From Web Cache to Cloud Cache

Thepparit Banditwattanawong

Information Science Institute of Sripathum University
Bangkok, Thailand
thepparit.ba@spu.ac.th

Abstract. To run off-premise private cloud, consumer needs budget for public cloud data-out charge. This amount of expenditure can be considerable for data-intensive organization. Deploying web cache can prevent consumer from duplicated data loading out of their private cloud up to some extent. In present existence, however, there is no cache replacement strategy designed specifically for cloud computing. Devising a cache replacement strategy to truly suit cloud computing paradigm requires ground-breaking design perspective. This paper presents a novel cloud cache replacement policy that optimizes cloud data-out charge, the overall responsiveness of data loadings and the scalability of cloud infrastructure. The measurements demonstrate that the proposed policy achieves superior cost-saving, delay-saving and byte-hit ratios against the other well-known web cache replacement policies.

Keywords: Cloud computing, cache replacement policy, contemporaneous proximity, cost-saving ratio, window size.

1 Introduction

More organizations are adopting cloud computing paradigm due to several benefits such as low up-front costs, better ubiquity, increased utilization of computing resources and reduced power consumption. These are enabled by statistical multiplexing and risk transferences of over- and under-provisionings through elasticity [1]. Public cloud providers like Amazon Web Services [2], Google AppEngine [3] and Windows Azure [4] currently offer several pricing criteria for building off-premise private clouds [5]. Those similarly include the volume charges of data loaded outgoing of private clouds down into consumer sites. These charges can be tremendous expenditures to the running costs of private clouds of data-intensive organizations. The significance of this problem can be realized through a realistic scenario where the data is transferred through 1 Gbps Metro Ethernet with 50% bandwidth utilization for 8 work hours a day, and 260 workdays per annum, which is 39 TB per month, would cost $44,280 per annum based on the Amazon's data-transfer-out pricing data. This is a representative scenario used throughout this paper.

Due to the fact that most of cloud services especially those of SaaS [5] are accessible via HTTP-supported applications such as web browsers and Web OS

R. Li, J. Cao, and J. Bourgeois (Eds.): GPC 2012, LNCS 7296, pp. 1–15, 2012.

[6], cloud data-out charge can be reduced by deploying a caching proxy on a consumer premise. This avoids as many repeated data loadings as possible by letting caching proxy reply succeeding requests for previously loaded data with the data fetched from local cache, unless spoiled, rather than reloaded from cloud.

Nevertheless, acquiring a caching proxy with sufficient space to cache entire data objects from private cloud might be infeasible for consumer organizations because, on one hand, the data-intensive consumers are supposed to export their huge amounts of business data onto their private clouds to truly benefit from cloud computing notion. On the other hand, the overall business data continues to grow with orders of magnitude as modern enterprises increasingly present their business contents in forms of videos, sounds, pictures and other forms of digital contents like electronic publications. Therefore, caching proxy must be equipped with a cache replacement strategy, such as LRU [7], GDSF [8] and LFU-DA [8] that are all supported by the most widely-used web caching software Squid [9]. Cache replacement policies control the provision of enough room inside limited cache spaces on the fly for caching missed objects.

However, there is no cache replacement policies in present existence that has been designed specifically to minimize cloud data-out charge. Additionally, all of the existing policies aim to maximize hit ratio, which means the frequency of serving small data in no time [7, 10, 11] as the first priority. This notion has been evolving with the advancement of broadband communication technologies, which have obviously made the delays (and stability) of the loadings of the small data objects from remote servers no longer distinguishable from those from local caching proxies. In contrast, fetching big objects such as those of multimedia has still kept users experiencing long delays although caching proxy has been in place since the objects have to potentially be retrieved across the network. The latter situation impedes SaaS's content evolution.

The core contributions of this work include: (1) opening up a new design perspective of cache replacement strategy that breaks new ground to suit cloud computing environment, (2) a new performance benchmark, cost-saving ratio, which can be used to capture the economical efficiency of cache replacement policy, (3) a novel replacement policy based on the principle of contemporaneous proximity and optimized for cost-saving, delay-saving and byte-hit ratios to be particularly of use in the era of cloud computing, and (4) a set of comparative measures of strategies in use worldwide including the proposed strategy that gives useful hints for developing more sophisticated cache replacement policies in cloud computing era.

The merits of the proposed policy to the communities of private cloud consumers include the reduction of cloud data-out expenditure and overall speeding up of cloud data loading as well as serving faster large data objects. To both private cloud consumers and public cloud providers, the policy is so network bandwidth friendly that it enables more scalable cloud infrastructure.

2 Proposed Strategy

This section describes the design rationales and practicality analysis of the proposed cache replacement policy.

