Multi-user Searching of Top-k Objects with Data on Remote Servers[1]

Erik Horničák, Matúš Ondreička, Jaroslav Pokorný, and Peter Vojtáš

Department of Software Engineering, Faculty of Mathematics and Physics
Charles University, Prague, Czech Republic
{ondreicka,pokorny,vojtas}@ksi.mff.cuni.cz

Abstract. This paper focuses on searching the best *k* objects with more attributes according to user preferences in the Web environment. Attributes of an object type are distributed on servers in a disjunctive way, i.e. values of one attribute are stored in only one remote server on the Internet. In our work, every user can express his/her preferences for each attribute by a fuzzy function and mutual relations between the attributes by an aggregation function. We use client/server architecture and communication via Web Services. We deal with the usage of Fagin's NRA algorithm, which can find the best *k* objects without accessing all the objects. Because of support of sorting objects according to a fuzzy function, an indexing method based on B+-tree is used in each remote server. Moreover, each server is stateless, i.e. independent from any previous request. Our solution is based on cache memory, which loads objects from remote servers in batches and thus reduces the amount of network communication. In this paper we present a system TOPKNET, which can efficiently find the best *k* objects for various users with data on remote servers.

1 Introduction

In today's world, virtually any information is accessible trough the Internet and users of different systems are trying to find descriptions of objects, such as laptops, jobs, holidays, etc. The biggest disadvantage is that all the information is scattered trough out various seemingly unrelated Internet locations. For better understanding, we can use an example of choosing the best holiday destination. When choosing a holiday, one must consider many variables, which may change frequently and can be located on various remote servers, such as airline tickets price, weather forecast, accommodation prices or distance from points of interest. When a user wishes to do a more complex search, he/she is forced to access every location separately and put the obtained data together by himself/herself.

In most cases, these objects are of the same type and have several attributes. Each object has different attribute values, which differ from each other. According to the values of these attributes, users are searching for objects that best suit their

[1] This work was supported by the grant SVV-2011-263312, by GACR grant P202-10-0761 and by grant number 9209 of Grant Agency of Charles University (GAUK) in Prague.

preferences [8][7][5]. Each user prefers different attribute values. In general, the user is only looking for a few objects that are the best according to his/her preferences. Sometimes the user is looking for only one best object, for example, for a new job.

The problem of searching the best *k* objects according to values of different attributes in the same time is indicated as a *top-k problem* [10][1][4]. A trivial solution of the top-k problem requires to load all relevant objects together with the values of their attributes from remote servers, to evaluate every object's rating by a rating function, and finally to select *k* objects with the highest rating.

In last few years, research of top-k problem solving has been progressing in various domains such as relational databases [3], XML [6], multimedia search [10] or distributed systems [9].

In this paper, we assume that attributes of an object type are distributed on servers in a disjunctive way. In general, we suppose that all the values of single attribute are situated only on single remote server. We developed system TOPKNET, which is capable to access all of these servers from single location and to find efficiently the best *k* objects according to user preferences based on fuzzy functions.

To the best of our knowledge, there are not many applications of top-k problem in the Web environment. Authors of [9] describe KLEE framework, which solves the efficient processing of top-k queries in wide-area distributed data repositories. The framework tries to minimize a computational cost, which includes network latency and local peer work. The KLEE framework uses approximate algorithms and searches for the best *k* documents but only according to an aggregation function.

In a survey [3] Ilyas et al. describe top-k query processing techniques in relational database systems. In our system TOPKNET, we used a model of user preferences based on fuzzy functions with usage of B+-trees. Our solution is original and it has not been included in exhaustive Ilyas's survey.

We focus on usage of Fagin's NRA algorithm [1], which can find the best *k* objects without searching all the objects according to a monotone aggregation function.

