# A Financial Cost Metric for Result Caching

Fethi Burak Sazoglu
Bilkent University
Ankara, Turkey
fethi.sazoglu@bilkent.edu.tr

B. Barla Cambazoglu
Yahoo! Labs
Barcelona, Spain
barla@yahoo-inc.com

Rifat Ozcan
Turgut Ozal University
Ankara, Turkey
rozcan@turgutozal.edu.tr

Ismail Sengor Altingovde
Middle East Technical University
Ankara, Turkey
altingovde@ceng.metu.edu.tr

Özgür Ulusoy
Bilkent University
Ankara, Turkey
oulusoy@cs.bilkent.edu.tr

## ABSTRACT

Web search engines cache results of frequent and/or recent queries. Result caching strategies can be evaluated using different metrics, hit rate being the most well-known. Recent works take the processing overhead of queries into account when evaluating the performance of result caching strategies and propose cost-aware caching strategies. In this paper, we propose a financial cost metric that goes one step beyond and takes also the hourly electricity prices into account when computing the cost. We evaluate the most well-known static, dynamic, and hybrid result caching strategies under this new metric. Moreover, we propose a financial-cost-aware version of the well-known LRU strategy and show that it outperforms the original LRU strategy in terms of the financial cost metric.

## Categories and Subject Descriptors

H.3.3 [**Information Storage Systems**]: Information Retrieval Systems

## General Terms

Design, Performance, Experimentation

## Keywords

Web search engines, result caching, metric, financial cost

## 1. INTRODUCTION

Web search engines cache query results for efficiency reasons. In the literature, the performance of different query result caching strategies is evaluated using different metrics. Hit rate (or miss rate), which is a widely used metric for result caches [3], measures the proportion of query requests that are answered (or missed) from the cache. This metric assumes that the processing cost of every cache miss is the same. Later, it was shown through cost-aware caching policies [7, 11] that queries have varying processing costs and the cache performance should be measured by taking these costs into account for cache misses. In [7], the cost of a query is simulated by considering the shortest posting list associated with the query terms. In a similar study [11], the cost is computed as the sum of the measured CPU time and simulated disk access time (under different posting list caching scenarios) and various static, dynamic, and hybrid cost-aware caching policies are proposed. In a more recent work [1], the performance of a hybrid dynamic result cache is also evaluated by the query cost metric.

Commercial web search engines rely on a large number of search clusters, each containing hundreds of nodes. Hence, they consume significant amounts of energy when processing user queries and the electricity bills for the large data centers form an important part of the operational costs of search engine companies[4]. In a recent work [8], energy-price-driven query forwarding techniques are proposed to reduce the electricity bills. The main idea in that work is to exploit the spatio-temporal variation in electricity prices and forward queries to search clusters that consume the cheapest electricity under certain performance constraints. Being inspired by that work, we propose a financial cost metric for evaluating the performance of query result caches. This new metric measures the total electricity cost incurred to the search engine company due to cache misses. Since the electricity prices and the query traffic of the search engine both show high volatility within a day, it is important to analyze the overall financial cost of query result caching techniques. To the best of our knowledge, the most similar metric to our financial cost metric is the power consumption metric used in [9]. In that work, a cache hierarchy consisting of result and list caches is evaluated in terms of the power consumption. Our financial cost metric considers the hourly electricity price rates and presents a more realistic financial cost evaluation.

The main contributions of this work are the following. First, we define a financial cost metric for query result caches. Second, the state-of-the-art static, dynamic, and hybrid caching techniques in the literature are evaluated using this new metric. Finally, a financial-cost-aware version of the well-known LRU strategy is proposed and shown to be superior to the original LRU strategy under this metric.


*SIGIR'13,* July 28–August 1, 2013, Dublin, Ireland.

ACM 978-1-4503-2034-4/13/07 ...$15.00.

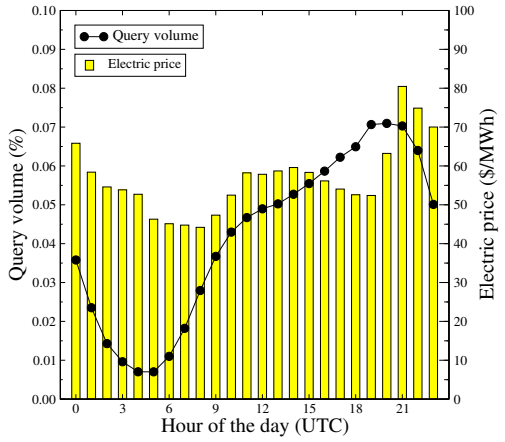


**Figure 1: Hourly query traffic volume distribution and hourly variation in electricity prices.**

The rest of the paper is organized as follows. Section 2 presents the proposed financial cost metric. In Section 3, we describe the well-known static, dynamic and hybrid result caching techniques that we evaluated under the hit rate and financial cost metrics. We also describe our financial-cost-aware LRU strategy. Section 4 presents the experimental results. We conclude the paper in Section 5.