2.1 Design Rationales

The proposed policy lies itself in two principles, temporal affinity and spatial locality, which are referred to together as contemporaneous proximity [12] for the sake of conciseness. Contemporaneous proximity refers to a time and space property indicating that a particular reference to a certain data object is likely to be repeated in short time (i.e. temporal affinity) and that a set of multiple references to a certain data object tentatively leads to another reference to the same object (i.e. spatial locality). The policy captures the manifest degrees of contemporaneous proximity of data objects by factorizing their recencies and frequencies of accesses.

Considering merely access recency and popularity, however, is unable to satisfy optimal data-out charge reduction in a consistent manner. The policy therefore mandates controlling data-out charge factor by explicitly embracing object sizes and data-out charge rate on a per-object-size basis that altogether accumulates data loading expenditure. It is intuitive that object whose monetary cost of transfer is high, if still usuable, should be retained longer in cache to minimize its cost-benefit ratio than inexpensive object.

As another important design facet, shifting into cloud computing paradigm requires that desktop applications be transformed into SaaS model in which requests to the applications are dispatched across the network. This paradigm requirement causes SaaS less responsive as compared to the desktop applications whose requests are received, processed and returned locally. As a result, using SaaS applications encounters network delays that in overall affect organization productivity. To relieve the effect, cache replacement policy for cloud computing ought to parameterize data loading latency in such a way that data object with short loading latency should be replaced before longer one. This allows higher utilizations of slowly loaded objects to improve overall responsiveness.

Next design consideration is time remaining before the expiration of each object. This characteristic is referred to herein as Time-To-Live (TTL). Data objects whose ages have gone nearly or beyond their expirations should be evicted from cache to give space to newly arriving object as the almost stale ones remain lower chances to get referenced than fresher ones.

Based on the above design rationales, the proposed policy works as follows. Whenever available cache space becomes inadequate to store a newly loaded object, the policy formulates a cluster of least recently referenced objects. The number of objects in the cluster is specified by a preset 'window size' value. Given the formulated cluster, the policy subsequently seeks out an object with the lowest current profit to give preference for eviction. The profit value associated with each object i is defined as:

```
algorithm Caching
description Manipulates hits & misses and calls Cloud
input rURL: requested URL
output requested object
declare cd: cache database (hash table with URL keys)
        ro: requested object,  fs: free cache space
begin
  if ((rURL ∈ cd)  ∧  (cd.getObject(rURL) not expired))   //if cache hit occurs
    ro ← cd.getObject(rURL)
    ro.updateFrequency()
    ro.setProfit(ro.getObjectSize() x ro.getChargeRate()
                 x ro.getLoadingLatency() x ro.getFrequency() x ro.getTTL())
    cd.updateObject(ro)
  else     //if cache miss occurs
    Use rURL to load ro from cloud and initialize its properties
    ro.setProfit(ro.getObjectSize() x ro.getChargeRate()
                 x ro.getLoadingLatency() x ro.getFrequency() x ro.getTTL())
    fs ← cd.getFreeSpace()
    if(fs < ro.getObjectSize())
      Cloud(ro, cd)      //invoking Cloud policy here
    cd.putObject(ro)
  return ro
end
```

```
algorithm Cloud
description Implements Cloud replacement policy
input ro: requested object,   cd: cache database (hash table with URL keys)
output -
declare rs: required cache space,  ws: window size
        cdq: cache database (recency-keyed min-priority queue)
        coq: profit-keyed min-priority queue of evictable objects
        eo: evicted object,  fs: free cache space
begin
  rs ← ro.getObjectSize(),     ws ← predetermined value
  cdq ← cd     //building cdq from cd
  if (cd.getTotalNumberOfObjects() < ws)
    ws ← cd.getTotalNumberOfObjects()
  for 1 to ws do
    coq.addObject(cdq.retrieveLeastRecentlyObject())
  do
    eo ← coq.removeMinProfitObject()
    cd.evict(eo)
  while (eo.getSize() + cd.getFreeSpace()) < rs
  fs ← (cd.getFreeSpace() + eo.getSize() − rs)
  cd.setFreeSpace(fs)
end
```

Fig. 1. Cloud (below) and related (top) algorithms

$$s_i \text{ x } c_i \text{ x } l_i \text{ x } f_i \text{ x } TTL_i$$

where s_i is the size of i, c_i is data-out charge rate in loading i, l_i is latency in loading i, f_i is i's access frequency, and TTL_i is the TTL of i. If revoked cache space is still not sufficient for the new object, additional objects with least profits are evicted in order. Note that cache miss on a highly profitable object imposes more penalty in terms of technical and/or economical efficiencies than a low profitable one.

