UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

TECHNICAL REPORT: TR-04-07

Best Bets: Thousands Queries in Search of a Client

Giuseppe Attardi Andrea Esuli Maria Simi

January 30, 2004

ADDRESS: via F. Buonarroti 2, 56127 Pisa, Italy. TEL: +39 050 2212700, FAX: +39 050 2212726

Best Bets: Thousands Queries in Search of a Client

Giuseppe Attardi
Dipartimento di Informatica
Università di Pisa
Via F. Buonarroti, 2, 56127 Pisa, Italy
+39 050 2212744
attardi@di.unipi.it

Andrea Esuli
Dipartimento di Informatica
Università di Pisa
Via F. Buonarroti, 2, 56127 Pisa, Italy
+39 050 2212775
esuli@di.unipi.it

Maria Simi
Dipartimento di Informatica
Università di Pisa
Via F. Buonarroti, 2, 56127 Pisa, Italy
+39 050 2212758
simi@di.unipi.it

ABSTRACT
A number of applications require selecting targets for specific contents on the basis of criteria defined by the contents providers themselves. Performing this task requires inverting the direction of search, retrieving queries from documents rather than vice versa, as in ordinary information retrieval. We present a retrieval system called *Best Bets* that handles this case. Best Bets generalize Information Filtering and encompass a variety of applications including editorial suggestions, promotional campaigns and targeted advertising. In particular we show how to model Google AdWords™ as a Best Bets system. We discuss how to address performance issues in the implementation of Best Bets, in particular efficient search, incremental updates and dynamic ranking. We report about experiments on an implementation of our techniques and a comparison with previous approaches.

Categories and Subject Descriptors
H.3.3 [**Information Systems**]: Information Search and Retrieval – *information filtering, retrieval models, search process.*

General Terms
Algorithms, Measurement, Performance, Experimentation, Theory.

Keywords
Information retrieval, information filtering, proactive contents delivery, search, query.

1. INTRODUCTION
A wealth of new applications is being spurred on the Web by the wish of contents providers to assume a more active role in contents delivery: query-free search [4][8][9], advertising services (e.g. Google's AdWords, AdSense and sponsored links), editorial promotions, shopping advice (e.g. Amazon), recommending systems [5], matchmaking applications in e-commerce [6].

In this paper we describe retrieval systems by means of an abstract retrieval model that also encompasses these new kinds of applications, where the interaction with the user is not necessarily explicit nor the sole source of criteria for the selection of results.

An unusual feature of the model is that it includes a mode of operation where the direction of search is reversed. In a traditional IR system, it is the user who indicates with his query which results he is looking for. The model instead considers also the case where the provider of information selects which users or users' needs he would like to satisfy. In other words the provider of information selects among the users rather than the other way round. In fact, we claim, this is the appropriate direction to use for a large class of applications, such as the ones mentioned earlier.

We call this class of applications *Best Bets* systems, since the contents that a provider selects for a specific user can be viewed as his *bet* on the fact that the user will find it interesting.

The traditional approach to search consists in finding all (or a subset) of documents in a collection that match a given query. In the Best Bets approach, the task it to find, given a set of queries, and a single document, all queries that match the document.

Moreover, since several providers can compete with each other for the user attention, a ranking score is typically essential to order those queries that satisfy the matching criteria.

Best Bets are a generalization of *Information Filtering* systems (also called *Selective Dissemination of Information* – SDI – in the context of digital libraries); these also involve matching objects against a query collection but ranking is not as important.

After introducing the Best Bets model and possible applications, we discuss how to address performance issues in their implementation, in particular efficient query search, incremental updates and dynamic ranking. We report about experiments on an implementation of our techniques and compare them with related results.

2. RETRIEVAL MODELS
Information Retrieval, Information Filtering and others tasks concerned with extracting information from documents share many features. Belkin and Croft [3] characterize the difference between a retrieval model and an information filtering model on the basis that "in filtering, an incoming stream of objects is compared to many profiles at the same time, rather than a single query being compared to a large, relatively static database". Figure 1 and Figure 2 illustrate graphically such difference.

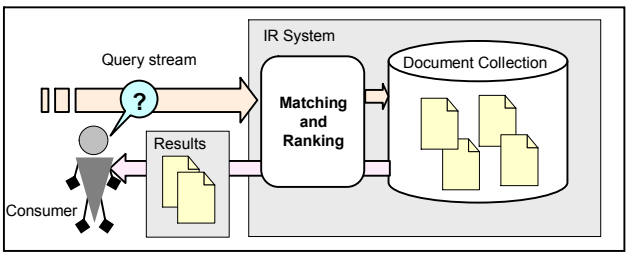

Figure 1. The IR model.

The main difference between the two cases is the information that is maintained in the collection and organized for fast retrieval: documents in IR, profiles or queries in IF.

It is helpful to refer to an abstract retrieval model encompassing most types of retrieval systems so that the essential differences between them can be highlighted.