## 2. FINANCIAL COST METRIC

Cost-aware caching strategies [7, 11] consider the time overhead of processing cache misses. The financial cost metric goes one step forward and computes the cost of electricity consumed when processing cache misses. The underlying motivation here is that the electricity prices show temporal variation and hence the financial cost of processing a query varies in time. In practice, the electricity price changes mainly based on supply-demand rates and certain seasonal effects [13]. The query traffic received by a search engine also fluctuates depending on time. As an example, Fig. 1 shows the hourly electricity prices taken from an electricity provider located in New York and the distribution of the query traffic received by Yahoo! web search. The electricity prices are normalized by the mean price value.

Our financial cost metric simply computes the processing time of the query weighted by the electricity price at the time of processing the query. The financial cost incurred by cache hits is ignored. Our financial cost metric is defined as

$$C_q = T_q \times P[t], \tag{1}$$

where $q$ is a query submitted at time $t$, $P[t]$ is the electricity price at time $t$, and $T_q$ is the time needed to process $q$.

## 3. RESULT CACHING TECHNIQUES

We evaluate the most well-known policies in terms of our financial cost metric. We also propose a new financial cost aware caching policy. In the rest of the paper, the frequency of a query $q$ is denoted as $F_q$. We now briefly describe each caching policy.

**Most Frequent (`MostFreq`)**: This policy basically fills the static cache with the results of the most frequent queries in the query log. Thus, the value of a cache item is simply determined as follows:

$$Value(q) = F_q. \tag{2}$$

**Frequency and Cost (`FC`)**: This policy [11] combines the frequency and cost of queries in a static caching setting. The value of a cached query is determined by the product of its (boosted) frequency and cost:

$$Value(q) = C_q \times (F_q)^K, \tag{3}$$

where $K > 1$.

**Least Recently Used (`LRU`)**: This well-known dynamic caching policy chooses the least recently requested query as the victim for eviction.

**Least Frequently Used (`LFU`)**: In this policy, each cache item has a frequency value that shows how many times the item is requested. The cache item with the lowest frequency value is replaced when the cache is full.

**Least Frequently and Costly Used (`LFCU`)**: This policy [11] is the dynamic version of the `FC` static caching policy. When the cache is full, the item with the lowest value (determined by its frequency and cost in Eq. (3)) is evicted.

**Greedy Dual Size (`GDS`)**: This policy [5] computes a so-called *H_value* metric for each cached query $q$ as

$$H\_value(q) = \frac{C_q}{S_q} + L. \tag{4}$$

This value combines the cost and size of the item with an aging factor $L$. This aging factor creates an effect similar to the recency component in the LRU policy. The cache item with the lowest *H_value* is chosen as the victim for eviction.

**Greedy Dual Size Frequency (`GDSF`)**: This policy [2] is a slightly modified version of the `GDS` policy. It further considers the frequency of the cache item when computing the *H_value*. We also boost the frequency component with a power coefficient $K$ as in [11].

$$H\_value(q) = (F_q)^K \times \frac{C_q}{S_q} + L. \tag{5}$$

**Static Dynamic Cache (`SDC`)**: `SDC` [6] is a hybrid caching policy that reserves a portion of the cache space for static caching and the remaining space for dynamic caching. Static cache handles long term popular queries while the dynamic cache handles short term (recent) popular queries. It is shown that this strategy outperforms both purely static and purely dynamic caches.

**Two-Part LRU Cache (`2P_LRU`)**: We propose a two-part LRU cache (similar to [1]) to optimize the result cache performance in terms of the financial cost metric. This strategy combines the segmented LRU idea (evaluated in [10]) with an admission control mechanism based on a financial cost threshold. Algorithm 1 presents the pseudocode for the `2P_LRU` policy, which exploits the financial cost variation in time and reserves a certain portion of the cache space for queries submitted in the high financial cost period (i.e., expensive cache ($E$)) and the rest for those submitted in the low cost period (i.e., cheap cache ($C$)). We decide on the expensive and cheap time periods based on a financial cost threshold ($T_P$). If the current financial cost ($T_q \times P[t]$) is less (greater) than this threshold, we say that the query is submitted in the cheap (expensive) time period, respectively. The `2P_LRU` policy operates as follows: If the result of a requested query $q$ is not found in the expensive and cheap caches, it is evaluated at the backend and then inserted into the cheap cache (incurs a financial cost of $T_q \times P[t]$) regardless of the submission time of the query, i.e., cheap or expensive periods. The intuition behind this choice is that,