The proposed policy is entitled 'Cloud' to imply its intended application domain. One possible algorithm solving the problem in choosing object(s) for eviction according to Cloud policy is shown in Fig. 1 together with a caller algorithm. It should be realized that in practice data-out charge rates for all objects to be loaded from clouds are preconfigured values provided by cloud providers from which the objects are loaded [2–4], while TTL values can be calculated from the values of 'Expires' or 'max-age' fields available inside HTTP message headers [13].

2.2 Practicality

With respect to the time complexity analysis of the algorithm of Cloud illustrated in Fig. 1, the statements that take significant part in processing time are: building the priority queue *cdq* from *cd* is traditionally $O(NlogN)$ where N is the number of data objects in a cache; the `for` loop takes $O(NlogN)$ as the window size can be set to as many as N while adding each object into *coq* is $O(logN)$; the `do` loop has the worst-case running time of $O(NlogN)$ because the number of evicted objects is bounded by N, while removing each object from *coq* takes $O(logN)$; deleting an object from the hash table *cd* is less significant and thus necgleted. The other statements are all identically $O(1)$. Therefore, the algorithm is $O(NlogN)$. In other words, Cloud strategy can be implemented with an algorithm whose worst-case running time is guranteed to be practical.

3 Performance Evaluation

This section describes the simulation configuration followed by comparative performance results and discussion of Cloud policy as well as another three popular policies: LRU, GDSF and LFU-DA, which have been supported by Squid caching proxy.

3.1 Input Data Sets

HTTP trace-driven simulation technique has been used for performance measurements. Provided by IRCache project [14], raw trace files are various in sizes and have been collected from three caching proxy servers located in Boulder (BO), Silicon Valley (SV) and New York (NY).

Each of the raw traces contains the stream of requests to large numbers of various HTTP domains. In order to emulate realistic HTTP accesses occuring

on private cloud(s) of a single midsize organization where totally 50 domains are running on the cloud(s), the traces have been preprocessed by counting up top 50 popular domains, and only the requests to these domains have been extracted into three new trace files. As a remark, the number 50 is the approximation of the number of domains administrated by the author's university.

As the other part of preprocessing, unused fields have been removed and an expiration field has been added to every record of every trace to be used to compute TTL values. Expiration field values have been figured out based on three following assumptions. First, an object expired right before its size changed as appeared in a trace. Second, as long as its size was constant, an object's lifespan was extended to its last request appearing in a request stream. Finally, an object appearing only once throughout a trace expired right after the only its use seen in a trace.

Table 1. Characteristics of each of the simulated traces

Traces	BO	SV	NY
Total requests	205,226	441,084	599,097
Requested bytes	2,401,517,003	7,113,486,583	4,712,041,132
Unique objects	70,944	248,508	158,552
Max. total bytes of unique objects	694,759,006	1,065,863,067	1,323,954,264

Table 1 summarizes the basic characteristics of the preprocessed traces. The 'Total requests' designates the total number of records contained in each trace as the results of top 50 domain filterings. The 'Requested bytes' is the total size of requested objects appearing in each trace. The 'Unique objects' represents the number of unique URLs appearing in each trace. As some unique objects had their sizes changed from time to time, by considering only their largest sizes, the 'Max. total bytes of unique objects' indicates minimum cache sizes without cache replacement at all (equivalent to infinite cache sizes).

3.2 Performance Metrics

The three traditional performance metrics, hit rate, byte-hit rate, delay-saving ratio, and the newly proposed economical performance metric 'cost-saving ratio' have been used. For an object i,

$$\textit{cost-saving ratio} = \sum_{i=1}^{n} c_i s_i h_i / \sum_{i=1}^{n} c_i s_i r_i$$

where c_i is the data-out charge rate of i, s_i is the size of i, h_i is how many times a valid copy of i is found in a cache, and r_i is the total number of requests to i.

This study has aimed for the best cost-saving ratio, delay-saving ratio and byte-hit rate, respectively, except hit ratio as justified in Sect. 1. Whilst it is clear why using cost-saving ratio, delay-saving ratio captures how responsive

SaaS would be in overall as the result of a certain cache replacement policy; byte-hit rate captures how good each particular policy foster cloud infrastructure's scalability by reducing as many total bytes transmitted across the network as possible.

3.3 Cost Models

For critical business such as hospital and stock trading, it is not acceptable to experience cloud downtimes and bottlenecks. Consumer organization of this kind must establish continuity plan by implementing private cloud running on more than one independent public cloud to achieve fault tolerance and load balancing. As a consequence, if public cloud providers offer different data transfer prices, objects of the same size loaded from different providers will have different monetary costs.

