Tuning the top-k view update process

Eftychia Baikousi Panos Vassiliadis
Dept. of Computer Science, Univ. of Ioannina
Ioannina, 45110 Hellas
{ebaikou, pvassil}@cs.uoi.gr

ABSTRACT
In this paper we handle the problem of maintaining materialized top-*k* views in the presence of high deletion rates. We provide a principled method that complements the inefficiency of the state of the art independently of the statistical properties of the data and the characteristics of the update streams.

1. INTRODUCTION
The *top-k querying problem* concerns the retrieval of the top-*k* results of a ranked query over a database. Specifically, given a relation R *(tid,* A_1, A_2,...A_m*)* and a query Q over R retrieve the top-k tuples from R having the k highest values according to a scoring function f that accompanies Q. Typically, f is a monotone ranking function of the form: $f : dom(A_1) \times ... \times dom(A_m) \to \Re$.

Related work has extensively dealt with the problem of efficiently computing the top-*k* results of a query. The first algorithms that occurred in this context are FA [3, 4] and TA [5], with various extensions that followed them for specific contexts (e.g., parallel or distributed computation, etc). In recent years, in an attempt to achieve improved performance, researchers solve the problem of answering top-*k* queries via materialized views [2, 7, 8]. In this setting, results of previous top-*k* queries are stored in the form of materialized views. Then, a new top-*k* query may be answered through materialized views resulting in better performance than making use only of the base relation from the database. As typically happens with materialized views, though, when the source relation is updated, we need to refresh the contents of all the materialized views in order to reflect the most recent data. To the best of our knowledge, only [10] has dealt with the problem so far. To sustain the update rate at the source relation without having to fully re-compute the materialized views, [10] maintain k_{max} tuples (instead of the necessary k) and perform *refill* queries whenever the contents of the materialized views fall below the threshold of k tuples. Yet, the approach of [10] suffers from the following problems: (a) the method is theoretically guaranteed to work well only when insertions and deletions are of the same probability (in fact, the authors deal with updates in their experiments), (b) there is no quality-of-service guarantee when deletions are more probable than insertions.

In this paper, we compensate for these shortcomings by providing a method that is able to provide quality guarantees when the deletion rate is higher than the insertion rate. The case is not so rare if one considers that the number of persons logged in a web server or a portal presents anticipated high peaks and valleys at specific time points or dates. The main intuition, thus, behind our work, is to deal with these phenomena efficiently. Consider for example, a database containing data about stores, products and customers visiting a shopping center near the metro station. When a train arrives, several potential customers arrive with it, at the same time though, there is a massive departure of existing potential customers due to the train's departure. We assume a pervasive environment, where customers are equipped with wireless devices and connect to the shopping center's server as they enter the building. Assume a relation *Customer* (*c_id*, *c_name*, *c_age*, *c_income*) as well as accompanying relations with the customer's profile, sales history, etc. For a salesman that needs to send the appropriate advertisements, it is important to know which customers are the top-*k* ones according to their characteristics. To achieve this, salesmen use queries that have scoring functions over customer data. For example, assume a salesman wants to advertise a new gadget about mobile phones. The salesman needs to create a profile for the new product, or register the product in an existing profile. The profile includes a formula that assigns a score for a potential customer according to several distance functions and matching of the gadget's and the customer's characteristics. To speed up things, it is reasonable to search for the top-*k* customers in order to send them the advertisement. When a train departs, many customers leave the shopping center; still, the top-*k* list of candidates per product must be maintained so that the remaining possibly interested clients are notified. Consequently, the top-*k* customer lists should be maintained when updates occur in the relation of customers.

The solution to the problem is not obvious for the following reasons. First, even if the value distributions of the attributes that participate in the computation of the score are known individually, it is not possible to compute the distribution of their linear combination, i.e., the score (unless they are stable probabilities – e.g., Normal, Cauchy). Second, even if we extend k with extra tuples to sustain the incoming stream of updates that eventually affects the top-*k* materialized view, the extra tuples increase the possibility that an incoming source update might affect the view, thus resulting in the need to recursively compute this extension. Finally, we need to accommodate statistical fluctuations from the expected values. In this paper, we provide a principled method that handles all the aforementioned problems. Moreover, we validate our approach with extensive experiments.

Roadmap. The structure of this paper is as follows: in Section 2 we review related work. We present our method for the fine-tuning of the actual size of a top-*k* materialized view in Section 3

VLDB '07, September 23-28, 2007, Vienna, Austria.

and present an example in Section 4. In Section 5 we present experimental results and in Section 6 we conclude our results.

2. RELATED WORK

The first algorithms that dealt with the problem of computing the top-*k* results of a query that utilizes a monotone function over the combination of a relation's attributes are due to Fagin [3, 4]. This first algorithms (a.k.a FA algorithm) where later complemented by the highly cited TA algorithm [5, 6] that appears to provide better performance. The research community was quick to provide additional means for the computation of the top-*k* tuples of such a query via the exploitation of materialized views. First, the PREFER system was introduced in [7, 8], which uses a core algorithm that answers top-*k* queries using materialized views in a pipelined way. The results of the PREFER research were further expanded in [2], where a linear programming algorithm was introduced for the same purpose.

To the best of our knowledge, the only paper that has dealt with the refreshment of top-*k* materialized views is [10]. [10] deal with the following problem: Given a base table *R (id, val)* where *val* is the score of the tuple according to a scoring function and a materialized view *V (id, val)* containing the top-*k* tuples from *R* according to their values, compute a k_{max} that is adjusted at runtime such that a refill query, that re-computes the view *V* from scratch for the missing part, is rarely needed. [10] formulates the problem through a random walk model. Still, the statistical properties of the model are guaranteed only when the probabilities for insertions and deletions are equal, or insertions are more frequent than deletions.