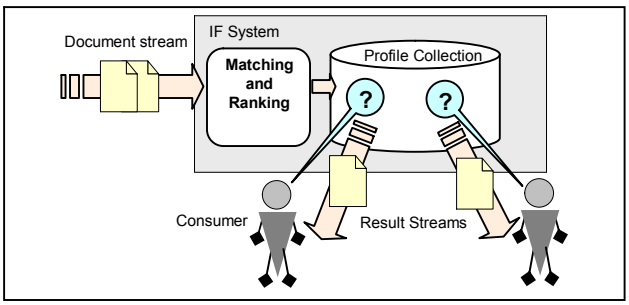

Figure 2. The IF model.

A retrieval model for an IR system provides an abstract description of the indexing process, the representation used for documents, the representation of queries, the matching process between a query and the documents and the ranking criteria.

Definition 1. A *retrieval model* consists of a tuple ‹*D*, *Q*, *match*, *rank*› where:

1. *D* is a collection of documents
2. *Q* is query language
3. *match*: $Q \times D \rightarrow \{0, 1\}$, a query matching Boolean function
4. *rank*: $Q \times D \rightarrow [0, 1]$ a ranking function

2.1 IR Models

The vector space model [12] represents documents and queries as weighted vectors of features. The features are usually terms in the documents, hence the dimensions of the 'terms space' is given by the vocabulary size. The weight of an index term reflects its significance in terms of representativeness and discrimination power, and influences the rank. Among several weighting schemes developed in the past, the most popular ones are based on a combination of term frequency and inverse document frequency (*tf-idf*) [11]. Retrieval does not use a match function, but only a rank function, comparing the query vector to each document vector. The computed rank value is used to sort query results in order of intended relevance. A typical ranking function *rank*(*q*, *d*) takes as argument a query *q*, turns it into a vector of features q_f, computes a similarity score *sim*(q_f, *d*) such as the *cosine*(q_f, *d*) and then combines it with a weight associated to the document itself like *PageRank*(*d*).

The Boolean model is a classical retrieval model based on set theory and Boolean algebra. Given a query term *t*, documents containing *t* are retrieved. In addition user queries may specify a set of Boolean conditions that retrieved documents must satisfy. In a pure Boolean model only a match function is used to compute a binary selection criteria and no ranking function.

Modern information retrieval systems and in particular Web search engines use a combination of the Boolean and vector space models: documents are selected according to Boolean conditions (*match*) and the results are ordered according to a similarity measure much like what is done in the vector space model (*rank*).

Additional query conditions may also be used: for instance *proximity* conditions require that two terms appear within a given distance in a document, *phrase search* conditions impose that words appear in the same sequence.

2.2 Retrieval Functions

Each retrieval system or application provides specific retrieval functions that exploit the features of the model. Retrieval systems specialize the abstract model in order to properly support such specific functions. For traditional IR, the main retrieval function is *document search*.

2.2.1 Document Search

Given a document collection *D*, the goal is to retrieve all or the top *k* best ranking documents satisfying a given query q, i.e.

$$search(q, D) = \{ d \in D \mid match(q, d) \}$$

$$searchTop(k, q, D) = \text{top } k \text{ elements in } sort(search(q, D), rank(q, .))$$

where *sort*(*S*, *rank*(*q*, .)) is the result of sorting the set of documents *S* according to *rank*(*q*, .), the ranking function partially applied to query *q*.

2.2.2 Query Search

Both Information Filtering and Best Bets systems involve the task of *query search*: given a collection of queries *Q*, the goal is to find all or the best *k* queries *q* that match a given document *d*, i.e.

$$QuerySearch(d, Q) = \{ q \in Q \mid match(q, d) \}$$

$$QuerySearchTop(k, d, Q) = \text{top } k \text{ elements in } sort(QuerySearch(d, Q), rank(., d))$$

where *rank*(., *d*) is the ranking function partially applied to *d*.

In *query search* the roles between documents and queries are swapped.

2.3 Search Asymmetry

In the abstract model, the direction of search is not accounted for, so document search and query search can both be expressed as easily. However, whenever a retrieval systems relies more heavily on retrieval functions operating in one direction, this provides an opportunity for specializing the model and the data structures and algorithms to support search in that direction.

For instance, several implementation techniques [14] [15] have been developed to optimize document search, like inverted lists, compressed posting lists, signature files and a number of query optimization techniques.

The same techniques are inappropriate for query search, even if in principle they could be used to compute the *match* and *rank* functions of the abstract model. In fact specific techniques for query search have been proposed for instance in [16] [17].

Besides Information Filtering, other retrieval systems that invert the direction of search include Information Routing and, in principle, all those applications in which the provider of information is willing to play a more active role in delivery process. Producers may want to indicate the set of consumers to whom they want to deliver their material. There may be several ways to do this, but in the context of IR it seems natural to use complex queries to express such selection criteria.

In a Best Bets application a user/consumer or his needs can be represented, often indirectly, as a short-lived document that collects some aspects about him or about his activity context: for instance a query to a search engine, his navigation history or the application context (e.g. the document being browsed).

Contents material is stored in a collection, where each item is composed of a content unit and selection criteria to identify its intended target. For each user request all queries that give a positive match are selected and the associated contents presented to the user ordered by a ranking measure.

While in IF who issues the query also receives the results, in BB producers build the query and consumers receive the results of the selection.