We focus on multi-user solution, where data is common for all users and each user can express his/her preferences for each attribute by a fuzzy function and mutual relations between the attributes by an aggregation function [8][7]. We apply this model in combination with NRA algorithm. It is possible, when we use a method for sorting objects according to a fuzzy function with using a B+-tree. This method was mentioned for the first time in our work [8] and used in several related works, which solve top-k problem for more users with usage of multidimensional B+-tree [5] or in more complex tree-oriented data structures [2]. In this work, we adapt this method for needs of network communication [11].

Finally, we developed a TOPKNET system [11], which is capable to minimize the amount of network communication by using the cache memory, which loads the data from remote servers in batches, and solves the network latency problems.

The paper is organized as follows. Sections 2 and 3 describe a model of user preferences and Fagin's NRA algorithm. Section 4 explains principles of the method for sorting objects according to a fuzzy function. In Section 5, we describe client/server architecture of the TOPKNET system. Sections 6 and 7 describe details of the Server and Client parts. Section 8 presents our experiments. Finally, Section 9 includes conclusions and provides some suggestions for a future research.

2 User preferences

Nowadays, in many search engines a user can only restrict the values of some attributes. The result of searching of these search engines can be empty set or a too large set of objects. The motivation for searching according user preferences is to find a few best *k* objects for the user.

In this paper, we focus on searching best *k* objects with more attributes according to the user preferences. We suppose a set of objects *X* with *m* attributes $A_1, ..., A_m$. Every object $x \in X$ has *m* values $A_1(x), ..., A_m(x)$ of these attributes.

A user chooses his/her preferences, which determine suitability of an object $x \in X$ in dependence on its *m* values of attributes. In this work, we use a *rating function R* (ranking function), which assigns a rating $R(x)$ for each object $x \in X$.

More formally, we suppose a rating function *R* with *m* variables $a_1, ..., a_m$ specified by scheme $R(a_1, ..., a_m)$: $dom(A_1) \times ... \times dom(A_m) \to [0, 1]$, where $a_i \in dom(A_i)$ for each $i = 1, ..., m$. We denote a rating of object *x* as a function $R(x) = R(A_1(x), ..., A_m(x))$ with one variable, where $A_1(x), ..., A_m(x)$ are values of attributes $A_1, ..., A_m$ of the object *x*, respectively. Then $R(x) : X \to [0, 1]$, maps every object $x \in X$ into interval [0, 1], where 0 means no preference and 1 means the highest preference.

According to object ratings it is possible to sort objects from *X* in descending order and determine the best *k* objects. In this work, we suppose that if there are more objects with the same rating as the rating of the best *k*-th object, a random object is chosen. We differentiate between local preferences and global preferences.

2.1 Local preferences

Local preferences reflect how the object is preferred according to only one attribute. In this case, we express a local preference for *i*-th attribute A_i, as a fuzzy function f_i. Fuzzy function f_i is understood as a mapping $f_i : dom(A_i) \to [0, 1]$, which maps every value of actual attribute A_i domain into [0, 1] interval.

Local preferences of the user *U* for the attributes $A_1, ..., A_m$ are represented by user fuzzy functions denoted as $f_1^U(x), ..., f_m^U(x)$, respectively [8][2]. Then a user fuzzy function $f_i^U(x) : X \to [0, 1]$, where $i = 1, ..., m$, maps every object $x \in X$ according to the value of its *i*-th attribute $A_i(x)$ into interval [0, 1] and it is possible to sort objects from *X* in descending order according to $f_i^U(x)$.

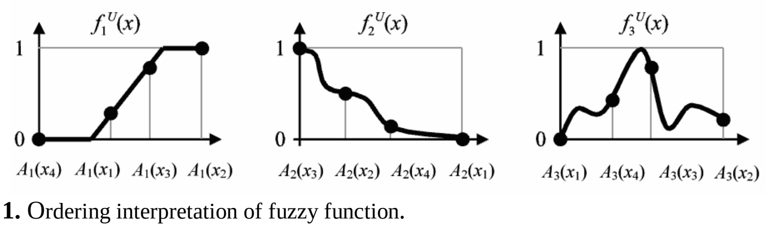

Fig. 1. Ordering interpretation of fuzzy function.

