The Optimality and Complexity of Offline Cache Replacement Policies for Nonuniform Objects

Thepparit Banditwattanawong

***Abstract*—The study of optimal offline cache replacement policies has theoretical significance from which near-optimal but practical cache replacement schemes can be sound established. This paper has re-defined known offline cache replacement policies for nonuniform objects and proposed novel policies. All of them have been evaluated for their optimalities and complexities. The results have showed that the optimal policy has the highest time complexity whereas the other policies are sub-optimal that have lower complexities.**

***Index Terms*—Cache replacement, nonuniform objects, optimal replacement, offline replacement.**

I. Introduction

When cache space is found inadequate to make room for a requested missing object, cache replacement takes place to make a decision, either cached object eviction or the requested object discard. The latter is referred to as *optional eviction* (AKA *bypassing* [1]), which arises from a fact that there is no benefit at all to cache missing objects requested only once.

As the theoretical foundation of online solutions, *offline* [1] cache replacement policies require the provision of the complete sequence of future object requests to be searched for an optimal state sequence, otherwise, ones need nondeterministic machines to execute the policies to process a real time request sequence by always choosing the right choices of next cache states toward a final optimal one.

A. Performance Metric for Nonuniform Object Costs

Web cache replacement policies with mandatory eviction have been extensively surveyed in [2], [3]. All of them aimed to optimize hit rate by not evict small cached objects at first for two reasons. Firstly as stated in our previous work [4], fetching small objects through the Internet in the past was so delayed that it was unacceptable for users to experience such delays on every request. Secondly and partly supported by [5], hit rate is an accustomed and popular metric because it could reflect at the same time the shortened access latency and saved transfer bandwidth of cache replacement policies for uniform caches (e.g., CPU caches) in which objects have identical size and read latency.

On the other hand, big objects have been being populated on the Internet as substantiated by average bytes per web page that increase year by year (by 13.66% from 2015 in 2016 and 36.13% KB from 2016 in 2017) [6]. Loading a number of ever-growing big objects from remote servers causes network congestion and subsequently access delays as well as huge data transfer charges [7] whereas fetching remote small objects at present Internet speed is fast as if they were available in user locality. In other words, the manifested impact of nonuniform objects whose costs, such as object sizes, downloading latencies, or data transfer monetary charges (typically proportional to the object sizes [7]), has been becoming so prevalent that modern web cache replacement policies should optimize for. Hit rate is thus no longer a reliable caching performance metric for nonuniform objects. Instead, a more generic performance goal, which has been used throughout this paper, is a *cost-saving ratio* (*CSR*).

Definition 1: *CSR* is defined as follows where n is the number of objects in a cache, c_i is the cost of fetching an object i from its remote server by the cache, h_i is the number of times the valid copy of i is fetched from the cache to serve requesting clients, and r_i is the total number of requests to i.

$$CSR = \frac{\sum_{i=1}^{n} c_i h_i}{\sum_{i=1}^{n} c_i r_i}$$

CSR also represents byte-hit ratio and delay-saving ratio by substituting the object fetching cost's value for object size and object downloading latency, respectively. The byte-hit ratio should be of interest when bandwidth is scarce whereas the delay-saving ratio should be of interest when access latency is restricted.

B. Overall Organization

This paper is discrete mathematical in nature instead of empirical. The paper begins with the aforementioned rationale for using the *CSR* rather than the hit rate as the optimization goal of cache replacement policies for nonuniform object costs. Section II re-defines an optimal offline cache replacement policy for nonuniform objects to support optional eviction to achieve the optimal cost-saving goal. Section III presents related works, offline cache replacement policies that were claimed to be optimal, along with their sub-optimality and the complexity proofs. The time complexity analysis enables the derivation of practical online cache replacement policies. We have also proposed the novel sub-optimal policies of lower or comparable complexities. Finally, findings and our future plan are summarized in order.

