficient and robust aggregation in sensor network streams. In *Proceedings of ACM SIGMOD*, 2005.
7. M. Sharifzadeh and C. Shahabi. Supporting spatial aggregation in sensor network databases. In *Proceedings of ACM-GIS*, 2004.