Figure 1 shows the ordering-based interpretation of user's fuzzy functions $f^U{}_1(x)$, $f^U{}_2(x)$, and $f^U{}_3(x)$. So $f^U{}_1(x)$ originates order of objects x_2, x_3, x_1, x_4, for $f^U{}_2(x)$ order of objects x_3, x_2, x_4, x_1, and for $f^U{}_3(x)$ order of objects x_3, x_4, x_2, x_1.

2.2 Global preferences

Global preferences express mutual relations between the attributes $A_1, ..., A_m$. For this purpose, we consider an *aggregation function*, which we denote as @, with m variables $p_1, ..., p_m$ specified as $@(p_1, ..., p_m)$: $[0, 1]^m \rightarrow [0, 1]$. The user U can express global preference by setting his/her user aggregation function $@^U$.

For the user U with his/her fuzzy functions $f_1^U, ..., f_m^U$ and aggregation function $@^U$, a *user rating function* R^U originates by means of substitution $p_i = f_i^U$ in user aggregation function $@^U$ [8][5]. Then $R^U(x) = @^U(f_1^U(x), ..., f_m^U(x))$, for every $x \in X$.

We say that an aggregation function @ is *monotone*, if $@(p_1, ..., p_m) \leq @(q_1, ..., q_m)$, whenever $p_m \leq q_m$, for every $i = 1, ..., m$. Because of NRA algorithm (see Section 3), in this paper we will be using monotone aggregation functions. We can use, e.g., the average, maximum, minimum, etc., as monotone aggregation function.

For example, for expressing of mutual relations between the attributes in the aggregation function, it is possible to use *weighted average*, where weights $w_1, ..., w_m$ of single attributes $A_1, ..., A_m$ determine how much user U prefers single attributes, i.e.

$$R^U(x) = (w_1 \cdot f_1^U(x) + ... + w_m \cdot f_m^U(x)) / (w_1 + ... + w_m)$$

When the user does not care about i-th attribute A_i, he/she can then set $w_i = 0$.

3 Top-k algorithm

In [1], Fagin et al. proposed top-k NRA (no random access) algorithm, which assumes that the objects from the set X are, together with values their m attributes $A_1, ..., A_m$, stored in m lists $L_1, ..., L_m$, where i-th list L_i contains pairs $(x, A_i(x))$, i.e. object (or object identifier) and its value of attribute A_i. When lists $L_1, ..., L_m$ are sorted in descending order according to the values of attributes $A_1(x), ..., A_m(x)$, respectively, then NRA algorithm is able to find the best k objects according to a monotone aggregation function @ [1].

The NRA algorithm uses only *sorted access* (sequential access) into the lists, which allows obtaining of the objects from each list L_i step by step from the top to the bottom by single pairs $(x, A_i(x))$. NRA algorithm obtains objects, in form of pairs $(x, A_i(x))$, by sorted access concurrently from sorted lists $L_1, ..., L_m$. For the obtained objects $x \in X$ the algorithm does not know all the values of its attributes. Therefore, it counts the worst possible rating and the best possible rating of the object x. In this way, it is possible to guess the value of not yet seen objects from the lists and algorithm NRA can stop earlier than it comes to the end of all the lists [1]. It means that not all the object $x \in X$ need be accessed.

4 A model of list based on B+-tree

For the multi-user solution with support local preferences based on fuzzy functions (see Section 2.1), we need to use a new model of lists, which allows obtaining objects in descending order according to user fuzzy functions $f_1^U, ..., f_m^U$.

NRA algorithm have to be obtaining objects, in form (x, $f^U(x)$), in descending order according to $f^U(x)$. Therefore, we apply a model of such a list based on well-known data structure B+-tree [8][5] and introduce an adaptation of a method for sorting objects according to a fuzzy function with using a B+-tree. Finally, we can use B+-trees B_1, ..., B_m instead of lists L_1, ..., L_m in NRA algorithm. In each B+-tree B_i all objects are indexed by values of i-th attribute A_i.