3. FINE-TUNING OF VIEWS TO SUSTAIN HIGH UPDATE RATES

In this section we present our method for the fine tuning of materialized views defined over a relation that goes through updates in high rates. First, we formally define the problem. Second, we sketch our method and then, we move on to further detail the individual steps of the method.

3.1 Formal definition of the problem

Given *a base relation R (ID, X, Y) that originally contains N tuples, a materialized view V that contains top-k tuples of the form (id, val) where val is the score according to a function f(x,y)=ax + by and a, b are constant parameters, the update ratios* Λ_{ins}, Λ_{del} *and* Λ_{upd} *for insertions, deletions and updates respectively over the base relation R,*

Compute k_{comp} *that is of the form* $k_{comp} = k + \Delta k$.

Such that *the view will contain at least k tuples,* $k \leq k_{comp}$*, with probability p, after a period T.*

Assume a base relation *R (ID, X, Y)*, that contains *N* tuples a materialized view *V* that contains top-*k* tuples of the form *(id, val)* where *val* is the score according to a function *f(x,y)=ax + by* and *a, b* are constant parameters. Assume that the last tuple in the view has value val_k. Given the aforementioned update rates, insertions, deletions and updates occur in the base relation *R* with probabilities P_{INS}, P_{DEL} and P_{UPD} respectively. These probabilities are expressed as: $P_{INS} = \frac{\Lambda_{INS}}{\Lambda_{INS}+\Lambda_{DEL}+\Lambda_{UPD}}$.

$P_{DEL} = \frac{\Lambda_{DEL}}{\Lambda_{INS}+\Lambda_{DEL}+\Lambda_{UPD}}$ and $P_{UPD} = \frac{\Lambda_{UPD}}{\Lambda_{INS}+\Lambda_{DEL}+\Lambda_{UPD}}$

In the rest of our deliberations, updates are treated as combinations of deletions and insertions. This is a quite reasonable treatment, since we are mainly interested in the statistical properties of the rates of these actions and not in their hidden semantics. A simple method for the conversion of the involved rates is given in Section 3.3.

Our problem is to find a k_{comp} that will guarantee that the view will be maintained when insertions and deletions will occur in *R*. In order to do so, we must estimate the number of insertions and deletions that might affect the view. In other words, we need to compute the probability of the view being affected by a tuple inserted in *R* or deleted from *R*.

Assume that a new tuple *z (id, x, y)* is inserted in *R*. The probability of this tuple affecting the view is $p\ (z > val_k)$. Hence, the probability of a new tuple to be inserted in *R* and affect the view *V* is p_{ins}^{aff} which is expressed as: $p_{ins}^{aff} = p\ (z > val_k) * p_{ins}$. The probability of a tuple to be deleted from *R* and affect the view *V* is p_{del}^{aff} which occurs as $p_{del}^{aff} = p\ (z > val_k) * p_{del}$.

A problem that occurs with the maintenance of k_{comp} tuples at the view side is that k_{comp} incurs extra maintenance overheads, since the tuples of Δk can suffer updates too. Thus, we need to compute p_{ins}^{aff} and p_{del}^{aff} for the case where k_{comp} tuples are maintained. Therefore, the view *V* will contain k_{comp} tuples instead of *k*. Assume that the last tuple of the view containing k_{comp} tuples is val_{kcomp}. Consequently, the probability of a new tuple *z* to affect the view *V* is $p(z>val_{kcomp})$ whereas the probability of a new tuple to be inserted in *R* and affect the view occurs as: $p_{ins}^{aff} = p(z>val_{kcomp}) * p_{ins}$. Respectively the probability of a tuple *z* to be deleted form *R* and affect the view *V* can be expressed as: $p_{del}^{aff} = p(z>val_{kcomp}) * p_{del}$.

3.2 Sketch of the method

The proposed method is focused around three main steps: first, we compute the percentage of the incoming source updates that affect a top-*k* materialized view; second, we compute a safe value for the additional view tuples that we need in order to sustain high deletion rates; third, we fine tune this value with a safety range of values. Specifically, the three main steps are:

1. Given Λ_{INS}, Λ_{DEL} and Λ_{UPD}, we can compute λ_{ins} and λ_{del}, p_{ins} and p_{del}, and finally, p_{ins}^{aff} and p_{del}^{aff} as well as λ_{ins}^{aff} and λ_{del}^{aff} .

   Λ_{INS}, Λ_{DEL} and Λ_{UPD} denote the ratios of insertions deletions and updates that occur in the base table *R*. p_{ins} and p_{del} denote the probabilities of an insertion and deletion occurring on the base table *R* respectively. p_{ins}^{aff} and p_{del}^{aff} denote the probabilities of insertions and deletions that affect the view *V* respectively. These probabilities are expressed as a function of k_{comp}. λ_{ins}^{aff} and λ_{del}^{aff} denote the ratios of insertions and deletions occurring in the view *V* in the period of operations *T*. Updates are treated as a combination of deletions and insertions thus λ_{ins} and λ_{del} denote the ratios of insertions and deletions including those occurring from updates.

2. Compute k_{comp} as a function of λ_{ins}^{aff} , λ_{del}^{aff} .

   k_{comp} denotes the number of tuples that the view V should initially contain, such that after a period of operations T, V will contain at least k tuples.

3. Fine-tune k_{comp} by using the variance of the probability that a deletion and insertion action affects the materialized view.