II. Optimal Offline Cache Replacement Policy with Optional Eviction for Nonuniform Objects

Hosseini−Khayat's OPT [8] has been proved in terms of *CSR* to be an optimal offline cache replacement policy with mandatory eviction for nonuniform objects. He has also proposed that the modification of the policy for optional eviction is optimal based on the principle of optimality.

Manuscript received June 15, 2018; revised August 1, 2018.

The author is with Computer Science Department, Faculty of Science, Kasetsart University, Bangkok, Thailand (e-mail: thepparit.b@ku.th).

doi: 10.18178/ijfcc.2018.7.3.522

Herein, we have re-defined the latter policy variant called *Hosseini−Khayat's OPT** algorithm and restated its optimality in Corollary 1.

Corollary 1: *Hosseini−Khayat's OPT* is an optimal offline cache replacement policy for nonuniform objects.*

To find the optimal multi-state cache replacement decision of nonuniform objects for the sequence of m requests since the first cache replacement decision occurs, possible cache state transitions is represented as a weighted directed acyclic graph $G = (V, E)$ consists of a set of vertices V representing cache states and a set of edges E denoting state transition due to cache hit or miss. Each edge $(v', v) \in E$ where $v', v \in V$, connecting a vertex v' to its adjacent vertex v is assigned a weight $c(v', v)$ representing a nonuniform cost, either 0 (resulting from cache hit or optional eviction) or a missing object fetching cost. Each vertex v is assigned an optimal cost $c(v)$, which is the total weight of the shortest weighted path from a root vertex (representing a cache state where free cache space is found smaller than a missing object size for the first time) and represents the least accumulated cost of fetching missing objects across consecutive cache states from the root to the current one. Let organize G into m layers, excluding a root layer; and let V_i, $\lambda(v)$, $\Lambda(v)$, and $\Gamma(v)$ denote the set of all vertices in layer i ($0 \le i \le m$ where 0 is the root layer, which contains a single vertex), the optimal parent of v, the set of parents of v, and the set of all possible child vertices of v, respectively. *Hosseini – Khayat's OPT** is shown in Table I, which is initially activated in state $v \in V_0$ (i.e., available cache space is found insufficient for the first time). The $(v^*, \lambda(v^*), \lambda(\lambda(v^*)), ..., v \in V_0)$ is an optimal state sequence in reverse chronological order, and the $c(v^*)$ is the least total cost of missing-object fetching against the sequence of entire m requests.

Proposition 1: *Hosseini−Khayat's OPT* has the worst-case running time bound of* $O(n^{3m})$.

TABLE I: HOSSEINI – KHAYAT'S OPT* ALGORITHM

algorithm: *Hosseini−Khayat's OPT**

```
begin
  c(v ∈ V0) ← 0
  i ← 0 /* current layer the algorithm is invoked for the first
time */
  while i ≤ m
  {
    for each v ∈ Vi
    {
      if i < m /* if true, generate all possible child vertices */
        Vi+1 ← Vi+1 ∪ Γ(v) ∪ v /*optional eviction causes ∪ v */
      if i ≠ 0
        c(v) ← min_{v∈Λ(v)}(c(v') + c(v', v))
    }
    i ← i + 1
  }
  v* ← argmin_{v∈Vm}(c(v)) /* a final optimal state */
return (v*, λ(v*), λ(λ(v*)), ..., v ∈ V0) and c(v*)
```

Proof: The running time complexity of *Hosseini−Khayat's OPT** is bound to the worst-case number of edges, $|E|$ in G. For each given cache state comprising n cached objects, since zero or more cached objects might be evicted at once in all possible combination to achieve the least cost, this produces at maximum 1 (i.e., evict none from cache due to either cache hit or optional eviction) +1 (for evicting all cached objects) outgoing edges for n=1, 1 (for evicting none) +n (for evicting all cached objects exclusively) +1 (for evicting all cached objects) outgoing edges for n=2, or $1+n+1+\sum_{l=1}^{n-2}\frac{n(n-l)}{2}$ for $n \ge 3$ equal to $2+n+\frac{n(n-2)(n+1)}{4}$ outgoing edges. Subsequently, the worst-case number of incoming edges to all vertices in a layer $i \ge 1$ is $(2+n+\frac{n(n-2)(n+1)}{4})^i$. Thus, the worst-case $|E|$ is $\sum_{i=1}^{m}(2+n+\frac{n(n-2)(n+1)}{4})^i$, equal to $\frac{(2+n+\frac{n(n-2)(n+1)}{4})^{m+1}-1}{2+n+\frac{n(n-2)(n+1)}{4}-1}-1$. Hence, *Hosseini−Khayat's OPT** = $O(n^{3m})$.