4.1 Usage of B+-tree

In B+-tree (see, e.g., [12]) objects from X are indexed according to the values of only one attribute in ascending order. We use a variant of the B+-tree, whose leaf nodes are linked in two directions. Every leaf node of B+-tree contains pointer to its left and right neighbor. Moreover, it is possible to cross the B+-tree through the leaf level and to get all the keys. In this way, it is possible to obtain objects from B+-tree in descending order according to course of user fuzzy function f^U [8][2][5].

When the function f^U is monotone on its domain then the following holds. If f^U is *nondecreasing*, we cross the leaf level of the B+-tree from the right to the left and get the pairs (x, $f^U(x)$) in the descending order according to f^U, because $A(x) \leq A(y) \rightarrow f^U(x) \leq f^U(y)$ holds. If f^U is *nonincreasing*, we cross the leaf level of the B+-tree contrariwise, i.e. from the left to the right.

4.2 A method for sorting objects according to a user fuzzy function

In general, the user fuzzy function f^U might not be monotone on attribute domain $dom(A_i)$. In this case, the domain can be divided into n continuous disjoint intervals $I = \{I_1, ..., I_n\}$, where f^U is monotone on each interval of I. This division into monotone intervals I is determined by the local maximums and local minimums of fuzzy function f^U, which identify bounds of these intervals.

Then the leaf level of B+-tree can be divided into *corresponding parts* according to these monotone intervals I. It is possible to obtain objects in the descending order according to f^U from each of these parts by crossing them in suitable direction. Moreover, objects can be obtained concurrently according to f^U from all these corresponding parts of the leaf level of B+-tree.

We introduce a *set of candidates* C, which contains some objects from the leaf level of B+-tree. The set of candidates C may contain only one object from each interval of I. For each object $x \in C$ we remember a pointer to the leaf level of B+-tree, its attribute value and the interval I_i to which it belongs.

A method for sorting objects according to a user fuzzy function with using a B+-tree is based on two steps:

1. a candidate set C is created that the object x with the highest $f^U(x)$ is chosen from each interval of I.
2. the pair (x, $f^U(x)$) with the actually highest $f^U(x)$ is obtained repeatedly.

The following pseudo-code illustrates an algorithm with two procedures, which describe the method for sorting objects according to a fuzzy function, i.e. getting pairs (x, $f^U(x)$) from the B+-tree in descending order according to user fuzzy function f^U.

```
Input: B+-tree, fuzzy function $f^U$;
begin
    candidate set $C$ := createCandidates(B+-tree, $f^U$);
    while($C \neq \emptyset$)do
        getNext(B+-tree, $C$, $f^U$);
    endwihile;
end.

procedure createCandidates(B+- tree, $C$, $f^U$)
begin
    Create monotone intervals $I = \{I_1, ..., I_n\}$ according to the course of $f^U$;
    $C$ := $\emptyset$;
    for each $I_i$ from $I$ do
        if($f^U$ is nondecreasing on the interval $I_i$)then
            If possible, find the first object $x \in I_i$ in leaf level of B+-tree,
            which has $A(x)$ less or equal to upper bound of interval $I_i$;
            $C$ := $C \cup \{x\}$;
        endif;
        if($f^U$ is nonincreasing on the interval $I_i$)then
            If possible, find the first object $x \in I_i$ in leaf level of B+-tree,
            which has $A(x)$ greater or equal to lower bound of interval $I_i$;
            $C$ := $C \cup \{x\}$;
        endif;
    endfor;
    For each $x \in C$ remember a pointer to the leaf level of B+-tree,
    $A(x)$ and the interval to which it belongs;
    return $C$;
end.

procedure getNext(B+-tree, $C$, $f^U$)
begin
    From candidate set $C$ choose object $x$ with the highest value $f^U(x)$;
    $I_i$ := an interval from $I$, for which $x \in I_i$;
    $C$ := $C - \{x\}$;
    if($f^U$ is nondecreasing on $I_i$)then
        If possible, move from object $x$ to the left neighboring object $y \in I_i$
        in leaf level of B+-tree;
        $C$ := $C \cup \{y\}$;
    endif;
    if($f^U$ is nonincreasing on $I_i$)then
        If possible, move from object $x$ to the right neighboring object $y \in I_i$
        in leaf level of B+-tree;
        $C$ := $C \cup \{y\}$;
    endif;
    return $(x, f^U(x))$;
end.
```