3.3 Handling of updates

Given Λ_{INS}, Λ_{DEL} and Λ_{UPD} and treating updates as a combination of deletions and insertions, the ratios λ_{ins} and λ_{del} can be computed through the following equations:

λ_{ins} = number of insertions including those from updates / T

λ_{del} = number of deletions including those from updates / T

Λ_{INS} = number of insertions / T

Λ_{DEL} = number of deletions /T

Λ_{UPD} = number of updates /T

Therefore, $\lambda_{ins}=\Lambda_{INS}+\Lambda_{UPD}$ $\lambda_{del}=\Lambda_{DEL}+\Lambda_{UPD}$. In addition, p_{ins} and p_{del} can be expressed through the usage of ratios as $p_{ins}=\frac{\lambda_{ins}}{\lambda_{ins}+\lambda_{del}}$ and $p_{del}=\frac{\lambda_{del}}{\lambda_{ins}+\lambda_{del}}$ respectively.

3.4 Computation of the actual rates that affect *V*

The problem now is to compute the probabilities p_{ins}^{aff} and p_{del}^{aff} that affect the view V. These probabilities can be computed as $p_{ins}^{aff} = p_{ins} * p\ (z > val_{kcomp})$ and $p_{del}^{aff} = p_{del} * p\ (z > val_{kcomp})$ respectively. Actually, p_{ins}^{aff} is the number of insertions affecting the view V divided by the number of insertions and deletions occurring on the base table R and p_{del}^{aff} is the number of deletions affecting the view V divided by the number of insertions and deletions occurring on the base table R. Now the problem is focused upon finding the probability $p(z>val_k)$.

In order to compute the above probability we will use the Empirical Cumulative Distribution Function $F_n(x)$ (ECDF). Instead of using of a particular parametric cumulative distribution function, we will use ECDF which is a non parametric cumulative distribution function that adapts itself to the data. ECDF returns the values of a function $F(x)$ such that $F_n(x)$ represents the proportion of observations in a sample less than or equal to x. $F_n(x)$ assigns the probability $1/n$ to each of n observations in the sample. In other words $F_n(x)$ estimates the true population proportion $F(x)$. ECDF is formally defined as follows [9]:

Let $X_1, X_2, \ldots, X_n$ be independent, identically distributed random variables and let $x_1<x_2<\ldots<x_n$ denote the values of the order statistics of the sample. Then the empirical distribution function $F_n(x)$ is defined by the following formula:

$$F_n(x) = \begin{cases} 0, & x<x_1 \\ \frac{i}{n}, & x_i \le x<x_{i+1} \\ 1, & x_n \le x. \end{cases}$$

The alternative definition of $F_n(x)$ is:

$$F_n(x) = \frac{number\_of\_values\_in\_the\_sample\_that\_are\_\le x}{n}$$

Assume that the base relation R contains N tuples and the view V should contain k_{comp} tuples. If we order these tuples according to their values then there are N-k_{comp} tuples in R with value less than the value of k_{comp}. The following theorem implies that when the sample size n is large, $F_n(x)$ is quite likely to be close to $F(x)$ over the entire real line.

Glivenko-Cantelli Theorem [1]:

Let $F(x)$ denote the density function of the distribution from which the random sample $X_1, X_2, \ldots, X_n$ was drawn. For each given number x ($-\infty<x<\infty$) the probability that any particular observation X_i will be less than of equal to x is $F(x)$. Therefore, it follows from the law of large numbers that as $n \to \infty$, the proportion $F_n(x)$ of observations in the sample that are less than or equal to x will converge to $F(x)$ uniformly over all values of x. Let $D_n = \sup_{-\infty<x<\infty} |F_n(x)-F(x)|$, the Glivenko-Cantelli theorem states that $D_n \xrightarrow{p} 0$. □

Therefore, the probability of a tuple z affecting the view V can be expressed as:

$p(z>val_{kcomp}) = 1 - p(z \le val_{kcomp}) = 1 - F_N(k_{comp})$

$$p(z>val_{kcomp}) = 1 - \frac{N-k_{comp}}{N} = \frac{k_{comp}}{N} \quad (1)$$

As a more general example, consider a base relation R where the score of its tuples according to a function follow an exponential distribution in the interval [0, 2] and that a view V requires the top-k tuples of R according to their score value. In Figure 1 the probability distribution function of an exponential distribution is illustrated. In addition, assume that the top-k tuples are the 20% of the relation R and thus the vertical line top-k shown in Figure 1 denotes the values of the tuples that participate in the top-k view. Thus, the values in the view are greater or equal to 0.3. Assume a new tuple t following the same exponential distribution being inserted in R. For the new tuple t the probability of its value participating in the top-k ones is again 20%.

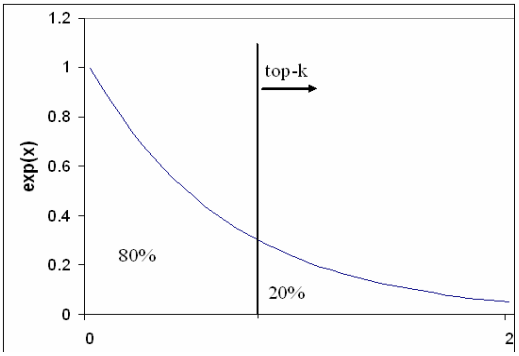


Figure 1. Exponential probability distribution.