**ALGORITHM 1:** Two-part LRU caching algorithm.

**Input**: $q$: query, $E$: Expensive cache, $C$: Cheap cache, $T_P$: financial cost threshold, $t_q$: submission time of $q$, $T_q$: time cost of $q$, $P[t]$: price at time $t$, $C_q$: financial cost of $q$

$R_q \leftarrow \emptyset$ ▷ initialize the result set of $q$;

**if** $q \notin E$ *and* $q \notin C$ **then** /* $C_q = T_q \times P[t_q]$ */
  evaluate $q$ over the backend and obtain $R_q$;
  insert $R_q$ into $C$ (if full, evict the LRU item);
**else if** $q \in E$ **then** /* $C_q = 0$ */
  get $R_q$ from $E$;
  update statistics of $q$ in $E$;
**else if** $q \in C$ *and* $T_q \times P[t_q] < T_P$ **then** /* $C_q = 0$ */
  get $R_q$ from $C$;
  update statistics of $q$ in $C$;
**else if** $q \in C$ *and* $T_q \times P[t_q] \geq T_P$ **then** /* $C_q = 0$ */
  evict $R_q$ from $C$ and insert into $E$ (if full, evict LRU item);
**return** $R_q$;

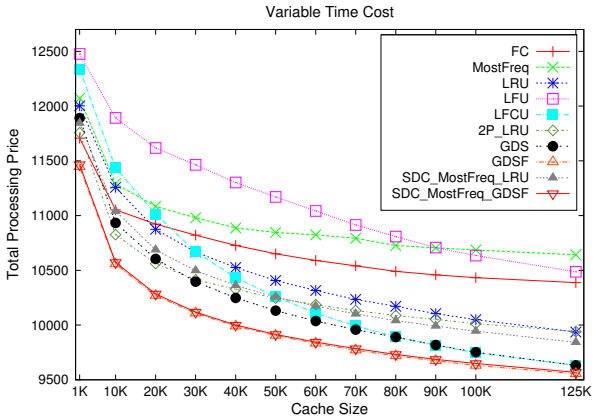


**Figure 2: Financial cost evaluation of caching policies assuming variable query processing time costs.**

as there is a high chance for a query to be singleton (i.e., submitted only once), we do not want to waste the expensive cache capacity without having enough evidence that the query will be re-submitted. If the query result is found in the cheap cache, then we check the current financial cost of the query and determine the time period. If we are in the cheap period, we simply serve from the cheap cache and update the query access statistics (i.e., housekeeping for LRU). Otherwise, we evict the query result from the cheap cache and insert it into the expensive cache (i.e., after seeing that the query is not a singleton). Finally, if the query result is found in the expensive cache, it is served from the cache and, again, the statistics are updated.

# 4. EXPERIMENTS

We evaluate the performance via a simulation using a subset of the AOL query log [12], which contains around 20 million queries. We use 2.2 million pages crawled from the Open Directory Project Web directory (`http://www.dmoz.org`) as our document collection. These pages are indexed using the Zettair search engine (`http://www.seg.rmit.edu.au/zettair/`) and a 1.1M query subset from the AOL query log is processed. We remove queries with no results. After this process, we end up with a stream of 809,795 queries over a period of six weeks. We reserve 446,952 queries (253,961

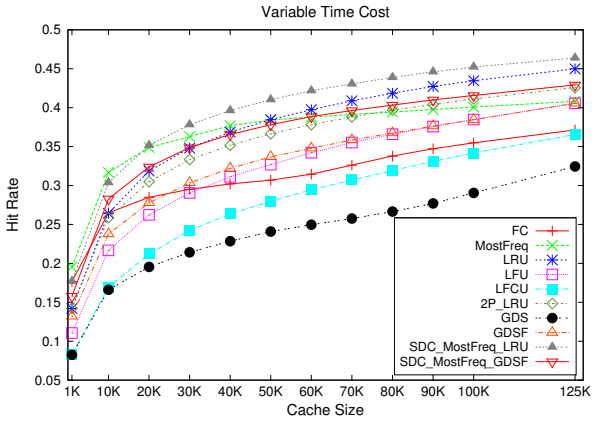


**Figure 3: Hit rates of caching policies assuming variable query processing time costs.**

unique) submitted in the first three weeks as the training set and use this set to fill the static caches and/or to warm up the dynamic caches. The remaining 362,843 (209,636 unique) queries form the test set, and the hit rate and financial cost metrics are computed over this set.