To realize this practice, the simulations have been conducted based on two cost models. One is uniform cost model where a single data-out charge rate is applied to organization who rents its private cloud from single public cloud provider. The rate of Amazon S3's, which is $0.117997 per GB by average (for the total amount of data transfer out between 11 to 51 TB per month in the US region as of August 2011), has been used in this model. (The range of 11 to 51 TB per month can cover the realistic scenario demonstrated in Sect. 1.) The other is nonuniform cost model, which employs dual charge rates to emulate situation where organization implements its private cloud(s) rented from a pair of independent public cloud providers. The rates used in the latter model are those of Amazon S3's $0.117997 per GB and Windows Azure's $0.15 per GB (for data transfers from North American locations as of August 2011). The simulator has associated the dual charge rates with unique objects found throughout the traces in an interleaving manner.

3.4 Window Sizes

Since data objects in different communities of interests manifest different degrees of contemporaneous proximity, it is not sensible to assume that any recency- and/or frequency-based policy performing perfectly in one environment will perform well against any other environments or even the same environment in different time periods. The control parameter window size is thus engaged to allow the fine tuning of Cloud policy to be adaptive and perform fairly well in any real working environments. In addition to the levels of contemporaneous proximity exhibiting in each workload, the superior value of window size is affected by cache size: a series of pre-experiments have shown that the larger the absolute cache size, the larger the optimal window size. Table 2 presents a set of fine-tuned window sizes (and relative ones inside the parentheses) used in the simulations against each workload and cache size regardless of the cost models. The simulated cache sizes are presented in percents of the maximum total bytes of unique objects belonging to each workload. At 100% cache size, there is no replacement at all, thus all the

policies yield the same upper-bound performance results in all metrics. The rightmost column provides absolute cache sizes in relation to those of BO workload that can be all used in conjunction with the percent cache sizes as a guideline to tune up optimal window sizes in other target environments.

Table 2. Optimal window sizes used in simulations of Cloud policy

Workloads	Simulated cache sizes (% of Max. total bytes of unique objects)			Relative absolute cache sizes
	10%	20%	30%	
BO	215(1.00,1.00)	625(1.00,2.91)	675(1.00,3.14)	1.00
SV	425(1.98,1.00)	1175(1.88,2.76)	1200(1.78,2.82)	1.53
NY	700(3.26,1.00)	1550(2.48,2.21)	1575(2.33,2.25)	1.91

3.5 Empirical Results

The simulation results of Cloud and the other three policies (LRU, GDSF and LFU-DA) are compared in this section. As for GDSF, its particular version called GDSF-Hits (whose cost parameter is equal to 1 for all objects) has been employed. It should also be noted that, unlike some previous works, uncacheable requests have not been excluded from the simulated traces to reflect actual performance ones can really gain from utilizing those certain replacement policies; the caching efficiencies of all the simulated policies would otherwise be improved in all the performance metrics but spurious.

Fig. 2 shows the economical performances rendered by using cost-saving ratio metric. The following findings can be drawn.

- As the main achievement of this study, it can be seen that Cloud has most economized among the other examined policies at all the investigated cache sizes, cost models and workloads (exceptions have lain in 10% cache size of BO workload where LFU-DA has outperformed Cloud slightly by about 0.14% of the Cloud's for both cost models). To realize the merit of Cloud policy implied by its superior performance, the cost-saving ratio of Cloud at 30% cache size of NY workload in the uniform cost model, when applied to the representative scenario in Sect. 1 can significantly save up to $10,569 per annum. Cloud could even save up to $427.26 annually, more than GDSF when using 10% cache size based on SV workload and the uniform cost.
- The cost-saving performances of LRU and LFU-DA have been closely alike, whereas GDSF has performed worst. This is because only GDSF chooses big objects to be replaced at first. To facilitate economical comparisons, 0.001 margin of the cost-saving ratios can be translated as $44.28 per annum as of the representative scenario.

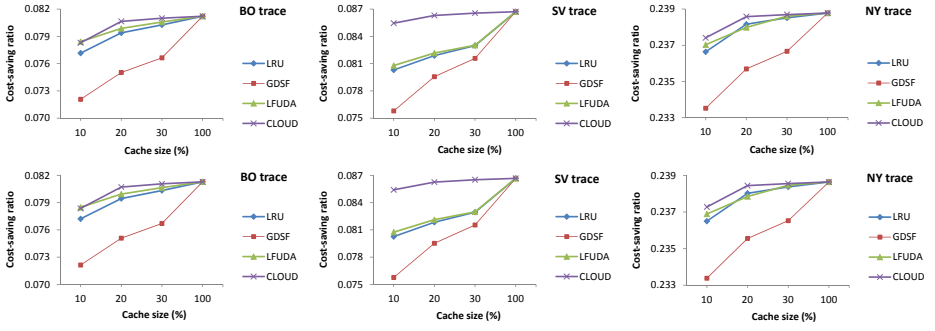

Fig. 2. Comparisons of cost-saving ratios using uniform cost (top row) and nonuniform cost (bottom row)