5 Architecture of TOPKNET system

Since this paper focuses on searching the best *k* objects in network environment, it was necessary to design a suitable architecture for needs of network communication.

For searching the best *k* object according to user preferences, we designed a client application, which is able to obtain objects with their attribute values from remote servers and efficiently find the best *k* objects for a user.

Since data is stored on remote servers, we have chosen *client-server* architecture and for network communication requirements the *Web Services* technology, specifically RPC type of SOAP protocol communicating via HTTP. Server operates as a service responding to the client application requests.

For large amount of objects stored on a server with values of attribute *A*, it is not efficient to load all the objects for searching the best *k* objects, since the NRA algorithm does not need to access all the objects (see Section 3). Therefore the client application loads objects from each server in batches. The client application sends requests repeatedly to individual servers, which respond to the requests by sending batches of objects. Figure 2 shows a simplified separation of the TOPKNET system into server and client component. The following two sections explain more detailed description of both components.

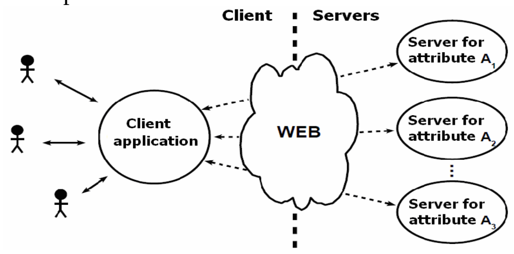
Client
Servers
Client application
WEB
Server for attribute A_1
Server for attribute A_2
Server for attribute A_3

Fig. 2. Architecture of TOPKNET system.

6 Server

The server part of TOPKNET system handles storage and fast access to data. All the values of a single attribute *A* of all the relevant objects are stored on server. The core of the server part is an algorithm, which manages data and communicates with client application via Web Service interface.

The algorithm is designed in such way, so as the server could operate statelessly, thus not to store any additional data for any connection. The server provides only response for immediate request and is independent on any previous request from client application. For minimizing the network communication amount, the server provides objects in batches.

To use local preferences in the NRA algorithm the server has to provide pairs, which are in format $(x, f^U(x))$, i.e., an object (or object identifier) and rating of its attribute value $A(x)$ according to user fuzzy function f^U. These pairs must be provided in batches in descending order according to f^U, which is the reason why objects are stored in B+-tree, where they are indexed according to values of the attribute *A*.

For the requirements of network communication we need to represent fuzzy functions in suitable manner. Handling the analytical representation of a function is unpractical, especially from the implementation point of view. Every fuzzy function can be approximated by a sequence of linear functions, which can be represented as sequence of points. In general, the course of user fuzzy functions is not too complicated; therefore such sequences of points contain only a few points. And that is why we use this representation of fuzzy functions in network communication.

6.1 Providing objects in batches

Since the server should be stateless and be able to provide objects in batches in descending order according to user fuzzy function f^U, each client application request has to contain information about this function. In TOPKNET we use for representation of a fuzzy function a sequence of points.

In case a client application requests the first batch of S objects, we proceed as shown in Section 4.2. We divide the definition domain of f^U according to its course into the intervals $I_1, ..., I_n$, where f^U is monotone.