Again, consider a similar situation where a view contains the top-k tuples from a base relation R according to a scoring function. Assume that the score values of R this time follow a beta

distribution in the interval [0, 1] with parameters given as 5 and 2. Figure 2 illustrates the probability distribution function of such a distribution. Similar to the previous example, the vertical line illustrated as top-k in Figure 2 denotes that the view contains 20% of *R*'s tuples where the values participating in the view are greater or equal to 1.7. Assume a new tuple denoted as *t* being inserted in *R*. The new tuple *t* will again follow the same beta distribution and the probability of *t* having a value greater than 0.8 is 20%.

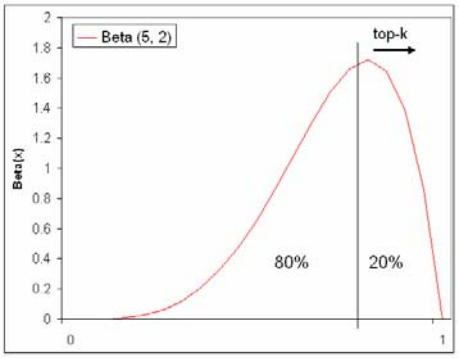


Figure 2. Beta probability distribution.

Therefore, λ_{ins}^{aff} and λ_{del}^{aff} are computed through the following equations:

$\lambda_{ins}^{aff} = p_{ins}^{aff} * (\lambda_{ins} + \lambda_{del})$ and $\lambda_{del}^{aff} = p_{del}^{aff} * (\lambda_{ins} + \lambda_{del})$.

According to equation (1), λ_{ins}^{aff} and λ_{del}^{aff} can be expressed as:

$$\lambda_{ins}^{aff} = p_{ins} * p(z > val_{kcomp}) * (\lambda_{ins} + \lambda_{del})$$

$$\lambda_{ins}^{aff} = p_{ins} * \frac{k_{comp}}{N} * (\lambda_{ins} + \lambda_{del}) \text{ and} \quad (2)$$

$$\lambda_{del}^{aff} = p_{del} * p(z > val_{kcomp}) * (\lambda_{ins} + \lambda_{del})$$

$$\lambda_{del}^{aff} = p_{del} * \frac{k_{comp}}{N} * (\lambda_{ins} + \lambda_{del}). \quad (3)$$

3.5 Computation of k_{comp}

The last step of the method is to compute k_{comp}, such that it will guarantee that the view will contain at least k tuples, $k \leq k_{comp}$, with probability p, after a period of operation T. In other words compute a k_{comp} that is of the form $k_{comp} = k + \Delta k$. In general, when the ratio of insertions λ_{ins} is greater than that of deletions λ_{del} it is clear that V will be maintained. The problem arises when the opposite occurs. That is when the ratio of deletions is greater than that of insertions. In such a case it is vital to compute a value for k_{comp} that can guarantee that V will contain at least k tuples after a period of operations.

1. Let us denote the frequency of deletions that affect the view V as λ_{del}^{aff}. In a period of time T, in order to keep the view maintained the following inequality should hold: $k_{comp}^{T} - \lambda_{del}^{aff} * \mathrm{T} \geq k$.

2. Thus, in case both insertions and deletions occur in a period of time T, in order to keep the view maintained for k_{comp} the following inequality should hold $k_{comp} \geq k + (\lambda_{del}^{aff} - \lambda_{ins}^{aff}) * T$. Clearly, to minimize memory consumption, we need to take the minimum possible k_{comp} and thus treat the above inequality as the equation $k_{comp} = k + (\lambda_{del}^{aff} - \lambda_{ins}^{aff}) * T$.

Therefore, by replacing λ_{ins}^{aff} and λ_{del}^{aff} from equations (2) and (3) the following equality occurs:

$$k_{comp} = k + (p_{del} - p_{ins}) * (\lambda_{ins} + \lambda_{del}) * \frac{k_{comp}}{N} * T \Rightarrow$$

$$k_{comp} = k + (\lambda_{del} - \lambda_{ins}) * \frac{k_{comp}}{N} * T \quad (4)$$

Thus, by solving the above equation according to k_{comp} we obtain:

$$k_{comp} = k * \frac{N}{N + (\lambda_{ins} - \lambda_{del}) * T} \quad (5)$$

Equation (5) has a meaning when $N + (\lambda_{ins} - \lambda_{del}) * T > 0$. This states that the size of the base relation R will not fall below 0, after updates occur in a period of operations T. At the same time, when $\lambda_{ins} - \lambda_{del} < 0$ (i.e., the case we are particularly interested in), then the fraction is greater than 1 and thus, $k_{comp} > k$.

3.6 Fine-tuning of k_{comp}

Although we now have a formula to compute the value of k_{comp}, we have expressed the probability of a new tuple *z(id, x, y)* affecting the top-k_{comp} tuples of the view as $p(z>val_{kcomp})$. Assume that a new tuple z is inserted in R. The probability of this tuple to affect the view is $p(z>val_{kcomp})$ whereas, the probability of this tuple not to affect the view is 1- $p(z>val_{kcomp})$. The above can be expressed as a Bernoulli experiment with two possible events. These are a) the new tuple being inserted in V with probability of success $p(z>val_{kcomp})$ and b) the exact opposite where the new tuple is not inserted in V with probability 1- $p(z>val_{kcomp})$. When the ratio of insertions occurring in the base relation R are λ_{ins}, a Bernoulli experiment is occurring λ_{ins} times where the probability of success is $p(z>val_{kcomp})$ and the number of successes follow a Binomial distribution. The probability of having Y_{ins} affected insertions in the view follow a Binomial distribution of the form $Binomial(\lambda_{ins}, p(z > val_{kcomp}))$. The variance of the above distribution can be expressed as:

$Var(Yins) = \lambda_{ins} * p(z>val_{kcomp}) * (1- p(z>val_{kcomp}))$.