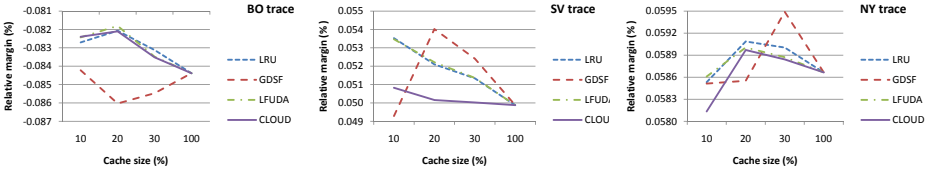

Fig. 3. Comparisons of relative margins of the cost-saving ratios of the uniform cost above the nonuniform one

- For both cost models and all the examined workloads, Cloud has produced the outstanding steady states of cost-saving ratios across the broad range of cache sizes when allocated beyond 20%.
- With the simulated values of data-out charge rates, the cost-saving performance gaps between the uniform and nonuniform cost models have came out subtle and thus magnified in Fig. 3. The relative margins are the cost-saving ratios of uniform cost deducted by those of nonuniform ones in percents of those of the nonuniform costs. The figure has demonstrated that the cost-saving performances of all the policies using the uniform cost model could be slightly better in SV and NY workloads and worse in BO workload than those in the nonuniform cost model. Further observation on these margins will be presented numerically at the end of this section.

With respect to delay saving, the simulation results are portrayed in Fig. 4. The findings from the results are as follows.

- For 20% or larger cache size, Cloud has achieved the best overall responsiveness of data loadings among the others since Cloud has considered retaining slowly loaded objects. This is the minor accomplishment of this work. To translate a merit implied by Cloud policy's superior performance, the delay-saving ratio of Cloud at 30% cache size using BO workload with the uniform

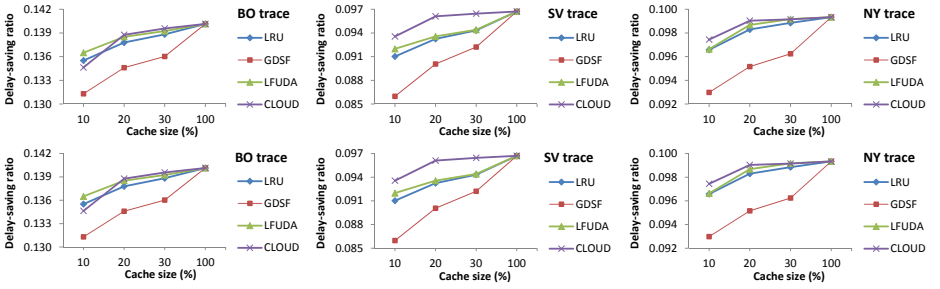

Fig. 4. Comparisons of delay-saving ratios using uniform cost (top row) and nonuniform cost (bottom row)

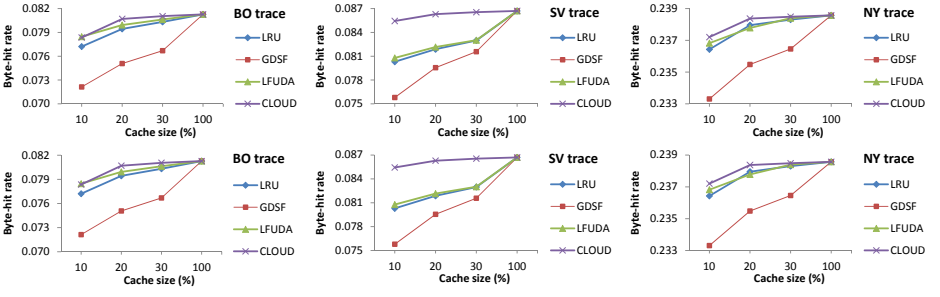

Fig. 5. Comparisons of byte-hit rates using uniform cost (top row) and nonuniform cost (bottom row)

cost, when applied to the representative scenario can significantly save up to around 290 work hours per annum.

- LRU and LFU-DA have delivered similar delay-saving performances, whereas GDSF has saved least total delays. This is because GDSF evicts bigger objects, which generally impose longer loading latencies.
- When cache sizes have been beyond 20% for both cost models and all the examined workloads, Cloud has delivered the most steady delay-saving ratios.
- The differences of delay-saving performances under the same workload between the different cost models have not been recognizable through the ranges of studied cache sizes. Further numerical observation on these differences will be presented at the end of this section.