The set of candidates C is created on the corresponding intervals in leaf level of B+-tree by the procedure **`createCandidates`** (see Section 4.2). Then the procedure **`getNext`** is executed S times, i.e., S pairs are obtained. In this way the server will provide the list of pairs *Batch* containing S objects, in form $(x, f^U(x))$, in descending order according to $f^U(x)$.

The following pseudo-code illustrates the server algorithm.

```
Input: B+-tree, fuzzy function $f^U$ , size $S$ of Batch, set of pairs $P$ ;
Output: list of pairs Batch;
begin
1.  if($P$ = Ø)then
2.     $C$ := createCandidates(B+-tree, $f^U$ );
3.  else
4.     $C$ := activateCandidates(B+-tree, $f^U$ , $P$);
5.  endif;
6.  Batch := create empty list of pairs $(x, f^U (x))$ ;
7.  for i:=1 to $S$ do
9.     $(x, f^U (x))$ := getNext(B+-tree, $C$, $f^U$ );
10.    Add pair $(x, f^U (x))$ into Batch ;
11. endfor;
12. $P$ := elements of $C$ in form $(x, A(x))$ ;
13. return Batch and $P$ ;
end.
```

When algorithm returns the list *Batch* of S pairs for the first time, it is sent as the first batch to the client application. In this moment the candidate set C contains a pointer to the leaf level of B+-tree. Further obtaining objects from B+-tree must continue from the identical state of C.

We require statelessness of the server and it is necessary to sent the batch of the next *S* objects according to f^U in the following request of client application. Therefore, the response for client application must contain not only the batch of pair, but also the set *C*, which determines where **`getNext`** procedure last stopped when obtaining objects from the leaf level of B+-tree.

As it is defined in Section 4.2, for object $x \in C$ we remember a pointer to the leaf level of B+-tree, $A(x)$, and the interval I_i to which the value belongs. Since transferring of pointers through network communication is overcomplicated, we represent elements of candidate set *C* as a *set of pairs P*, elements of which are pairs $(x, A(x))$ consisting of object *x* (or object identifier) and its attribute value $A(x)$ (see line 12 of the server algorithm).

Consequently, the server sends, as the response to the client application, the batch of objects with rating according to f^U and the set of pairs *P* (see line 13 of the server algorithm). If the client application sends request for the next batch of objects, it must contain the size of batch *S*, the sequence of points of f^U and the set of pairs *P*.

In this way we ensured statelessness of the server with minimal increasing the size of transferred data in both request and response.

The first request from the client application differs from the following ones, because its set of pairs *P* is empty. In this case, the set *C* must be created in leaf level of B+-tree via procedure **`createCandidates`**.

The following requests contain set of pairs *P*, from which the candidate set *C* is reconstructed (see line 4) in order to be able to continue obtaining objects in leaf level of B+-tree, where the algorithm stopped after obtaining objects in the previous batch.

The following pseudo-code illustrates the procedure **`activateCandidates`** for the reconstruction of candidate set *C* from the set of pairs *P*.

```
procedure activateCandidates(B+-tree, $f^U$, $P$)
begin
  Create monotone intervals $I = \{I_1, ..., I_n\}$ according to the course of $f^U$ ;
  $C$ := $\emptyset$;
  for each $x \in P$ do
    If possible, find the object $x$ according to its attribute value in B+-tree,
    which is traversed top to bottom only once (starting at the root) ;
    For object $x$ remember a pointer to the leaf level of B+-tree,
    $A(x)$ , and the interval $I_i$ to which it belongs ;
    $C$ := $C \cup \{x\}$;
  endfor;
  return $C$;
end.
```

7 Client application

The client application of TOPKNET system is able to obtain objects with their attribute values from remote servers and efficiently find the best *k* objects for a user. It uses the Web services interface for communication with remote servers.

In our solution, we try to minimize the amount of network communication by using the *cache memory*, which loads the data from remote servers in batches, and solves the network latency problems. The client application uses one cache memory for each single remote server.