The above formula indicates that insertions expected to affect the view may vary by $Var(Yins)$. Correspondingly, if there are λ_{del} deletions occurring in the base relation R, then the variance of these deletions expected to affect the view is

$Var(Y_{del}) = \lambda_{del} * p(z>val_{kcomp}) * (1- p(z>val_{kcomp}))$. This occurs as the variance of the Binomial distribution $B(\lambda_{del}, p(z>val_{kcomp}))$, which is similar to the one used for insertions.

Therefore in the worst case, in order to guarantee that the view will contain at least k tuples with confidence 95%, where $k \leq k_{comp}$, equation (4) becomes as stated below:

$$k_{comp} = k + (\lambda_{del} - \lambda_{ins}) * \frac{k_{comp}}{N} * T + 2 * Var(Ydel) + 2 * Var(Yins) \quad (6)$$

The confidence rate of 95% occurs from statistical properties concerning the variance factor appearing in formula (6). In case another confidence percentage is needed, formula (6) can be adjusted according to typical statistical methods [1].

3.7 Discussion

The problem of maintaining a view when updates occur in a base relation *R*, mainly lies in the problem of estimating the number of updates that will affect the view. Statisticians have contributed in this by providing formulas that compute the value of a probability of the form $p(z>val_{kcomp})$. However, the formula of such a probability depends on the distribution that the variable *z* follows. In our context, the variable *z* is a linear combination of the form *ax + by* where *x* and *y* are values from the attributes *X* and *Y* of the base relation. Even if the distributions that *X* and *Y* follow are known, the distribution of the score *Z* can not be computed unless *X* and *Y* follow a *stable* distribution. A stable distribution (e.g., Normal, Cauchy) has the property of stability. This property states that if a number of independent identically distributed (iid) random variables have a stable distribution, then a linear combination of these variables will have the same distribution. Therefore, the distribution of the variable *Z* can only be known in few cases. However, even if the distribution of the score was known, the probability $p(z>val_{kcomp})$ could be computed only with respect to the val_k instead of the value val_{kcomp}. This is because the val_{kcomp} could not been know in advance. Therefore, an iterative procedure would be needed. This occurs from the fact that we could compute the effect top-*k* tuples could have but not the effect the extra tuples would arise. Thus, a recursive procedure would be required.

4. EXAMPLE

As an example, consider the base relation *R (ID, X, Y)* initially containing *N* tuples with *N*=20 where attributes *X* and *Y* follow a uniform distribution over the interval [0, 100]. In addition, consider a materialized view *V* that contains the top-3 tuples (*k*=3) of the form *(id, val)* where *val*=3*x*+7*y* is the score according to a function *f(x,y)=ax + by* and *a*=3, *b*=7. The base relation *R* and the initial state of *V* are shown in Figure 3. Finally, the update ratios are Λ_{ins}=5, Λ_{del}=15 and Λ_{upd}=0. We will compute k_{comp} such that the view would contain k_{comp} tuples instead of *k* in order to be kept maintained when insertions, deletions and updates will occur in the base relation *R*. Moreover, let the period of operations occurring set as *T*=1.

According to the method of section 3.2, the ratios λ_{ins} and λ_{del} are 5 and 15 respectively. Therefore, p_{ins}=0.25 and p_{del}=0.75. The probability $p(z \geq val_{kcomp})$ can be calculated according to the following:

$p(z \leq val_{kcomp}) = F_N(val_{kcomp})$

$p(z \leq val_{kcomp})$ = (number of elements is score sample $\leq val_{kcomp}$) / N

$p(z > val_{kcomp}) = k_{comp} / 20$

In consequence, the probabilities p_{ins}^{aff} and p_{del}^{aff} can be calculated as: $p_{ins}^{aff} = p_{ins} * p(z \geq val_{kcomp}) = 0.25 * \frac{k_{comp}}{20}$ and

R		
id	**X**	**Y**
1	56	41
2	58	62
3	15	97
4	78	86
5	69	10
6	96	60
7	12	43
8	74	76
9	26	71
10	95	92
11	34	51
12	27	36
13	19	25
14	68	81
15	91	82
16	84	65
17	41	59
18	37	37
19	23	17
20	47	27

V	
id	**Z**
10	929
15	847
4	836

Figure 3. Base relation *R*

$$p_{del}^{aff} = p_{del} * p(z \geq val_{kcomp}) = 0.75 * \frac{k_{comp}}{20}.$$

Given the previous probabilities, the effective update ratios for the view *V* are then:

$$\lambda_{ins}^{aff} = p_{ins}^{aff} * (\lambda_{ins} + \lambda_{del}) = 0.25 * \frac{k_{comp}}{20} * (5+15)$$

$$\lambda_{del}^{aff} = p_{del}^{aff} * (\lambda_{ins} + \lambda_{del}) = 0.75 * \frac{k_{comp}}{20} * (5+15)$$

The above formulas state that if 5 insertions will occur in the base relation *R*, λ_{ins}^{aff} will affect the view and if 15 deletions occur then λ_{del}^{aff} will affect the view respectively. To be more specific the ceiling function is applied on λ_{ins}^{aff} and λ_{del}^{aff}. Therefore, for k_{comp} the following inequality holds:

$$k_{comp} \geq k + (\lambda_{del}^{aff} - \lambda_{ins}^{aff}) * T$$

$$k_{comp} \geq 6$$

where actually k_{comp} = 6. Thus, k_{comp} should be 6 in order to keep the view maintained after insertions, deletions and updates will occur in the base relation *R*. Suppose that insertions and deletions, shown in Figure 4, occur in the base relation *R*. The view *V* contains initially top-6 tuples and after updates the view will contain top-3 tuples. These are shown in Figure 5 where the dark shading denotes the initial top-3 tuples of *V* whereas the light shading denotes the extra top-3 tuples in order to have top-k_{comp} tuples.