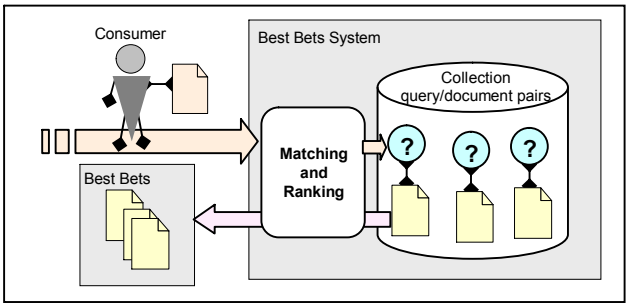

Figure 3 Best Bets model.

BB has similarities with both IF and IR and some distinguishing characteristics of its own. Both BB and IF systems deal with collection of queries: in principle the same algorithms and data structures may be used, with minor changes, for both tasks.

In BB ranking is essential to model the competition between documents, and hence between their producers, to reach a specific consumer target. Besides being effective, ranking must then be fair and transparent to producers.

Data in a BB collection may change quite frequently, as producers strive to compete for consumer attention. To take into account this aspect, a requirement in the design of a BB system is the ability to update the collection quickly without degrading performance. Performance is crucial in any interactive retrieval system (thus for IR and BB but not for IF) since users have been accustomed by Web search engines to expect almost immediate results.

In IF consumers may provide feedback by updating their profiles. In BB consumers provide feedback by showing interest in one of the bet results, for instance by clicking on it. Such feedback does not directly affects the profiles but it is still quite valuable, as it can be used to influence future ranking or even the pricing of the service [1].

2.4 Best Bets Search

In the Best Bets model the collection $D = Q \times C$ consists of pairs $\langle q, c \rangle$ where $q \in Q$ is a query and $c \in C$ is the associated document to be retrieved when the query matches a given input document d. Queries in Q represent target selection criteria for the items in C. For example, in an advertising application, d may be a user profile expressed as a list of keywords, Q contains queries expressing criteria for selecting potentially interested users for advertised products in the collection C.

The Best Bets task consists in retrieving all or the best k documents associated to queries matching a given document d, i.e.

$$BestBets(d, D) = \{c \in C \mid \langle q, c \rangle \in QuerySearch(d, D)\}$$

$$BestBetsTop(k, d, D) = \text{top } k \text{ elements in } sort(BestBets(d, D), rank(., d))$$

Devising a suitable query that will trigger the selection of the desired content is the subject of much empirical analysis. Google for instance provides tools that help estimating the match rate of a given query [1]. One can also envisage automated tools that generate queries extracting features from the contents and analyzing traces of previous submissions to the service.

2.5 Query Language

Since Best Bets match documents to queries, we need knowledge on the form of queries to devise suitable matching algorithms.

In the following, we will consider a simple language for expressing queries:

$$Q := term \mid \neg term \mid Q\ Q \mid Q \text{ or } Q$$

The matching function for this language must satisfy the following conditions:

$match(t, d)$	iff $t \in d$
$match(\neg t, d)$	iff $t \notin$ d
$match(q_1\ q_2, d)$	iff $match(q_1, d) \wedge match(q_2, d)$
$match(q_1 \text{ or } q_2, d)$	iff $match(q_1, d) \vee match(q_2, d)$

Definition 2. (*Disjunctive normal form*). A query is in disjunctive normal form if it has the form:

$$Q_1 \text{ or } \ldots \text{ or } Q_n$$

and none of the Q_i contains terms in this form.

Definition 3. Two queries q_1 and q_2 are semantically equivalent iff $\forall d \in D\ .\ match(q_1, d) \Leftrightarrow match(q_2, d)$

Lemma 1. Any query can be transformed in a semantically equivalent normal disjunctive form.

Without loss of generality, we will assume queries in *disjunctive normal form*. Queries of the form Q_1 or … or Q_n will be split in the n component queries, each of the form $w_1 \ldots w_k, \neg w_{k+1} \ldots \neg w_{k+j}$ with $k \geq 0$ positive terms and $j \geq 0$ negative terms.

3. BEST BETS APPLICATIONS

The Best Bets model encompasses a variety of applications ranging from almost traditional IR (i.e. query-free search) to more innovative contents delivery services such as promotional campaigns or advertising.

Editorial advice. Given a collection of news, books, music or videos, a Best Bets service may suggest relevant items to users according to the user context or the importance of the item itself. Each item in the collection can be annotated with features like freshness, authoritativeness of the source, reviews, editorial assessments on its quality and importance. Selection criteria may involve keywords in the user profile, the area of the site or news category being visited or other useful information to identify potential interests of the target user.

Amazon suggestions about related items chosen by other customers are an example of this kind of service.

ClickNews is a Best Bets system for news that we developed for a major news portal in Italy. It selects the three most relevant news of the day to be shown in the home page, sensitive to the user profile; it suggests news related to the search keywords used in the site search engine; it selects related news based on keywords drawn from the news article currently read.

Promotional campaigns. Promotional campaigns typically address specific target audiences. A common way to describe customers is by means of user profiles consisting in a list of keywords or phrases computed by means of some profiling technique, as it is typical in personalized services or recommendation systems. Using a Best Bets system, the target of a campaign can be expressed by selecting a set of user

profiles through a suitable query kept in the profile database. Different campaigns may compete with each other according to ranking policies decided by the provider of the promotion service, in order to choose the best suited promotions for each user.