7.1 Cache memory

TOPKNET system works on the client/server architecture principle. For the searching top-k objects it is necessary to be as fast as possible. Therefore, it is not acceptable for search by the NRA algorithm, which uses sequential access, to wait for every single object from every list to be loaded from the server. Thus the algorithm does not access the server directly, but through a cache memory. NRA algorithm then obtains objects from the cache sequentially in a descending order according to f^U. The objects are obtained as pairs $(x, f^U(x))$, i.e., object (or object identifier) and rating of object x by f^U.

The main part of the cache memory consists of two parallel threads using one *shared list*, which contains pairs $(x, f^U(x))$ loaded from the server. These pairs are sorted in a descending order according to f^U. We denote the size of this list as h.

The first thread, we call it *consumer*, provides pairs for top-k algorithm only. The second thread, we call it *producer*, ensures the loading of pairs in batches from remote servers. The consumer waits only for top-k algorithm to request another pair $(x, f^U(x))$, it loads the requested value from the shared list and returns it to the algorithm for further processing.

The main duty of the producer is to manage providing a sufficient amount of pairs in the shared list, in order to prevent top-k algorithm to wait for it. As soon as the consumer realizes that the amount of objects in shared list decreases under previously established level l (e.g. $l = h / 2$), it wakes up the producer, which starts obtaining further pairs from server in batches again and increases their amount in the shared list.

However, if the network communication with server is so slow that the producer is not able replenish the shared list, the consumer is put to sleep and waits for the producer to inform it that further pairs are now available. If this case occurs too frequently, it is suitable to increase the size of shared list h or to set higher value of level l.

7.2 Users of client application

Each one of the remote servers contains all the values of single attribute A_i and is able to provide the pairs $(x, f_i^U(x))$ in descending order according to user fuzzy function f_i^U. A particular user U of client application can specify his/her local preferences with fuzzy functions f_1^U, ..., f_m^U and a global preference with an aggregation function $@^U$. Then the client application gets pairs $(x, f_i^U(x))$ in batches from remote servers and can efficiently find the best k objects.

8 Implementation and Experiments

The implementation of TOPKNET system has been developed in Java. We used Web Service framework Apache Axis2 [13] for a communication between the client application and remote servers, where the network messages are sent in XML format. The remote servers of the TOPKNET system have been running on the Apache Tomcat Application Server [14]. These technologies are optimized to work together and ensure stability of the TOPKNET system.

In this paper, we focus on experiments in a network environment. There have been used three sets of 100 000 objects with 5 attributes values for testing. It means, we used five remote servers and a client application. We focused on the time, which is needed for finding the best *k* objects in the client application according to a user preference. We used user preferences, where some attribute values were the most preferred, and an arithmetic average which has been used as the aggregate function.

We find out that the computation time of NRA algorithm is most dependent on network communication between client application and remote servers. We used *k* = 20, i.e. we searched the best 20 objects, because the difference in the computation time in our particular experiment was relatively small for any *k* smaller than 250.

We tested the TOPKNET system with a usage of various values of network latency, which was simulated by a delay of server response. We focused on dependence of computation time of NRA algorithm according to size of cache memory. Naturally, the best of results have been achieved by zero network latency and the biggest size of cache memory. Figure 3 shows the results of this comparison.

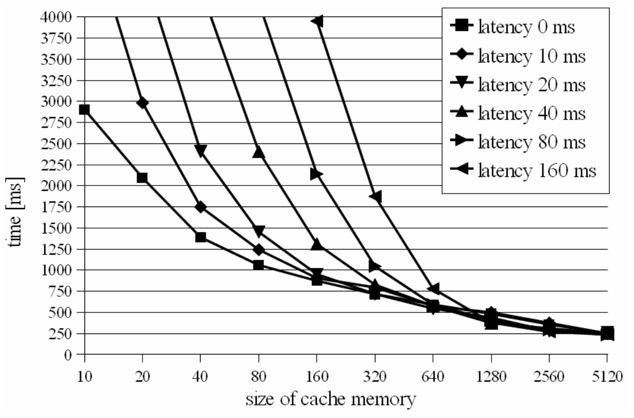

Fig. 3. Computation time of the TOPKNET system for different values of network latency and cache memory size.