III. SUB-OPTIMAL OFFLINE CACHE REPLACEMENT POLICIES FOR NONUNIFORM OBJECTS

A. Breadth Limiting (BL)

BL replacement policy [8] approximated *Hosseini−Khayat's OPT*.

Definition 2: *BL* generates child vertices only for k vertices in each nonroot layer that have the least costs rather than for all vertices.

Proposition 2: *BL has the worst-case running time of* $O(mn^3\log_2 n^3)$.

Proof : For each given cache state, since zero (due to cache hit) or more cached objects might be evicted, this produces at most $2+n+\frac{n(n-2)(n+1)}{4}$ outgoing edges for $n \ge 3$. Thus, maximum $|V_1|$ is equal to$2+n+\frac{n(n-2)(n+1)}{4}$. However, *BL* selects only k least cost child vertices that at the layer 1 requires $|V_1|\ \log_2|V_1|$ sorting and the reading of k vertices, hence equal to $|V_1|\ \log_2|V_1| + k$. Since $|V_{i\ge 2}| = k|V_1|$, to reduce $|V_i=2|$ to k takes $k|V_1|\ \log_2(k|V_1|) + k$ time units. Subsequently, the worst-case work of *BL* is $(|V_1|\ \log_2|V_1|+k)+((m-1)(k|V_1|\ \log_2(k|V_1|)+k))$. Hence, *BL* is $O(mn^3\log_2 n^3)$.

B. Periodic Elimination (PE)

PE replacement policy [8] approximated *Hosseini−Khayat's OPT*.

Definition 3: *PE* generates child vertices for only a vertex, which has the least cost, in every T layers where T is an elimination period.

Proposition 3: *PE has the worst-case running time of* $O(mn^{3T})$.

Proof : The running time complexity of *PE* is bound to the worst-case $|E|$ plus the least cost vertex searching. For each given cache state, since zero or more cached objects might be evicted, this produces at most $2+n+\frac{n(n-2)(n+1)}{4}$ outgoing edges for $n \ge 3$. Subsequently, $|V_{i\ge 1}| = (2+n+\frac{n(n-2)(n+1)}{4})^i$. In every T layers, the least cost vertex is obtained by scanning $(2+n+\frac{n(n-2)(n+1)}{4})^T$ vertices. Thus, the worst-case $|E|$ plus scanning time is $\frac{m}{T}(\sum_{i=1}^{T}(2+n+\frac{n(n-2)(n+1)}{4})^i + (2+n+\frac{n(n-2)(n+1)}{4})^T)$, equal to $\frac{m}{T}((\frac{(2+n+\frac{n(n-2)(n+1)}{4})^{T+1}-1}{2+n+\frac{n(n-2)(n+1)}{4}-1}-1)+(2+n+\frac{n(n-2)(n+1)}{4})^T))$. Hence, $PE = O(mn^{3T})$.

C. C_0

Definition 4: Let $\mathcal{U}$ be an object universe, $O = \{o_j \in \mathcal{U} \mid o_j$ *is a unique cacheable object* for $j = 1, 2, ..., N \wedge N$ *is the total*

number of o_j}, p_j be the probability (i.e., frequency) of o_j to be requested, c_j be the cost of fetching o_j from an original server to a cache, and $V_t \subset O$ be the set of objects in the cache at time *t*. Suppose that cache replacement takes place at *t*, C_0 replacement policy [9] (originally named *OPT−C* [10]) evicts o_j iff ($j = argmin(p_jc_j)$) ∧ ($o_j \in V_t$).