Fig. 5 demonstrates the byte-hit performances with the following findings.

- Cloud has saved the largest volume of data transfers among the other policies across all the simulated cache sizes, cost models and workloads (exceptions have lain in 10% cache size with the BO workload where LFU-DA has outperformed Cloud slightly by about 0.14% of the Cloud's for both cost

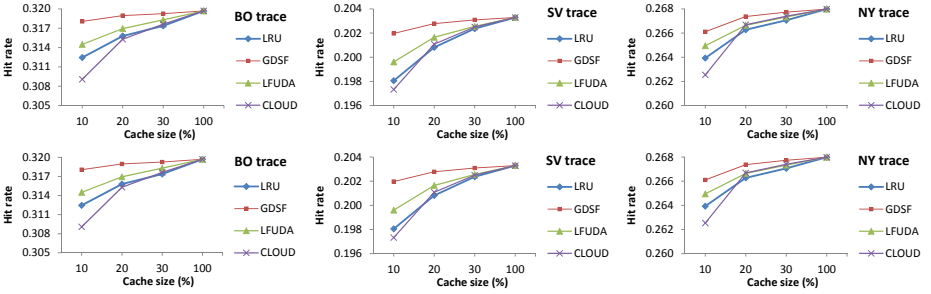

Fig. 6. Comparisons of hit rates using uniform cost (top row) and nonuniform cost (bottom row)

models). This achievement arises because Cloud policy favors large objects to be retained in cache.
- LRU and LFU-DA have delivered similar byte-hit performances, whereas GDSF has performed worst. This is partly because only GDSF evicts bigger objects at first.
- For both cost models and all the examined traces, when cache sizes have grown beyond 20%, Cloud has produced more stable byte-hit rates than the other policies.
- The differences of byte-hit performances of the same workload between the different cost models have not been noticeable through the ranges of investigated cache sizes. Further observation on these differences in terms of numerical data will be presented at the end of this section.

In terms of hit rates, the performances are illustrated in Fig. 6. The below findings have been reached.

- Cloud's performance has been reasonably worst at 10% cache sizes but quickly increased and better than LRU in most other cases and even better than LFU-DA in some cases. This phenomenon can be generally clarified by the fact that a strategy evicting larger objects is optimized for hit rate [7, 10, 11], the opposite applies to Cloud strategy as it tends to retain larger objects in cache.
- Though worst in all previous metrics, GDSF has outperformed all the other policies in hit rate metric. This finding can be explained by the same reason as in the above finding.
- The differences of hit rates of the same workload between the different cost models have not been discernible via the ranges of simulated cache sizes. Further observation on these differences will be presented in the next paragraph.

In a big picture, the following facts have been inferred.

- Since data transfer costs are proportional to object sizes, the cost-saving performances have shown the same growth rates as those of the byte-hit performances meaning that ones can save both data-out costs and network bandwidths simultaneously of the same order of magnitude regardless of utilized policy.
- By looking at Fig. 2, Fig. 5 and Fig. 6 together, the policies that have given higher ratios of cost-saving or byte-hit have tended towards lower hit rates. This behavioral trade-off reinforces the finding that strategy revoking cache space from bigger objects for smaller ones is good at hit rate but poor at byte-hit ratio [7, 10, 11] (and cost-saving ratio).
- Further experiment has revealed that the performance gaps of all kinds of metrics between the uniform and nonuniform cost models will become more noticeable over the wider range of charge rates: using the nonuniform costs of \$0.117997 and \$1.17997 instead of \$0.117997 and \$0.15 at 20% cache size under the NY workload, Cloud has delivered the cost-saving, delay-saving, byte-hit and hit rates of 0.00026097, 0.00000348, -0.00000057 and 0.00000334, respectively, lower than those of the uniform cost.
- In terms of cost savings, delay savings and byte hits, Cloud with optimal window sizes running on 20% or more cache size has delivered almost steady-state performance for both cost models as if it was running with an infinite cache size. Therefore, Cloud policy can be characterized by graceful degradation as it has continued to deliver the best performances over the differently constrained cache sizes.

4 Related Work

4.1 Object Sizes, Loading Costs and Access Frequencies

A number of policies surveyed in [7]: LRU, LFU-DA, EXP1, Value-Aging, HLRU, LFU, LFU-Aging, α-Aging, swLFU, SLFU, Generational Replacement, LRU*, LRU-Hot, Server-assisted cache replacement, LR, RAND, LRU-C, Randomized replacement with general value functions, including policies ARC [15], CSOPT [16], LA2U [17], LAUD [17], SEMALRU [18] and LRU-SLFR [19] have not parameterized object sizes. If big objects were requested frequently but often evicted by these policies (as blind to object sizes), caching proxy would have to frequently reload the big objects from their original servers. Therefore, object-size uncontrollable scheme permits unnecessarily poor cost-saving ratios.