Advertising. The selection of advertising banners to display in a Web page is a Best Bets application, whose selection criteria consist in a set of features associated to each advertisement: the ad to be displayed on a Web page is chosen by matching such features. More sophisticated variants have been introduced recently, e.g. Google's AdWords and AdSense [1], which take into account other factors, like the advertising budget or goals of competing advertisers. We will discuss in more detail the AdWords model in a later section.

Query-free search. Query-free search [4] is an information retrieval process that has the goal of finding documents that are relevant to a user's activity, without requiring an explicit query. It can be implemented as a Best Bets system that exploits information derived from context or user profile. An application to TV news broadcasts has been reported in [8]. Keywords extracted from the text of the running headlines make up the current context. Documents from the Web matching such context are selected and displayed. Lieberman and Liu [9] have developed a tool that suggests photos related to the text being edited, based on annotations or captions obtained from the user behaviour in previous sessions; these are learned by detecting correlations between a photo and features from the context of the document where the photo is inserted.

4. IMPLEMENTATION

We have explored several techniques for the efficient implementation of Best Bets applications. Our goal was to support search and updates to the query collection fulfilling the following requirements:

Efficient query search: search time on complex query collections as fast as search on document collections of comparable size.

Incremental queries updates: possibility of on-line modification of the query collection with immediate effect on the results, with good performances.

Dynamic ranking: support for a ranking model based on continuously updated ranking parameters.

We use implementation techniques similar to those described in [15] [17] for SDI systems, with important differences due to the different concerns imposed by Best Bets applications, i.e. dynamic ranking and query modifications. In particular the starting point is a technique described in [17] and called *ranked-key*.

4.1 Query Matching

In order to perform efficient query matching, the query collection is organized in a query index consisting of a dictionary of terms, alphabetically sorted and kept in main memory, and of associated inverted lists memory mapped from disc. As in [17], a query is indexed by choosing as key just the term with the lower frequency in documents appearing in a query; in fact since query terms are in *and*, all the terms will have to match against an incoming document including the one used as key. The desired effect is that the inverted lists associated to most frequent terms are kept small, since queries tend to be distributed over less frequent terms which are the majority. Figure 4 shows an example of this organization.

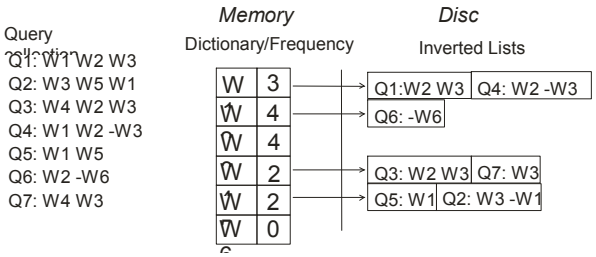

Figure 4. Ranked-list index

In [17] an estimate of terms frequency according to their use is assumed in building the ranked-key index, so that frequently used terms do not have lots of queries associated to them. In our implementation term frequency is computed from the query collection and is kept up to date when new queries are added or updated. Differently from [17] we deal with queries with negative terms. These however are not considered when selecting the index key (a query is never indexed under one of its negative terms) so they are not used in computing terms frequency.

Actual Web search engines have richer query languages, with other operators like phrase search (to search documents that contains a sequence of words) or proximity search (to search document that contains a set of words that occur within a specified distance of each other). In our implementation these operators can be implemented by adding terms' position constraints to the query representation in the inverted lists and checking for their satisfaction during search.

When a new document arrives, all the queries in the inverted list of each term in the document are matched against the document. Matching queries (where all positive terms are present in the document and none of the negative terms is present) are selected and passed on to the ranking phase described later.

4.2 Query Updates

For Best Bets it is important to allow frequent query updates without disruptions or performance degradation to the on-line search service. To achieve this goal, our implementation exploits a cache mechanism. Update requests do not affect immediately the query index but are temporarily registered in a main memory cache. The cache serves the dual purpose of aggregating and optimizing update requests as well as supporting searches on updated data, as if the updates had been performed.

The cache consists of three components, corresponding to three different kinds of operations: adding a new query, query update, query removal.

- *New queries* are added to an index with the same structure of the main ranked-key index, whose inverted lists are maintained on disc. Each new query receives a unique ID.
- *Query update* requests are stored in a hash table with key the query ID, after checking the presence of the ID in the query collection.
- *Query removal* requests are collected in a list where only the queries ID are stored.

In order to prevent losing data in case of a system crash, changes requests are sequentially recorded in a log file that enables a fast cache rebuilding after crashes.

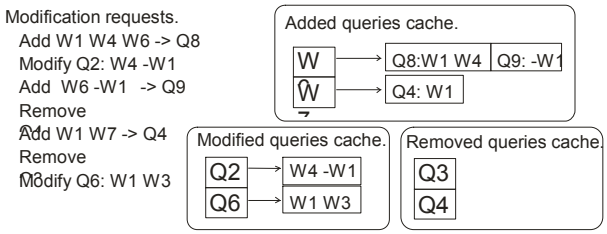

Figure 5. Cache components

Given a search request the following steps are performed: a search on disc is first performed and an array of results obtained; those results whose ID is in the query removal list in cache are discarded; a second search is performed in cache and results added to those already obtained; each result whose ID appears in the table of modified queries is discarded, the matching is computed again and the modified query added or discarded accordingly.