insertions			deletions		
id	X	Y	id	X	Y
21	25	33	1	56	41
22	18	64	2	58	62
23	97	83	3	15	97
24	31	50	4	78	86
25	53	82	5	69	10
			7	12	43
			8	74	76
			10	95	92
			11	34	51
			12	27	36
			13	19	25
			15	91	82
			16	84	65
			17	41	59
			20	47	27

Figure 4. Insertions and deletions occurring in base relation *R*

V		
id	Z	
10	929	Deleted
23	872	Inserted
15	847	Deleted
4	836	Deleted
14	771	
8	754	Deleted
25	733	Inserted
3	724	Deleted

V	
id	Z
23	872
14	771
25	733

Figure 5. The view *V* prior and subsequent to updates

5. EXPERIMENTS

In this section we report on the experimental assessment of our method. We start with presenting the experimental methodology and then we discuss our findings.

5.1 Experimental Methodology

Our experimental study has been conducted towards assuring the following two goals:

1. *Effectiveness*. The first desideratum of the experimental study has been the verification of the fact that the proposed method can accurately sustain intervals with high deletion activity in the workload. In other words, the experimental goal was to verify that a top-*k* materialized view contains at least *k* items, in at least 95% of the cases.
2. *Efficiency*. The second desideratum of the experimental study has been the establishment of the fact that the computation of the exact number of auxiliary view tuples is faster than the computation of refill queries as proposed in the related literature. As well as the number of auxiliary view tuples is less than the number proposed in [10].

To achieve the first goal we have estimated k_{comp} via two methods: (a) without the fine tuning that uses the rates' variances (i.e., through formula 5) and (b) with this fine tuning (i.e., through formula 6). For both methods, we have computed the number of tuples that were deleted from the view, below the threshold of *k*.

In the context of the second goal, we have measured three metrics: (a) the memory overhead for k_{comp} and k_{comp} with tuning, measured as the number of extra tuples that we need to keep in the view, (b) the time overhead for computing k_{comp} and k_{comp} eith tuning as compared to the necessary time to compute the refill queries of [10] and (c)the time needed to compute the formula for k_{comp}. Again, we have evaluated these metrics using both the aforementioned methods.

In all our experiments we have used one relation $R(RID, X, Y)$ and one view $V(RID, score)$ with a formula $score = 3X+7Y$. The parameters that we have tested for their effect over the aforementioned measures are: (a) the number of relation tuples, (b) the number of materialized top-*k* results, (c) the fraction of the delete rate, over the insertion rate and (d) the percentage of the update stream over the relation size. We have not altered the time window *T* in our experiments; nevertheless, this is equivalent to varying the last parameter (denoted as λ), which measures the amount of modifications that take place as a percentage of the size of *R*. In other words, it is equivalent to increase the modifications number instead of reducing the window size.

We have tested the method over data whose attributes *X* and *Y* followed the Gaussian (with mean μ=50 and variance σ=10 for both *X*, *Y*), negative exponential (with a=1.5 for *X* and a=2.0 for *Y*) and Zipf distributions (with a=2.1 for both *X*, *Y*). The notation for the parameters and the specific values that we have used are listed in Table 1.

Table 1. Experimental parameters

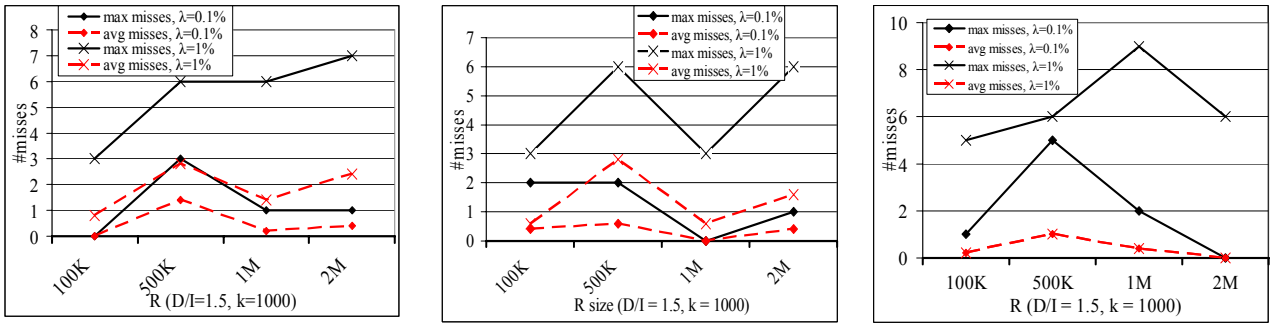
Size of source table *R* (tuples)	$\|R\|$	$1x10^5$, $5x10^5$, $1x10^6$, $2x10^6$
Size of mat. view (tuples)	*k*	5, 10, 100, 1000
Size of update stream (pct over $\|R\|$)	λ	1/1000, 1/100
Deletion rate over insertion rate (ratio)	*D/I*	1.0, 1.5, 2.0

5.2 Effectiveness of the method

The effectiveness of the method is demonstrated in Fig. 6 and Fig. 7. We present results organized by the data distribution of the attributes and compare two methods for computing k_{comp}, (a) the method including the fine-tuning part and (b) the method simply based on formula (5). We have conducted the full range of combinations of the values listed in Table 1, but we only present some indicative combinations for lack of space.