9 Conclusion

We developed the TOPKNET system, which can efficiently find the best *k* objects for various users with data on remote servers.

Our solution for server part of TOPKNET system assumes that fuzzy function and data on the server are not changing during NRA algorithm computation. It will be interesting to solve problem in a more dynamic environment, e.g. data stored in B+-tee on the server is changing very fast.

Moreover, this solution permits that the size of batch can be different in each request of client application. This property can be utilized for further, more advanced optimization of the network communication.

In [4] the authors describe heuristics for NRA algorithm, which can automatically change a progress in sorted access to the lists. Similarly, a motivation for future research could be to develop some heuristics based on different network latency for each of the servers. For example, it is possible to monitor communication latency of servers during the run of NRA algorithm and it would be more suitable to adjust the properties of cache memory so that it could change its settings, i.e. *h* and *l* (see Section 7.1), automatically.

References

1. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. Journal of Computer and System Sciences 66, 2003, pp. 614-656.
2. Ondreička, M., Pokorný, J.: Efficient Top-K Problem Solvings for More Users in Tree-Oriented Data Structures. In: Proc. of IEEE Fifth International Conference on Signal Image Technologies and Internet-Based System, Marrakech, Morocco, 2009, pp. 345-354.
3. Ilyas, I. F., Beskales, G., and Soliman, M. A.: A survey of top-k query processing techniques in relational database systems. ACM Comput. Surv. 40, 4, Oct. 2008, pp. 1-58.
4. Gurský, P., Vojtáš, P.: Speeding Up the NRA Algorithm, in SUM 2008 - Scalable Uncertainty Management, Napoli , Italy, Springer, Sep. 2008, pp. 243-255.
5. Ondreička, M., Pokorný, J.: Extending Fagin's algorithm for more users based on multidimensional B-tree. In: Proc. of ADBIS 2008, P. Atzeni, A. Caplinskas, and H. Jaakkola (Eds.), LNCS 5207, Springer-Verlag Berlin Heidelberg, 2008, pp. 199-214.
6. Theobald, M., Bast, H., Majumdar, D., Schenkel, R., Weikum, G.: TopX: efficient and versatile top-k query processing for semistructured data. The VLDB Journal, Vol. 17, No. 1, January 2008, pp.81-115.
7. Gurský P., Vaneková V., Pribolová, J.: Fuzzy User Preference Model for Top-k Search. In: Proc. of IEEE World Congress on Computational Intelligence, Hong Kong, 2008.
8. Eckhardt, A., Pokorný, J., Vojtáš, P.: A system recommending top-k objects for multiple users preference. In: Proc. of 2007 IEEE International Conference on Fuzzy Systems, July 24-26, 2007, London, England, pp. 1101-1106.
9. Michel, S., Triantafillou, P., and Weikum, G.: KLEE: a framework for distributed top-k query algorithms. In: Proc. of the 31st international Conference on Very Large Data Bases, Trondheim, Norway, 2005, VLDB Endowment, pp. 637-648.
10. Chaudhuri, S., Gravano, L., Marian, M.: Optimizing Top-k Selection Queries over Multimedia Repositories. IEEE Trans. On Knowledge and Data Engineering, August 2004, Vol. 16, No. 8, pp. 992-1009.
11. Horničák, E.: Preference querying, indexing, optimization. Master’s thesis, Charles University, Faculty of Mathematics and Physics, Prague, 2011, (in Slovak).
12. Comer, D.: The Ubiquitous B-Tree. ACM Computing Surveys, Vol. 2, No. 11, June 1979, pp. 121-138.
13. Apache Axis2 - a Web Services / SOAP / WSDL / XML engine. http://axis.apache.org/
14. Apache Tomcat 6.0 Application Server. http://tomcat.apache.org