Theorem 1: *C_0 is a suboptimal offline cache replacement policy for nonuniform objects.*

Proof : We have conducted the proof by counterexample in terms of *CSR*s as follows. Let **P** be the sequence of requests to cacheable objects in {$o_j \mid c_j = j \wedge c_j$ *is unique*}. Let **C** be the sequence of saved costs resulting from cache hits with respect to **P** and based on an assumption that there is no update of o_j throughout the course of **P**. Given **P** = (o_3, o_2, o_1, o_2, o_3) where every o_j has the identical size of 1 and the cache size of 2, C_0 works in such a way that resulting in the following cache states:

$$(\emptyset, \{o_3\}, \{o_3, o_2\}, \{o_3, o_1\}, \{o_3, o_2\}, \{o_3, o_2\})$$

Subsequently, **C** = (0,0,0,0,3) resulting in $CSR = \frac{3}{11}$. Nonetheless, the optimal sequence of cache states is ($\emptyset$, {o_3}, {o_3, o_2}, {o_3, o_2}, {o_3, o_2}, {o_3, o_2}) that results in **C** = (0,0,0,2,3) and $CSR = \frac{5}{11}$. Consequently, we have shown that "C_0 is optimal for nonuniform objects" is false.

Proposition 4: *C_0 has the worst-case running time of* $O(mn\log_2 n + m^2n)$.

Proof: For the 1st cache replacement, each cached object in {$o_i \mid i = 1, 2, ..., n$} is searched throughout the remaining $m - 1$ requests to figure out p_i and c_i. Additionally when a single o_i eviction cannot make enough room for the first missing object, {$p_ic_i | i = 1, 2, ..., n$} is sorted in $n \log_2 n$ to evict at most n o_i in an ascending p_ic_i order, taking another $n \log_2 n$. Therefore, the first cache replacement requires $n(m-1) + n \log_2 n + n \log_2 n$ work at maximum. Similarly, for the 2nd cache replacement, each o_i is searched in the remaining $m-2$ requests, adding a new object into the sorted cache requires $\log_2 n$, and at most $n \log_2 n$ eviction takes place, and so forth. Therefore, the maximum time required is ($n(m-1) + n \log_2 n + n \log_2 n$) + ($n(m-2) + \log_2 n + n \log_2 n$) +...+ ($n + \log_2 n + n \log_2 n$) or ($2n + (m-2)(n+1)$) $\log_2 n + n\sum_{i=1}^{m-1} i$ equal to $(2n+(m-2)(n+1)) \log_2 n + \frac{(m-1)mn}{2}$. Hence, $C_0 = O(mn\log_2 n + m^2n)$.

D. C_0^*

Definition 5: Let o_t be a missing object requested at time *t*. Suppose that cache replacement takes place at *t*, C_0^* replacement policy [9] evicts o_j iff ($j = argmin(p_jc_j)$) ∧ ($o_j \in V_t \cup \{o_t\}$).

E. *Theorem 2: C_0^* Is a Suboptimal Offline Cache Replacement Policy for Nonuniform Objects*

Proof: Given **P** = (o_4, o_1, o_1, o_1, o_1, o_4, o_1, o_4, o_1, o_1, o_1, o_1) where every o_j has the identical size of 1 and a cache size is equal to 1, C_0^* can produce four possible sequences of cache states as some to-be-requested objects hold the same values of p_jc_j :

($\emptyset$, {o_4}, {o_4}, {o_4}, {o_4}, {o_4}, {o_4}, {o_4}, {o_4}, {o_1}, {o_1}, {o_1}, {o_1}),
($\emptyset$, {o_4}, {o_4}, {o_4}, {o_4}, {o_4}, {o_4}, {o_1}, {o_1}, {o_1}, {o_1}, {o_1}, {o_1}),
($\emptyset$, {o_4}, {o_1}, {o_1}, {o_1}, {o_1}, {o_1}, {o_1}, {o_1}, {o_1}, {o_1}, {o_1}, {o_1}),

or the worst state sequence:

($\emptyset$, {o_4}, {o_1}, {o_1}, {o_1}, {o_1}, {o_1}, {o_1}, {o_4}, {o_1}, {o_1}, {o_1}, {o_1})

Of the last state sequence, **C** = (0,0,1,1,1,0,1,0,0,1,1,1) resulting in $CSR = \frac{7}{21}$ even though the first sequence's *CSR* = $\frac{11}{21}$ and the second and the third sequences' *CSR* = $\frac{8}{21}$. Nonetheless, the optimal sequence of cache states is ($\emptyset$, {o_4}, {o_4}, {o_4}, {o_4}, {o_4}, {o_4}, {o_4}, {o_1}, {o_1}, {o_1}, {o_1}) that results in **C** = (0,0,0,0,4,0,4,0,1,1,1) and $CSR = \frac{11}{21}$. Consequently, this counterexample has shown that "C_0^* is optimal for nonuniform objects" is false. As a remark, C_0^* will produce an optimal state sequence only on a nondeterministic machine, otherwise, C_0^* (to run offline on deterministic machines) has to ensure that optimal choices are always chosen when encounters the identical p_jc_j values of different o_j.

Proposition 5: *C_0^* has the worst-case running time of* $O(mn\log_2 n + m^2n)$.

Proof: For the 1st cache replacement, each object in $V_t \cup \{o_t\}$ is searched throughout the remaining $m - 1$ requests. As $|V_t| = n$, the replacement imposes totally $(n+1)(m-1)$ comparisons plus $n \log_2 n$ sorting plus n cached object eviction at worst. Similarly, for the 2nd cache replacement, each object in $V_t \cup \{o_t\}$ is searched in the remaining $m-2$ requests, adding a newly missing object, and at most n cached object eviction takes place, and so forth. Therefore, total time to resolve all of the requests is at most ($(n + 1)(m- 1) + n\log_2 n + n \log_2 n$) + (($n+1)(m-2) + \log_2 n + n \log_2 n$) + ... + (($n + 1) + \log_2 n + n \log_2 n$) equal to $(2n + (m-2)(n+1)) \log_2 n + \frac{(m-1)m(n+1)}{2}$. Hence, $C_0^* = O(mn\log_2 n + m^2n)$.

F. *Mattson's OPT*

Mattson's OPT replacement policy has long been known to be optimal for uniform objects [11], [12]. It is also known as longest forward distance policy [8].

Definition 6: *Mattson's OPT* evicts $o_j \in V_t$ if *o_j has no more request* else if *o_j has next request furthest in time*.

Theorem 3: *Mattson's OPT is a suboptimal offline cache replacement policy for nonuniform objects.*

Proof: Given **P** = (o_3, o_5, o_4, o_3, o_5) where every o_j has c representing object size and a cache size is equal to 10, *Mattson's OPT* produces two following equivalent sequences of cache states:

($\emptyset$, {o_3}, {o_3, o_5}, {o_3, o_4}, {o_3, o_4}, {o_3, o_5}) or
($\emptyset$, {o_3}, {o_3, o_5}, {o_3, o_4}, {o_3, o_4}, {o_5, o_4})

Thus, **C** = (0,0,0,3,0) resulting in $CSR = \frac{3}{20}$. However, the optimal sequence of cache states is ($\emptyset$, {o_3}, {o_3, o_5}, {o_4, o_5}, {o_3, o_5}, {o_3, o_5}) that results in **C** = (0,0,0,0,5) and $CSR = \frac{5}{20}$. This proof by counterexample has shown that "*Mattson's OPT* is optimal for nonuniform objects" is false.

Proposition 6: *Mattson's OPT has the worst-case running time of* $O(mn\log_2 n + m^2n)$.