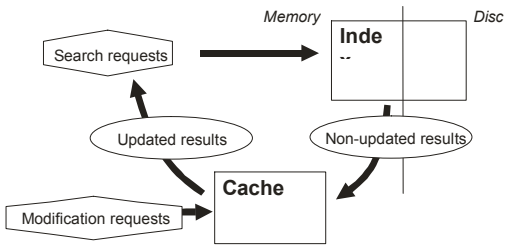

Figure 6. Index-cache search

When the cache reaches its maximum size its contents must be combined with the main index in order to obtain a new index. Building the new index involves: merging the dictionaries of the two indexes, constructing the new inverted lists by concatenating those lists associated to the same terms and appearing in both indexes. In doing this, information from the other cache components is taken into account in order to update queries or remove them.

During this operation it is still possible to search the collection since both the cache and the index on disk are used in reading mode. At the end of the update process the new index is swapped with the old one with an atomic operation during which search is suspended for a short time.

While the new index is being built, update requests cannot be accepted to avoid producing inconsistencies. To overcome this limitation we introduce a secondary cache, with the same structure of the first one, where the update requests are recorded.

The system alternates between two states:

- *Collection*: the update requests are collected until there is space in the primary cache; the search process uses the main index and the primary cache (see Figure 6).
- *Update*: the new index is constructed; the search process uses the main index, the primary cache and the secondary cache (see Figure 7).

When updates are complete the secondary cache becomes the primary one and starts working as an extension of the newly built index.

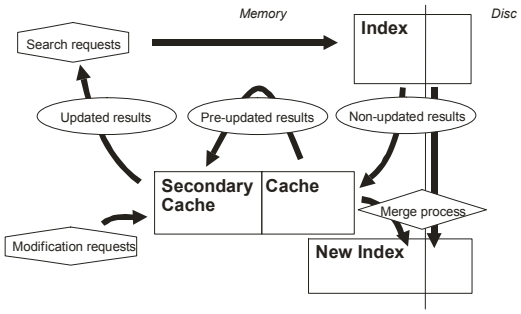

Figure 7. Update process

4.3 Query Ranking

The possibility to associate ranking policies to queries is fundamental in Best Bets applications. Some ranking criteria depend on the query content or that of the associated document (for example the rank may be based on the frequency of terms in the collection); other criteria may depend on external factors, e.g. the rank in Google's AdWords.

The information used for computing ranks should be accessible and updatable at any time. A solution where ranking information is maintained along with the inverted lists and updated with the above described schema is not acceptable. There are applications, e.g. AdWords, where ranking parameters need to be updated after each search; however in this solution these would not be directly accessible outside the search process.

As an alternative solution ranking parameters are maintained in a separate data structure, a sequential file with fixed length records. If dr is the dimension of the record, the ranking parameters for the i-th query are stored at the $dr \cdot i$ position in the file. The problem that some disc space may be wasted when queries are removed can be solved by reusing the IDs of removed query for new queries. In this solution the amount of data read from disc is kept to a minimum during search: in fact only those records for the matching queries need to be accessed.

4.3.1 Google AdWords

As an example of a complex ranking schema we will discuss how to implement the ranking for Google AdWords.

AdWords [1] is a service provided by Google to promote small text advertisements that redirect to vendor sites. The service is associated to the Google search engine and the Ads are selected according to the user's information needs as expressed by the keywords input to the search engine. Advertisers may specify queries to select a target audience for promoting their products and may influence the ranking criteria by specifying the maximum amount they are willing to pay for a click on their URL. Advertisers also specify a monthly budget to limit the maximum amount they want to spend in a month. The ranking system guarantees that the monthly budget will not be exceeded and that the Ad impressions (i.e. when an Ad is shown to a user) are uniformly distributed throughout the days of the month and also during each day.

Results are ranked by a function that takes into account four parameters:

- Cost per Click (*CPC*): the maximum cost the advertiser accepts to pay when an Ad link is followed.

- Click-Through Rate (*CTR*): the number of times an Ad has been clicked relative to the number of times the Ad has been shown.
- Daily Budget: the maximum amount of money the advertiser is willing to spend in a day. It is computed dividing the monthly budget by the number of days in the month.
- Daily Bill: the amount spent since the beginning of the day.

The ranking function is computed in three steps. First the *CTR* is checked to be greater than a global minimum *CTR* value (used to discard unsuccessful Ads).

Then a Spent Ratio value is computed as the ratio of the Daily Bill to the Daily Budget; if it exceeds the current elapsed portion of day time, the result is discarded. This allows for a more uniform distribution of Ad impressions throughout the day, avoiding that the whole daily budget is consumed too early.

The final value used for ranking is the product *CPC*·*CTR*.

The AdWords model involves also an intensive update process of ranking parameters. For each result shown to the user the corresponding counter is increased. After the presentation of results to the user, for each result that has been clicked the corresponding counter is increased and the amount of the Daily Bill is updated accordingly.