Our experiments consider two different cases. In the first case, query processing times of queries ($T_q$) are assumed to be variable. These times are measured as CPU times using the Zettair search engine. We refer to this case as "nonuniform (variable) time costs". In the second set of experiments, we consider only the price rate at the hour of the query submission ($P[t_q]$) and set the processing cost of queries to 1 (i.e., $T_q = 1$ for all queries). We refer to this case as "uniform (fixed) time costs". This latter scenario is motivated by the fact that search engines limit the time spent processing a query [8], i.e., we assume that the processing times of queries are nearly the same and close to this limit.

In our static cache simulation, when we compute the cost ($T_q \times P[t]$) for a query, we use the average electricity price observed in the training set when the query is processed at different times. For the dynamic caching setup, we consider the last time the query is issued and use the electricity price at that time point. We set various parameters as follows: For the `FC`, `LFCU`, and `GDSF` policies, $K$ is set to 2.5 [11]. When dividing the cache space in `SDC`, %20 is reserved for the static portion and the rest is for the dynamic portion (tuned empirically). We also experimentally tune the `2P_LRU` policy and allocate %60 of the cache space for the cheap cache, and the remaining %40 for the expensive cache. We set the financial cost threshold ($T_P$) parameter to 0.02 and 0.9 in the variable and fixed time cost scenarios, respectively.

Figs. 2 and 3 present the performance of caching policies for the variable time cost scenario, in terms of the financial cost and hit rate metrics, respectively. For a typical large cache (90K or 100K), the policies can be ordered according to their performance as follows:

- Financial cost: `LFU` ≅ `MostFreq` > `FC` > `LRU` > `2P_LRU` > `SDC_MostFreq_LRU` > `GDS` ≅ `LFCU` > `GDSF` ≅ `SDC_MostFreq_GDSF`
- Hit rate: `GDS` < `LFCU` < `FC` < `GDSF` ≅ `LFU` < `MostFreq` < `2P_LRU` ≅ `SDC_MostFreq_GDSF` < `LRU` < `SDC_MostFreq_LRU`

It is interesting to note that even though `LRU` and `SDC_MostFreq_LRU` policies are the best-performing policies, according to the hit rate metric, they are outperformed

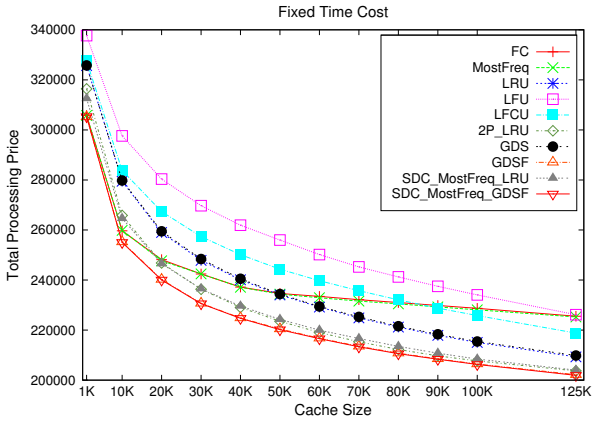


**Figure 4: Financial cost evaluation of caching policies assuming fixed query processing time costs.**

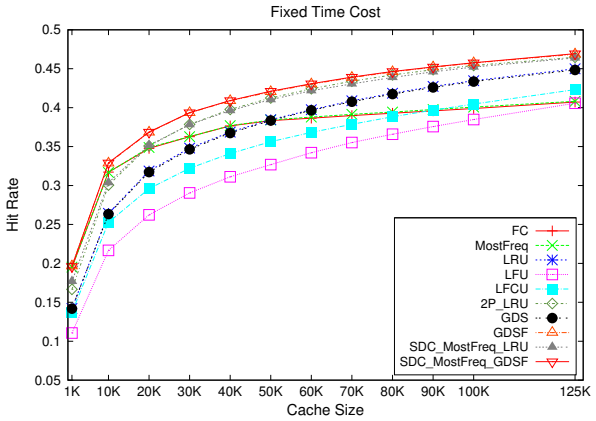


**Figure 5: Hit rates of caching policies assuming fixed query processing time costs.**

by the cost-based policies (`LFCU`, `GDS`, `SDC_MostFreq_GDSF`, `GDSF`) in terms of the financial cost metric. The proposed `2P_LRU` policy incurs lower financial costs than the `LRU` policy. The reductions in financial cost reach up to 3.8% and 1% for small and large cache sizes, respectively. The best policies, based on the financial cost metric, namely `GDSF` and `SDC_MostFreq_GDSF`, outperform `LRU` by around 4%.