Another group of policies surveyed in [7]: GDSF, LRU-Threshold, LRU-Min, SIZE, LOG2-SIZE, PSS, LRU-LSC, Partitioned Caching, HYPER-G, CSS, LRU-SP, GD-Size, GD*, TSP, MIX, HYBRID, LNC-R-W3, LRV, LUV, HARMONIC, LAT, GDSP, LRU-S, including LNC-R-W3-U [20], SE [21], R-LPV [22], Min-SAUD [23], OPT [24], LPPB-R [25], OA [26], CSP [27] and GA-Based Cache Replacement Policy [28] have considered object sizes in such a way that replacing bigger objects first, thus not aiming for cost-saving performance. The other policies M-Metric [7], NNPCR-2 [29] and Bolot and Hoschka's [30] have favored bigger objects like Cloud. In particular, M-Metric allows bigger objects to stay

longer in cache but does not support loading cost parameter; NNPCR-2 applied neural network to decide the evictions of small or big objects but does not embed cost parameter; Bolot and Hoschka's policy replaces bigger objects first but ignores spatial locality by not considering access frequencies and does not support nonuniform costs.

4.2 Access Recencies

All known policies have prioritized the recencies of object references either implicitly or explicitly. By implicitly, every policy always accepts a newly loaded missing object (i.e., the most recently used object) into cache rather than rejects it. By explicitly, several policies such as LRU, LRU-Threshold, SIZE, LRU-Min, EXP1, Value-Aging, HLRU, PSS, LRU-LSC and Partitioned Caching have parameterized elapsed times since the last requests to objects. Cloud policy has explicitly regarded the recency property of objects in its model.

4.3 Object Loading Latencies

Several policies: GD-Size, GDSF, GD*, GDSP, HYBRID, LAT, LUV, MIX, LNC-R-W3, LNC-R-W3-U, LRU-SLFR and GDSP have taken object loading latencies into account. All of them have replaced objects with shorter latencies first. Cloud policy also follows such a design approach.

4.4 Object Expirations

Very rare policy considers object expiration. LA2U, LAUD and LNC-R-W3-U have replaced frequently updated objects first. The former two have not described how update frequencies are derived. The latter has estimated update frequencies from changes detected in HTTP's 'Last-Modified' header fields; however, if frequently updated objects are seldom requested, updated 'Last-Modified' values will be rarely perceived by policies and update frequencies will be then underestimated. This problem can be solved by using explicit expiration times or TTL as in Bolot and Hoschka's policy even though this parameter has not yet been implemented in their empirical

5 Conclusion

This paper addresses an economical and technical perspective from which a new cache replacement policy must be devised specifically for cloud computing era. A simple and efficient policy, Cloud, is proposed. The Cloud's efficiencies in terms of cost-saving, delay-saving and byte-hit ratios except hit ratios (which are justifiable) are found fairly outperforming all the other investigated policies at most cache sizes.

A concrete finding from this study is that if most recently used objects with large sizes, costly charge rates, long loading latencies, high access frequencies, and long lifespans last longer in cache in a profit-inside-recency-window manner, cost-saving, delay-saving and byte-hit performances will be greatly improved.

Left as future work, to compare Cloud policy with others by using both technical and economical performance metrics based on longer traces is challenging and requires considerable effort. Also, we have been planning to conduct a future research to analyze static and dynamic factors as well as their interrelationship to help determine the optimal values of window sizes for a given environment that are dynamically and timely adjusted according to workload evolution.