Proof: For the 1st cache replacement, each object in V_t is searched in the remaining request sequence at most for $m-1$

requests to determine its furthest distance in the sequence, and sorted for $n \log_2 n$ and evicted for n cached objects at worst. Likewise, for the 2^{nd} cache replacement, each object in V_t is searched in the remaining $m - 2$ requests, adding a newly missing object, and at most n cached object eviction takes place. Therefore, the maximum time required is $(n(m-1) + n\log_2 n + n \log_2 n) + (n(m-2) + \log_2 n + n \log_2 n) + ... + (n + \log_2 n + n \log_2 n)$, equal to $(2n+(m-2)(n+1)) \log_2 n + \frac{(m-1)mn}{2}$. Hence, *Mattson's OPT* is $O(mn\log_2 n + m^2 n)$.

G. OPT^*

The first of our novel algorithms has extended *Mattson's OPT* mainly to allow optional eviction.

Definition 7: OPT^* evicts $o_j \in V_t \cup \{o_t\}$ if o_j *has no more request* else if o_j *has next request furthest in time*.

Theorem 4: OPT^* *is a suboptimal offline cache replacement policy for nonuniform objects.*

Proof: Given $\mathbf{P} = (o_4, o_1, o_1, o_1, o_4)$ where every o_j has the identical size of 1 and a cache size is equal to 1, OPT^* produces the following sequence of cache states:

$$(\emptyset, \{o_4\}, \{o_1\}, \{o_1\}, \{o_1\}, \{o_4\})$$

It yields $\mathbf{C} = (0,0,1,1,0)$ resulting in $CSR = \frac{2}{11}$. However, the optimal sequence of cache states is $(\emptyset, \{o_4\}, \{o_4\}, \{o_4\}, \{o_4\}, \{o_4\})$ that results in $\mathbf{C} = (0,0,0,0,4)$ and $CSR = \frac{4}{11}$. This has shown that "OPT^* is optimal for nonuniform objects" is false.

Proposition 7: OPT^* *has the worst-case running time of* $O(mn\log_2 n + m^2 n)$.

Proof: For the 1^{st} cache replacement, each object in $V_t \cup \{o_t\}$ is searched in the remaining request sequence at most for $m-1$ requests to determine its furthest distance in the sequence, and sorted for $n \log_2 n$ and evicted for n cached objects at worst. Likewise, for the 2^{nd} cache replacement, each object in $V_t \cup \{o_t\}$ is searched in the remaining $m - 2$ requests, adding a newly missing object, and at most n cached object eviction takes place, and so forth. Therefore, total time to resolve all of the requests is at most $((n + 1)(m - 1) + n\log_2 n + n \log_2 n) + ((n+1)(m-2) + \log_2 n + n\log_2 n) + ...+ ((n+1) + \log_2 n + n \log_2 n)$ equal to $(2n + (m-2)(n+1)) \log_2 n + \frac{(m-1)m(n+1)}{2}$. Hence, OPT^* is $O(mn\log_2 n + m^2 n)$.

H. Shortest Maximum Forward Distance (SMFD)

Our second proposed algorithm evicts from cache an object having the soonest final request.

Definition 8: *SMFD* evicts $o_j \in V_t$ if o_j *has no more request* else if o_j *has the final request the soonest*.

Theorem 5: *SMFD is a suboptimal offline cache replacement policy for nonuniform objects.*

Proof: Given $\mathbf{P} = (o_2, o_5, o_3, o_5, o_2)$ where c in o_c represents object size and a cache size is equal to 8, *SMFD* works in such a way that resulting in the following cache states in order:

$$(\emptyset, \{o_2\}, \{o_2, o_5\}, \{o_2, o_3\}, \{o_2, o_5\}, \{o_2, o_5\})$$

Subsequently, $\mathbf{C} = (0,0,0,0,2)$ resulting in $CSR = \frac{2}{17}$. However, the optimal sequence of cache states is $(\emptyset, \{o_2\}, \{o_2, o_5\}, \{o_3, o_5\}, \{o_3, o_5\}, \{o_2, o_5\})$ or $(\emptyset, \{o_2\}, \{o_2, o_5\}, \{o_3, o_5\}, \{o_3, o_5\}, \{o_3, o_2\})$ that result in the same $\mathbf{C} = (0,0,0,5,0)$ and $CSR = \frac{5}{17}$. Consequently, we have shown that "*SMFD* is optimal for nonuniform objects" is false.