Figs. 4 and 5 show the performance for the fixed time cost scenario. For large caches, ordering of policies in decreasing financial cost and increasing hit rate are as follows:

- Financial cost: `LFU` $>$ `MostFreq` $\cong$ `FC` $\cong$ `LFCU` $>$ `GDS` $\cong$ `LRU` $>$ `2P_LRU` $\cong$ `SDC_MostFreq_LRU` $>$ `GDSF` $\cong$ `SDC_MostFreq_GDSF`
- Hit rate: `LFU` $<$ `MostFreq` $\cong$ `FC` $\cong$ `LFCU` $<$ `GDS` $\cong$ `LRU` $<$ `2P_LRU` $\cong$ `SDC_MostFreq_LRU` $<$ `GDSF` $\cong$ `SDC_MostFreq_GDSF`

We note that, in this case, the two orderings are the same. This is because, when we consider only the electricity price as the cost, the variation between the costs of different queries is not high. Let $P(q)$ be the probability that query $q$ leads to a cache hit. In this case, our objective function is to minimize the sum of all $(1 - P(q)) \times C_q$ values, i.e., the cost of all cache misses. If the variation between the $C_q$ values of different queries is not high, then the objective becomes similar to minimizing the $(1 - P(q))$ function, i.e., increasing the hit rate. Therefore, the hit rate and financial cost metrics are highly correlated in this case.

# 5. CONCLUSION

We proposed the financial cost as a new evaluation metric for query result caches in web search engines. We evaluated the well-known static, dynamic, and hybrid result caching strategies using this metric. In general, we observed that the improvement of cost based strategies in terms of the financial cost metric over cost-unaware strategies is more emphasized when there is a sufficient variation between the costs of queries. The proposed financial-cost-aware LRU cache, `2P_LRU`, outperforms the original LRU strategy. For fixed time costs, `2P_LRU` achieves performance results that are very close to the best performing policies, `GDSF` and `SDC_MostFreq_GDSF`. However, in practice, `2P_LRU` could be a better option as the latter policies require a priority queue structure that results in higher computational complexity.

# 6. REFERENCES


[1] I. S. Altingovde, R. Ozcan, B. B. Cambazoglu, and O. Ulusoy. Second chance: a hybrid approach for dynamic result caching in search engines. In *Proc. of ECIR 2011*, pages 510–516. Springer-Verlag, 2011.

[2] M. F. Arlitt, R. J. F. L. Cherkasova, J. Dilley, and T. Y. Jin. Evaluating content management techniques for web proxy caches. *ACM SIGMETRICS Perform. Eval. Rev.*, 27(4):3–11, 2000.

[3] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. The impact of caching on search engines. In *Proc. of SIGIR 2007*, pages 183–190, 2007.

[4] L. A. Barroso and U. Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2009.

[5] P. Cao and S. Irani. Cost-aware www proxy caching algorithms. In *Proc. of the USENIX Symposium on Internet Technologies and Systems*, pages 18–18, 1997.

[6] T. Fagni, R. Perego, F. Silvestri, and S. Orlando. Boosting the performance of web search engines: caching and prefetching query results by exploiting historical usage data. *ACM Trans. Inf. Syst.*, 24:51–78, 2006.

[7] Q. Gan and T. Suel. Improved techniques for result caching in web search engines. In *Proc. of WWW 2009*, pages 431–440, 2009.

[8] E. Kayaaslan, B. B. Cambazoglu, R. Blanco, F. P. Junqueira, and C. Aykanat. Energy-price-driven query processing in multi-center web search engines. In *Proc. of SIGIR 2011*, pages 983–992. ACM, 2011.

[9] M. Marin, V. Gil-Costa, and C. Gomez-Pantoja. New caching techniques for web search engines. In *Proc. of HPDC 2010*, pages 215–226. ACM, 2010.

[10] E. Markatos. On caching search engine query results. *Comput. Commun.*, 24(2):137–143, Feb. 2001.

[11] R. Ozcan, I. S. Altingovde, and O. Ulusoy. Cost-aware strategies for query result caching in web search engines. *ACM Trans. Web*, 5, 2011.

[12] G. Pass, A. Chowdhury, and C. Torgeson. A picture of search. In *Proc. of InfoScale 2006*, page 1, 2006.

[13] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs. Cutting the electric bill for internet-scale systems. In *Proc. of SIGCOMM 2009*, pages 123–134. ACM, 2009.