The cost of a click on an Ad is determined by a minimum cost policy that estimates the minimum cost that will not affect the rank ordering. For example if a clicked Ad has *CPC* = 0.60 $ and *CTR* = 0.5 and it is followed in rank by an Ad with *CPC*·*CTR* = 0.2, the clicked Ad will pay 0.41 $ (0.41·0.5 = 0.205 > 0.2).

At the beginning of each day the Daily Bill is set to a negative value equal to the Daily Budget not spent the day before. This allows spending the entire monthly budget without ever exceeding it. At the beginning of each month the Daily Bill is reset to zero.

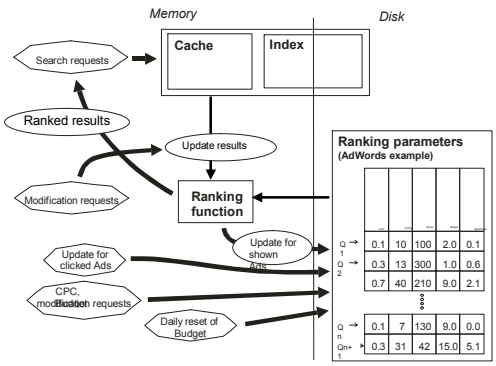

Figure 8. AdWords Ranking

5. Experimental Results

We report on some tests conducted on our BB prototype system.

The aim of the tests is to measure the impact on performance caused by the use of the cache and by the use of a complex dynamic ranking model, like AdWords.

5.1 Comparison test

The first tests are concerned with a comparison of our implementation of the BB framework with the techniques for IF proposed in [17].

As in [17], we built an artificial dictionary of 18,000 terms using a distribution function based on the Zipf law [14]. Input documents of 12,000 terms are built choosing random terms from the dictionary using its frequency distribution. Queries of the collection, composed of 5 terms, are built choosing, with a uniform probability distribution, terms from the 10% most frequent ones in the dictionary; these terms cover the 94% of occurrences of terms in documents [13]. The collection is composed of 300,000 queries.

We repeated the same tests made on the original ranked-key implementation (without the cache component) and we obtained comparable performance in terms of disk space usage, I/O and CPU load.

5.2 Cache test

We started from this first result to evaluate the performance of the complete system, including the use of the cache component. In particular we wanted to measure the overall degradation in performance introduced by the cache during search and the cost of the incremental updates.

The query collection and test documents are generated from a realistic dictionary extracted from the Aquaint text corpus [7], instead of the previously used theoretical distribution function. Input documents are built using the frequency distribution of the terms in a vocabulary.

The parameters used in the tests are:

- Query collection size: 2,000,000
- Query size, i.e. number of terms in a query: 5
- Document size: 200.

The query collection size is representative of a typical advertising Web service, according to recent figures reported about Google's AdWords. The size of the input document is estimated to cover the case of a profile obtained from the long-term observation of user behaviour.

Distinct measurements taken on I/O costs and CPU costs of search confirm that the impact of the cache on system performance only affects CPU time, and it is linearly proportional to the number of queries in cache (Figure 9).

In order to obtain an overall evaluation of performance degradation we have measured search time relative to the number of modifications pending in cache and to the index size. The system used for tests is a PC with an AMD Athlon 2.0 GHz CPU, 1 GB RAM, 80 GB disk, running the Windows 2003 operating system.

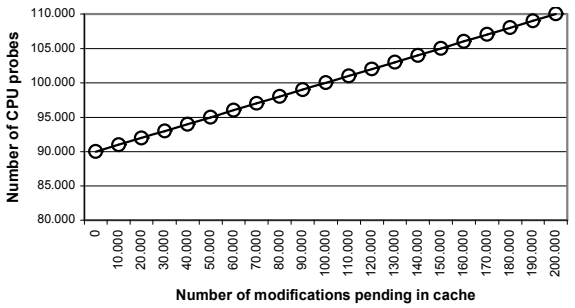

Figure 9. Number of CPU probes per search vs. number of pending changes

The test shows (Figure 10) that the number of pending cache updates does not impact search time significantly; the main cost is

determined by the I/O necessary to access the index on disk, and it is proportional to the index size. The portion of the search cost determined by the cache is linearly proportional to the number of pending cache updates. With an index size of two million queries the search time degradation caused by the presence of a 100,000 cache updates, with respect to the search on the same index without cache, is below 4%; it is around 2% for an index size of four millions queries. The time required for storing updates in cache is negligible.

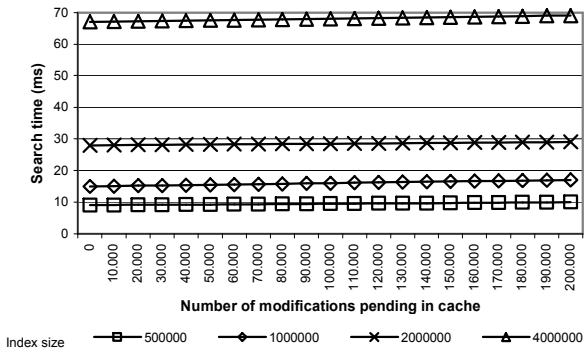

Figure 10. Search time vs. number of pending updates and index size.