Proposition 8: *SMFD has the worst-case running time of* $O(mn\log_2 n + m^2 n)$.

Proof: For the 1^{st} cache replacement, each object in V_t is searched in the remaining request sequence at most for $m-1$ requests to determine its shortest maximum forward distance in the sequence, and sorted for $n \log_2 n$ and evicted for n cached objects at worst. Similarly, for the 2^{nd} cache replacement, each object in V_t is searched in the remaining $m-2$ requests, adding a newly missing object, and at most n cached object eviction takes place. Therefore, the maximum time required is $(n(m-1) + n \log_2 n + n \log_2 n) + (n(m-2) + \log_2 n + n \log_2 n) + ... + (n + \log_2 n + n \log_2 n)$, equal to $(2n + (m-2)(n+1))\log_2 n + \frac{(m-1)mn}{2}$. Hence, *SMFD* is $O(mn\log_2 n + m^2 n)$.

I. $SMFD^*$

Our lastly proposed algorithm has extended *SMFD* to allow optional eviction.

Definition 9: $SMFD^*$ evicts $o_j \in V_t \cup \{o_t\}$ if o_j *has no more request* else if o_j *has the final request the soonest*.

Theorem 6: $SMFD^*$ *is a suboptimal offline cache replacement policy for nonuniform objects.*

Proof: Given $\mathbf{P} = (o_5, o_2, o_5, o_2)$ where c in o_j represents object size and a cache size is equal to 5, $SMFD^*$ produces the following cache states, respectively:

$$(\emptyset, \{o_5\}, \{o_2\}, \{o_2\}, \{o_2\})$$

Subsequently, $\mathbf{C} = (0,0,0,2)$ that is $CSR = \frac{2}{14}$. Nevertheless, the optimal sequence of cache states is $(\emptyset, \{o_5\}, \{o_5\}, \{o_5\}, \{o_2\})$ or $(\emptyset, \{o_5\}, \{o_5\}, \{o_5\}, \{o_5\})$ that result in the same $\mathbf{C} = (0,0,5,0)$ and $CSR = \frac{5}{14}$. Consequently, this counterexample has shown that "$SMFD^*$ is optimal for nonuniform objects" is false.

Proposition 9: $SMFD^*$ *has the worst-case running time of* $O(mn\log_2 n + m^2 n)$.

Proof: For the 1^{st} cache replacement, each object in $V_t \cup \{o_t\}$ is searched in the remaining request sequence at most for $m-1$ requests to determine its shortest maximum forward distance in the sequence, and sorted for $n \log_2 n$ and evicted for n cached objects at worst. Likewise, for the 2^{nd} cache replacement, each object in V_t is searched in the remaining $m-2$ requests, adding a newly missing object, and at most n cached object eviction takes place, and so forth. Therefore, total time to resolve all of the requests is at most $((n+1)(m-1) + n \log_2 n + n \log_2 n) + ((n+1)(m-2) + \log_2 n + n\log_2 n) + ...+ ((n+1) + \log_2 n + n \log_2 n)$ equal to $(2n + (m-2)(n+1)) \log_2 n + \frac{(m-1)m(n+1)}{2}$. Hence, $SMFD^*$ is $O(mn\log_2 n + m^2 n)$.

IV. Conclusion

This paper presents one optimal and eight suboptimal offline cache replacement policies for nonuniform object costs. OPT^*, *SMFD*, and $SMFD^*$ are our distinct contributions. We have found that the optimality and respective time complexity of all investigated policies could be summarized below.