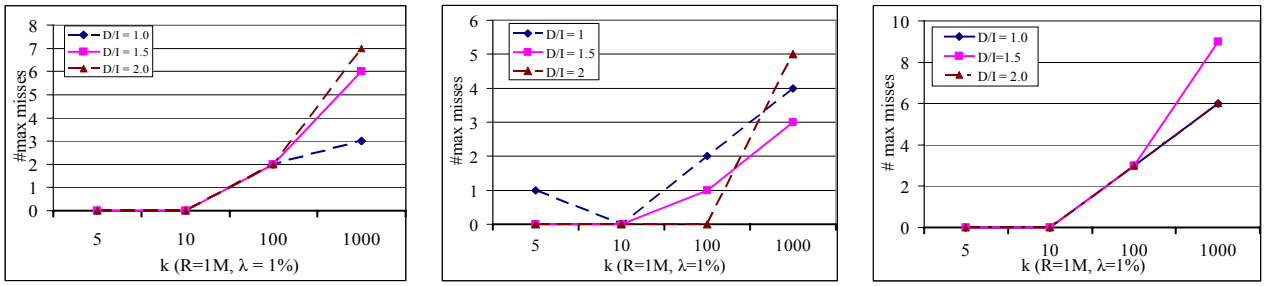
In Fig. 6, we fix *D/I* to 1.5 and *k* to 1000 (the largest possible value) and vary the size of *R* (in the X-axis) and the update stream size (in different lines in the Figure). Each experiment has been conducted 5 times. We measure both the average and the maximum number of misses. In Fig. 7 we report only the maximum number of misses, as it appears to be in analogy with the average in almost all the cases, and we vary *k* and *D/I*, while keeping *R* fixed to 1M rows and λ to 1%. The findings are as follows:

- The fine tuning method gives 0 losses, and thus describes the bold line lying on top of the X-axis in Fig. 6 and 7.
- If the fine tuning was not included, misses would have been encountered. In cases where insertions are close to deletions, the underestimation of the value of k_{comp} would lead to potentially important errors (in the Zipf case, errors have come up to 9 misses which is almost 1% of the top-*k* view size).
- It is also interesting how the distribution of data affects the stability of the error (Gaussian seems to converge, as expected, whereas the Zipf drops when the percentage of *k* is small over *R*, as the hot values are rather fixed)



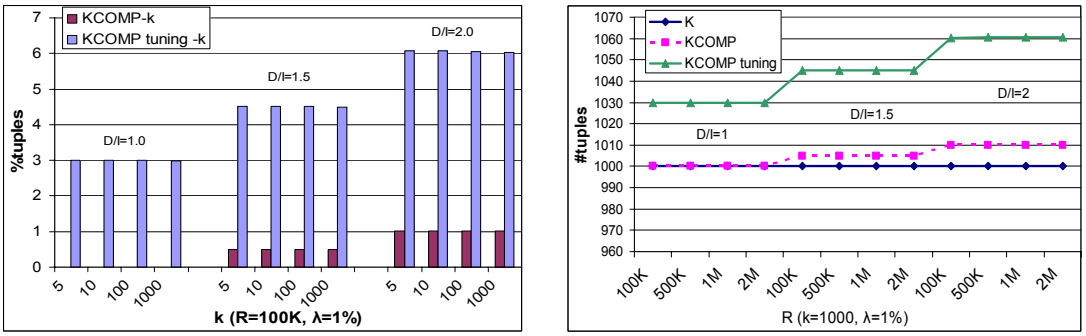
Gaussian Negative Exponential Zipf

Fig. 6. Maximum and Average misses as a function of |R| and λ



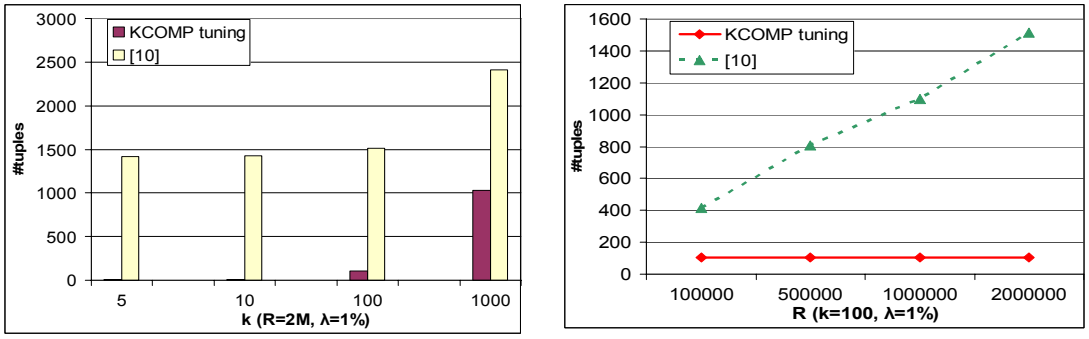
Gaussian Negative Exponential Zipf

Fig. 7. Maximum misses as a function of *k* and *D/I*



(a) Percentage of extra tuples as a function of *k* and *D/I* (b) Number of extra tuples as a function of *R* and *D/I*

Fig. 8 Comparison of *k*, k_{comp}, and k_{comp} with tuning



(b) Number of extra tuples as a function of *k* (a) Number of extra tuples as a function of *R*

Fig. 9 Comparison of k_{comp} with tuning and [10]