In the tests related to the update process, i.e. the merging of index and cache to create a new index, time grows proportionally to the index size, with no relevant contribution from the cache.

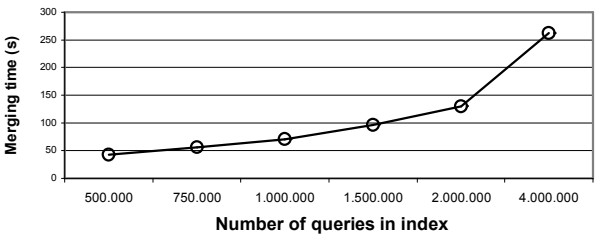

Figure 11. Update time vs. index size.

The results show that the cost of maintaining the pending changes in cache is low. This could be convenient when the index is growing, i.e. most of the pending changes are insertion of new queries, because the search cost on index and cache can be smaller than the one on the index produced by merging (here we consider a stationary state where insertions and deletions are balanced).

The upper limit to the cache size is equal to half of the available memory space, considering that during the update process the secondary cache is used. A lower bound is instead determined by the time required to complete the update process. In fact if the secondary cache gets full before the update terminates the system will not be able to accept more requests.

An estimation of a lower bound on cache size is determined by four factors:

- Index Size (*IS*): the size of the query index on disk
- Change requests frequency (*CF*)
- Search requests frequency (*SF*)
- Load factor (*LF*): the time spent by the system in the fusion process w.r.t. the time spent answering search requests.

These parameters are combined in a formula that provides an estimate on the total time for completing the update process:

$$UpdateTime = \left(1 + \frac{st(IS)}{SF}\right)\left(\frac{ft(IS)}{LF}\right)$$

where time values are expressed in seconds, *st* is a function that estimates an average search time given the index size and *ft* estimates the time required for fusion. These functions can be derived from the above graphics or by experimentation.

The cache size must be big enough to avoid being filled during the update time:

$$CacheSize \geq \frac{UpdateTime}{CF}$$

By instantiating this formula with typical values: IS = 1,000,000; CF = 0.5 *s*; SF = 0.005 *s*; LF = 0.01; and estimating: $ft(IS)$ = 60 s; $st(IS)$ = 0.017 *s*; we obtain:

$$CacheSize \geq 52800 = \frac{\left(1 + \frac{0.017}{0.005}\right)\left(\frac{60}{0.01}\right)}{0.5}$$

Using this lower bound on the cache size will cause the system to be most of the time in the update state. With a greater size, the system remains in a collection state that lasts for the time necessary to fill the cache again. At the end of the update process the cache has an average number of pending updates equal to the estimated minimum cache size and all the remaining space determines the duration of the collection state. Continuing the above example, with a cache size of 200,000 the collection state will have duration of:

$$200{,}000 - 52{,}800 = 147{,}200 \cdot CF = 73{,}600 \text{ s} \sim 20 \text{ h}$$

This means that the system can run a whole day in collection mode without interruptions for rebuilding the index or overhead for the use of the secondary cache.

5.3 Rank test

A complex dynamic ranking model can have a relevant impact on defining the search cost, for the additional computation required as well as for updating of ranking parameters in the index structures.

To have a measure of this impact we compared two AdWords-like services, one that uses a single fixed value ranking model (called Simp) and another that uses the ranking model described in 4.3.1 (called AdW).

Each service was running on the same query collection, based on a dictionary generated as in 5.2, and with the following parameters:

- Query collection size: 1,000,000
- Maximum query size: 5
- Minimum query size: 1
- Maximum negative terms in query: 2
- Minimum negative terms in query: 0

The single ranking parameter for queries in Simp was a constant float value, randomly generated. The ranking parameters for AdW are the monthly budget and the maximum cost per click, which determine the position and frequency of impressions of the Ad. The most critical aspect in this simulated AdW ranking model is

the correct determination of these parameters, since data from real world services such as AdWords were not available. Their values have been determined empirically with the goal of generating a heterogeneous and plausible set of Ads. The used values are:

- Monthly budget: between 10$ and 100$.
- Maximum CPC: between 0.01$ and 1.0$.

The documents used in the search process are queries generated in the same way of those in the collection. This can be considered as a worst case situation because the high correlation between queries and documents produces a high recall. It's expected that queries and documents produced in real applications have a lower correlation, thus producing a smaller number of results per search with an improvement of the overall system performance.

The test consisted in sessions where 20 concurrent threads were continuously querying one service. The services have been compared in average search time on various index sizes. Results show an average 20% performance degradation of AdW with respect to Simp (Figure 12).

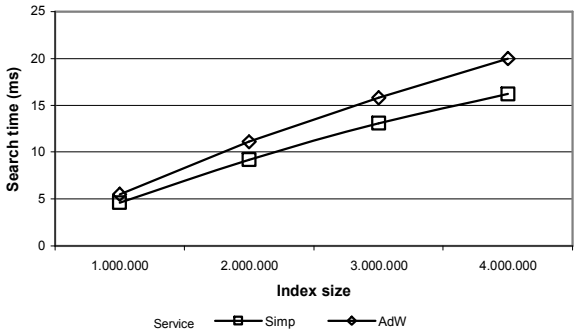

Figure 12. Comparison of search time for various index sizes.