- Optimal offline cache replacement policy for nonuniform objects

- $Hosseini–Khayat's\ OPT^* = O(n^{3m})$

- Suboptimal offline cache replacement policies for nonuniform objects

  - $BL = O(mn^3 \log_2 n^3)$
  - $PE = O(mn^{3T})$
  - $C_0 = O(mn\log_2 n + m^2 n)$
  - $C_0^* = O(mn\log_2 n + m^2 n)$
  - $Mattson's\ OPT = O(mn\log_2 n + m^2 n)$
  - $OPT^* = O(mn\log_2 n + m^2 n)$
  - $SMFD = O(mn\log_2 n + m^2 n)$
  - $SMFD^* = O(mn\log_2 n + m^2 n)$

As n represents the number of objects in a cache, it is a pitfall to interpret the above complexities that the smaller cache size, the faster caching speed. Typically, a smaller cache size yields faster cache replacement but more frequent cache miss resolution. In other words, an overall caching speed is bound to both cache size and miss rate.

REFERENCES

[1] M. Brehob, S. Wagner, E. Torng, and R. Enbody, "Optimal replacement is np-hard for nonstandard caches," *IEEE Transactions on Computers*, vol. 53, no. 1, pp. 73–76, Jan 2004.

[2] S. Podlipnig and L. Bőszőrmenyi, "A survey of web cache replacement strategies," *ACM Comput. Surv.*, vol. 35, no. 4, pp. 374–398, Dec. 2003.

[3] A. Balamash and M. Krunz, "An overview of web caching replacement algorithms," *Communications Surveys Tutorials, IEEE*, vol. 6, no. 2, pp. 44–56, 2004.

[4] T. Banditwattanawong, M. Masdisornchote, and P. Uthayopas, "Multi-provider cloud computing network infrastructure optimization," *Future Generation Computer Systems*, vol. 55, pp. 116–128, 2016.

[5] J. Xu, Q. Hu, W.-C. Lee, and D. L. Lee, "Performance evaluation of an optimal cache replacement policy for wireless data dissemination," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 1, pp. 125–139, Jan 2004.

[6] HTTP Archive. (15 September 2017) Interesting stats. [Online]. Available: http://httparchive.org/interesting.php

[7] Amazon.com, Inc. (24 October 2017) Amazon Web Services. [Online]. Available: https://aws.amazon.com/s3/pricing/

[8] S. Hosseini-Khayat, "On optimal replacement of nonuniform cache objects," *IEEE Transactions on Computers*, vol. 49, no. 8, pp. 769–778, Aug 2000.

[9] O. Bahat and A. M. Makowski, "Optimal replacement policies for nonuniform cache objects with optional eviction," in *Proc. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428*, March 2003, pp. 427–437 vol. 1.

[10] D. Starobinski and D. Tse, "Probabilistic methods for web caching," *Perform. Eval.*, vol. 46, no. 2-3, pp. 125–137, Oct. 2001.

[11] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," *IBM Systems Journal*, vol. 9, no. 2, pp. 78–117, 1970.

[12] P. Michaud, "Some mathematical facts about optimal cache replacement," *ACM Trans. Archit. Code Optim.*, vol. 13, no. 4, pp. 50:1–50:19, Dec. 2016.

Thepparit Banditwattanawong received B.Eng. in computer engineering from King Mongkut's Institute of Technology Ladkrabang, Thailand and M.Eng. in computer science from Asian Institute of Technology, Thailand. He obtained Ph.D. in Informatics from The Graduate University for Advanced Studies, Japan.

He was a researcher at National Electronics and Computer Technology Center (NECTEC) and a specialist at Electronic Government Agency (EGA), Thailand. He joined the Department of Computer Science at Mahidol International College as a full time lecturer and later acted in a Ph.D.IT. program director position at Sripatum University. He is currently a full time lecturer in the Department of Computer Science and Ph.D.CS. program chairman at Kasetsart University, Bangkok, Thailand.

Assist. Prof. Dr. Thepparit received research grants from TRF and NRCT. His main areas of research interests include cloud computing and distributed computing. He professionally specializes in information security and data networking as the certificate holders of CISSP, Cloud Essential, CCNA, ITPE, etc.