Acknowledgment. This research is financially supported by Sripatum University. The author would like to thank Duane Wessels, National Science Foundation (grants NCR-9616602 and NCR-9521745) and the National Laboratory for Applied Network Research for the trace data used in this study.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: Above the clouds: A berkeley view of cloud computing. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28 (February 2009), http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html
2. Amazon.com, Inc., Amazon web services (2011), http://aws.amazon.com/s3/
3. Google Inc., Google app engine (2011), http://code.google.com/intl/en/appengine/
4. Microsoft, Windows azure (2011), http://www.microsoft.com/windowsazure/
5. Mell, P., Grance, T.: The NIST definition of cloud computing (draft): Recommendations of the national institute of standards and technology. NIST Special Publication 800-145 (Draft) (2011), http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145_cloud-definition.pdf
6. Wright, A.: Ready for a web os? Commun. ACM 52, 16–17 (2009)
7. Podlipnig, S., Böszörmenyi, L.: A survey of web cache replacement strategies. ACM Comput. Surv. 35, 374–398 (2003)
8. Arlitt, M., Cherkasova, L., Dilley, J., Friedrich, R., Jin, T.: Evaluating content management techniques for web proxy caches. SIGMETRICS Perform. Eval. Rev. 27, 3–11 (2000)
9. Wessels, D.: Squid: The Definitive Guide. O'Reilly & Associates, Inc., Sebastopol (2004)
10. Abrams, M., Standridge, C.R., Abdulla, G., Fox, E.A., Williams, S.: Removal policies in network caches for world-wide web documents. SIGCOMM Comput. Commun. Rev. 26, 293–305 (1996)
11. Balamash, A., Krunz, M.: An overview of web caching replacement algorithms. IEEE Communications Surveys and Tutorials 6(1-4), 44–56 (2004)
12. Banditwattanawong, T., Hidaka, S., Washizaki, H., Maruyama, K.: Optimization of program loading by object class clustering. IEEJ Transactions on Electrical and Electronic Engineering 1 (2006)

13. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Rfc 2616, hypertext transfer protocol – http/1.1, United States (1999)
14. National Laboratory for Applied Network Research, Weekly squid http access logs, `http://www.ircache.net/`
15. Megiddo, N., Modha, D.S.: Outperforming lru with an adaptive replacement cache algorithm. Computer 37, 58–65 (2004)
16. Jeong, J., Dubois, M.: Cache replacement algorithms with nonuniform miss costs. IEEE Transactions on Computers 55, 353–365 (2006)
17. Chen, H., Xiao, Y., Shen, X.S.: Update-based cache access and replacement in wireless data access. IEEE Transactions on Mobile Computing 5, 1734–1748 (2006)
18. Geetha, K., Gounden, N.A., Monikandan, S.: Semalru: An implementation of modified web cache replacement algorithm. In: NaBIC, pp. 1406–1410. IEEE (2009)
19. Shin, S.-W., Kim, K.-Y., Jang, J.-S.: Lru based small latency first replacement (slfr) algorithm for the proxy cache. In: Proceedings of the 2003 IEEE/WIC International Conference on Web Intelligence, WI 2003, pp. 499–502. IEEE Computer Society, Washington, DC (2003)
20. Shim, J., Scheuermann, P., Vingralek, R.: Proxy cache algorithms: Design, implementation, and performance. IEEE Transactions on Knowledge and Data Engineering 11, 549–562 (1999)
21. Sarma, A.R., Govindarajan, R.: An Efficient Web Cache Replacement Policy. In: Pinkston, T.M., Prasanna, V.K. (eds.) HiPC 2003. LNCS (LNAI), vol. 2913, pp. 12–22. Springer, Heidelberg (2003)
22. Chand, N., Joshi, R., Misra, M.: Data profit based cache replacement in mobile environment. In: 2006 IFIP International Conference on Wireless and Optical Communications Networks, p. 5 (2006)
23. Xu, J., Hu, Q., Lee, W.-C., Lee, D.L.: Performance evaluation of an optimal cache replacement policy for wireless data dissemination. IEEE Transactions on Knowledge and Data Engineering 16, 125–139 (2004)
24. Yin, L., Cao, G., Cai, Y.: A generalized target-driven cache replacement policy for mobile environments. In: IEEE/IPSJ International Symposium on Applications and the Internet, p. 14 (2003)
25. Kim, K., Park, D.: Least popularity-per-byte replacement algorithm for a proxy cache. In: Intl. Conf. on Parallel and Distributed Systems, p. 0780 (2001)
26. Li, K., Nanya, T., Qu, W.: A minimal access cost-based multimedia object replacement algorithm. In: International Symposium on Parallel and Distributed Processing, p. 275 (2007)
27. Triantafillou, P., Aekaterinidis, I.: Web proxy cache replacement: Do's, don'ts, and expectations. In: IEEE International Symposium on Network Computing and Applications, p. 59 (2003)
28. Chen, Y., Li, Z.-Z., Wang, Z.-W.: A ga-based cache replacement policy. In: Proceedings of 2004 International Conference on Machine Learning and Cybernetics, vol. 1, pp. 263–266 (August 2004)
29. El Aarag, H., Romano, S.: Improvement of the neural network proxy cache replacement strategy. In: Proceedings of the 2009 Spring Simulation Multiconference, SpringSim 2009, Society for Computer Simulation International, San Diego (2009)
30. Bolot, J.-C., Hoschka, P.: Performance engineering of the world wide web: application to dimensioning and cache design. Computer Networks and ISDN Systems 28, 1397–1405 (1996)