5.3 Efficiency of the method

We compared the values of k_{comp} without the fine tuning (i.e., through formula 5) and k_{comp} *tuning* with this fine tuning. The comparison of the above values was conducted for all three distributions as well as for all parameters listed in Table 1. Due to the fact that our formula is independent of the distribution the tuples follow and due to space limitations we only present some indicative results. In Fig. 8 we compare k_{comp} and k_{comp} *tuning* (a) as a function of k, where the size of R is 100000 tuples and (b) as a function of the size of R where we have fixed k=1000. For both of them and for all possible values of D/I the size of the update stream λ is 1% and the distribution is the Negative exponential. In Fig 8 (a) the Y-axis denotes the percentage of extra tuples. From both graphs in Fig. 8 we observe that k_{comp} is slightly greater than k and k_{comp} *tuning* is slightly greater than k_{comp} in all cases. The number of the auxiliary tuples in the view (i.e., k_{comp} and k_{comp} *tuning*) in the maximum case is approximately 1% and 6% respectively. Thus, the number of the auxiliary tuples does not cause a great extra memory cost.

In Fig. 9, we compare the value of k_{comp} *tuning* with the one proposed by [10]. Again, we compare the above (a) as a function of k where the size of R is set to 2M (the largest possible value) and (b) as a function of R where k is fixed to 100. In both graphs the distribution is the negative exponential. The parameter D/I=1, since it is the only value that can be compared with the proposed method in [10]. We notice that the number of tuples proposed by [10] is significantly larger than the one proposed in our method. Thus the memory cost in our method is considerably less.

N	K	Gauss	Negative exponential	Zipf
100K	5	328000	348500	242000
100K	10	333000	345667	239667
100K	100	335500	343000	239667
100K	1000	395333	406000	299500
500K	5	1650667	1715500	1216333
500K	10	1650667	1713000	1208333
500K	100	1653167	1710500	1205667
500K	1000	1736667	1796167	1291833
1M	5	3298667	3429000	2427167
1M	10	3301333	3426667	2429667
1M	100	3304000	3439500	2422167
1M	1000	3403167	3520500	2606667
2M	5	6650667	6900500	5406333
2M	10	6653167	6900833	4909000
2M	100	6747167	6906000	4906500
2M	1000	6895500	7082833	4992167

Fig. 10. Time to build the top-k view (microseconds)

The second part of our experimental results had to do with the comparison of the time needed to compute the value of k_{comp} as compared to the time needed to re-compute the view as part of a refill query. Fig. 10 measures the computation time needed for the view computation for a value of k in microseconds. On the contrary, the time necessary to perform the computation of k_{comp} has consistently been negligible (practically 0 in all occasions).

6. CONCLUSIONS

In this paper we handle the problem of maintaining materialized top-k views in the presence of high deletion rates. We provide a principled method that complements the inefficiency of the state of the art independently of the statistical properties of the data and the characteristics of the update streams. The method comprises the following steps: (a) a computation of the rate that actually affects the materialized view, (b) a computation of the necessary extension to k in order to handle the augmented number of deletions that occur and (c) a fine tuning part that adjusts this value to take the fluctuation of the statistical properties of this value into consideration. Our experiments have verified that the computation of this value is particularly fast and method results in almost zero losses of tuples due to the streaming deletions. Future work can be pursued towards an efficient refreshment method for large numbers of materialized views.

7. ACKNOWLEDGMENTS

This research co-funded by the European Union - European Social Fund (ESF) & National Sources, in the framework of the program "Pythagoras II" of the "Operational Program for Education and Initial Vocational Training" of the 3rd Community Support Framework of the Hellenic Ministry of Education.

8. REFERENCES

[1] Morris H. DeGroot, Mark J. Schervish, Probability and statistics, Addison Wesley, 2002.

[2] Gautam Das, Dimitrios Gunopulos, Nick Koudas, Dimitris Tsirogiannis. Answering Top-k Queries Using Views. In Proc. of the 32nd VLDB conference, pp. 451-462, Seoul Korea, 2006.

[3] Ronald Fagin. Combining fuzzy information from multiple systems. In Proc. of the 15th ACM Symposium on principles of database systems, pp. 216-226, Montreal Canada, 1996.

[4] Ronald Fagin. Fuzzy queries in multimedia database systems. In Proc. of the 17th ACM Symposium on principles of database systems, pp. 1-10, Seattle USA, 1998.

[5] Ronald Fagin, Amnon Lotem, Moni Naor. Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci. 66(4), pp. 614-656, 2003.

[6] Ulrich Güntzer, Wolf-Tilo Balke, Werner Kießling. Optimizing Multi-Feature Queries for Image Databases. In proc. of the 26th VLDB conference, pp. 419-428, Cairo Egypt, 2000.

[7] Vagelis Hristidis, Nick Koudas, Yannis Papakonstantinou. PREFER a system for the efficient execution of multi-parametric ranked queries. In Proc. of the ACM Special Interest Group on Management of Data Conference (SIGMOD), pp. 259-270, Santa Barbara USA, 2001

[8] Vagelis Hristidis, Yannis Papakonstantinou. Algorithms and applications for answering ranked queries using ranked views. VLDB journal, 13(1), pp. 49-70, 2004.

[9] Kishor S. Trivedi. Probability and statistics with reliability, queuing and computer science applications. John Wiley & Sons, Inc, 2002.

[10] Ke Yi, Hai Yu, Jun Yang, Gangqiang Xia, Yuguo Chen. Efficient Maintenance of Materialized Top-k Views. Proceedings of the 19th International Conference on Data Engineering (ICDE), pp.189-200, Bangalore, India, 2003.