To investigate the cause of the performance degradation we compared the search cost in terms of CPU probes and I/O operations for the two services.

The results (Figure 13) show that, while the CPU load is comparable, the major overhead observed in AdW is caused by I/O write operation, because each search does require reading five times more data to compute a rank (number of clicks, number of impressions, monthly budget, CPC, daily bill) and also to write back some of this fields, updated from the returned results (the number of impressions must be updated for all the shown results, and the daily bill and number of clicks for the ones clicked).

In the AdW model, threads are not only readers but also writers. This requires adopting locking and synchronization techniques to prevent erroneous rank computation and data corruption, and this in another cause of overhead. A performance improvement can be obtained, with a change in the ranking model, by not requiring an instantaneous update of ranking parameters after each search and packing write operations in blocks that will be reported on the index at regular time intervals.

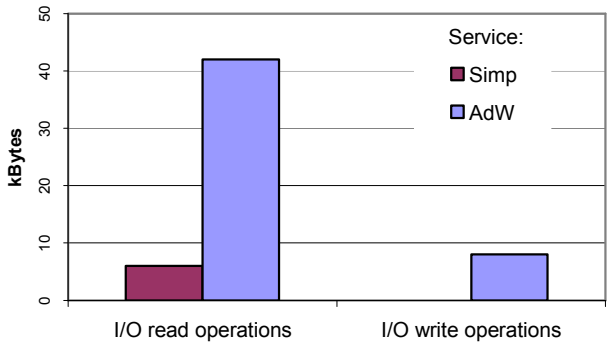

Figure 13. Comparison of average I/O generated by ranking operations for a search.

6. CONCLUSIONS AND FUTURE WORK

The Best Bets model describes a retrieval process that is suitable for an emerging class of applications where the contents providers define criteria for proactive content delivery. Relevant applications are related to e-commerce and on-line advertising but other fields of application are covered, like editorial advice and query-free search.

We discussed techniques for implementing BB applications, which allow concurrent search and updates, provide better performance with respect to an implementation relying on traditional search technology, and provide scalability with respect to index and cache sizes thus showing the feasibility of a high performance Best Bet service.

The combination of IR and BB models introduces a *bidirectional IR* model, where both users and providers may express their respective selection criteria. The model appears promising in supporting evolutions of BB and IR applications. In the future we plan to investigate the way to perform matching and ranking in the combined model.

7. ACKNOWLEDGMENTS

KSolutions supported this research with a grant within the ClickWorld Project. We thank Antonio Cisternino for useful brainstorming sessions in earlier stages of this work.

8. REFERENCES

[1] AdWords, https://adwords.google.com/, Google.

[2] Baeza-Yates R., Ribeiro-Neto B. *Modern Information Retrieval*. Addison-Wesley & ACM Press, 1999.

[3] Belkin N. J., Croft W. B. Information Filtering and Information Retrieval: two sides of the same coin? *Communications of the ACM*, 35(12), 29–38, 1992.

[4] Brin S., Motwani R., Page and Winograd T. What can you do with a Web in your Pocket? *Data Engineering Bulletin*, 21(2), 37–47, 1998.

[5] Burke R., "Hybrid Recommender Systems: survey and experiments", to appear in *User Modeling and User-adapted interacti*on.

[6] Di Noia T., Di Sciascio E., Donini F. M., Mongiello M., A System for Principled Matchmaking in an Electronic Marketplace, *Proceedings of The Twelfth International World Wide Web Confere*nce, Budapest, May 2003.

[7] Graff D. The AQUAINT Corpus of English News Text. http://www.ldc.upenn.edu/Catalog/

[8] Henzinger M., Chang B., Milch B., Brin S., "Query-Free News Search", *The Twelfth International World Wide Web Confere*nce, 707–717, 2003.

[9] Lieberman H., Liu H., Adaptive Linking between Text and Photos Using Common Sense Reasoning, *Conference on Adaptive Hypermedia and Adaptive Web Systems*, Malaga, Spain, May 2002.

[10] Loeb S., Terry D. Information Filtering. *CACM*, Vol. 35, No. 12, December 1992.

[11] Robertson S. E. and Sparck Jones K. Simple, proven approaches to text retrieval. Technical Report 356, Computer Laboratory, Cambridge University, May 1997.

[12] Salton G., Wong A. and Yang C. S. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975

[13] Tomasic A. and Garcia-Molina H., Performance of inverted indices in distributed text document retrieval systems. *Proc. of the International Conference on Parallel and Distributed Information Systems*. IEEE Computer society press, 8–17, 1993.

[14] Van Rijsbergen C. J. *Information Retrieval - 2nd edition*. Butterworths, London, 1979.

[15] Ian H. Witten, Alistair Moffat, and Timothy C. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishing, 1999.

[16] Yan T.W. and Garcia-Molina H., Index structure for Information Filtering under the Vector Space Model. In *Proc. of International Conference on Data Engineering,* 1994, pp 337-347.

[17] Yan T.W. and Garcia-Molina H., Index structure for Information Filtering under the Boolean Model. In *ACM Transaction on Database Systems* (TODS), Vol 19 Issue 2, 332–364, 1994.