Adaptive Query-Caching in Peer-to-Peer Systems[*]

Zuoning Yin, Hai Jin, Chao Zhang, Quan Yuan, Chucheng Zhao

Cluster and Grid Computing Lab
Huazhong University of Science and Technology, Wuhan, 430074, CHINA
znyin@hust.edu.cn

Abstract. Peer-to-Peer (P2P) architectures are very prevalent in today's Internet. Lots of P2P file sharing systems using Gnutella protocol emerge out and draw attractions of millions of people. The "flooding" search mechanism of Gnutella makes it easy to be deployed, but also spawns numerous messages which leads to serious scalability problems. However, the locality discovered in both user's share files and queries, enables us to use query-caching to shorten the search length and reduce the messages traffic. This paper makes an extensive study of query-caching in P2P systems and proposes an adaptive query-caching mechanism to manage the cached query reply messages according to the heterogeneity of the uptime of different peers. Along with several other techniques we proposed, our approach achieves a 30% reduction of average search length and a 61% reduction of query message traffic comparing with the previous query-caching mechanisms in the simulation, which indicates that our approach makes Gnutella more scalable.

1. Introduction

There are mainly two kinds of P2P systems: unstructured [1] and structured [2][3], characterized by the search mechanism and the organization of peers. Due to the simplicity of the flooding search and loosely coupled structure, the unstructured P2P systems are more widely deployed than the structured ones. However the flooding search also spawns numerous messages, making the system not scalable. Lots of efforts have been made to tackle this problem, such as to limit the broadness of the searching, such as random k-walker [4] and routing index [5], to use expanding ring [4] and iterative deepening [6] to shorten the depth of the searching, to adjust the topology in order to reduce the message traffic [7]. Besides, replication [8] is used to enable peer to find desirable resources more quickly in time and shorter in distance. To achieve this, query-caching [9][10] caches query reply messages in a passive way.

In query-caching, peers cache the query reply messages they received. When a peer receives a query request, it searches its local share files as well as the cache. According to previous researches [11] on Gnutella workloads, the locality is a distinct characteristic in both users' queries and share files, which enables a considerable amount of queries hit in the cache. Hence the total message traffic and average search length can be reduced.

[*] This paper is supported by National Science Foundation of China under grant 60433040.

However the current query-caching techniques [9][10] are rather simple and unfledged. The cache management policy is mainly controlled by a fix period of time. If a query reply message has stayed in the cache more than a threshold, the message is evicted from the cache. In Gnutella, the uptime of peers is heterogeneously distributed. Therefore, treating all cached query reply messages with the same deadline is too simple to match this heterogeneity, which leads to an ineffective exploitation of the contribution of query-caching. In this paper, we propose a new mechanism called Adaptive Eviction to take peers' heterogeneity of uptime into consideration and to evict a cached query reply message (a cached record) according to each peer's uptime. Besides, we propose some additional techniques, such as Exclude List and Cache Transfer. Exclude List is used to reduce the message duplication and Cache Transfer prolongs the use of valid cached records.

This paper is organized as follows. In section 2, we discuss all the issues related to the design of query-caching. In section 3, we perform the simulation to evaluate the performance. Finally, we conclude in section 4.

2. Design of Query-Caching

We made modifications on the client of Limewire [12] to turn it into a crawler of the Gnutella network. The crawling lasted for three weeks from April 5th to April 26th. After the crawling, we found some supportive evidences to query-caching:

1) *The uptime of a peer is growing longer.* Only 25% of total peers have a short uptime less than 30 minutes. Peers that have an uptime ranging from 1 hour to 8 hours account for 46% of total peers. The average uptime is 4.08 hours. This is very significant to the deployment of query-caching, because the longer the uptime of peers is, the better the overall performance of query-caching will be.

2) *During one full session, peers seldom delete their shared files.* More than 97% peers will not delete their shared files during one full session (from the join of a peer till its leave). Hence we can approximately regard that a peer will not delete its shared files during its current session. Therefore the life cycle of files can be ignored to simplify our design of the query-caching.

2.1 Design Considerations

Peers in Gnutella show a heterogeneous characteristic in uptime. Hence the design of query-caching should consider this heterogeneity. Previous techniques employ a fix time eviction policy. This strategy does reduce the average search length and the total message traffic, but due to its simple design, it has two primary drawbacks: 1) To those peers whose uptime is below the threshold, a cache hit gives false guidance, because the origin peer of the cached record may have already left the network. 2) To those peers whose uptime is beyond the fixed threshold. Eviction of such records limits their contribution in the future.

The previous approaches neglect the duplication of query reply messages generated by the cache. In Gnutella, a peer will not respond to the same query with duplicate reply. With cache support, the duplication may happen. Those duplicated messages increase the traffic on network, waste the processing capacity of peers and eventually dispel the benefit of query-caching.

A peer's cache will be inaccessible when the peer leaves the network. While the cached records may still be useful, how to make full use of these valuable records is also not addressed in previous approaches.

Our design tackles these problems, and Adaptive Eviction, Exclude List and Cache Transfer are proposed to deal with the above three issues respectively.

2.2 Adaptive Eviction

Adaptive Eviction is the core part of our design. The word "adaptive" means that the eviction is conducted according to each peer's uptime, rather than treating every peer uniformly. However, in the real environment, it is hard to attain the accurate uptime. So we try to predict the uptime of a peer's current session instead.

Every peer keeps track of the dynamic of its uptime. Then we get a sequence $Uptime_1$, $Uptime_2$, $Uptime_3$, ……, $Uptime_i$, representing its uptime in different days. We define transition Tr as a 2-tuple ($Uptime_i$, $Uptime_{i+1}$), where $Uptime_i$ and $Uptime_{i+1}$ are the uptime of two consecutive days (two consecutive sessions). If $Uptime_i > Uptime_{i+1}$, the transition is regarded as a decrease change. Otherwise the transition is regarded as an increase change. The value of a transition V_{Tr} is $\left| Uptime_{\mathrm{i}} - Uptime_{i+1} \right|$. A peer calculates the value of *Decrease Change Ratio* ($Ratio_{DC}$) and *Average Decrease Change Range* ($AvgRange_{DC}$) as follows:

$$Ratio_{DC} = \frac{\text{the number of decrease changes}}{\text{the number of all transitions}}$$

$$AvgRange_{DC} = \frac{\sum V_{Tr} \, (Tr \in \text{decrease changes})}{\text{the number of decrease changes}}$$

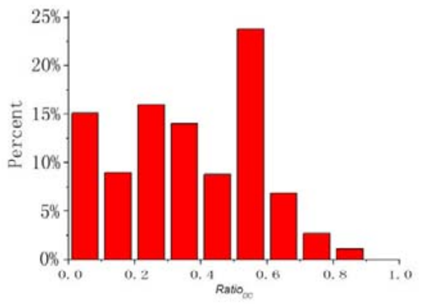
A peer's uptime of current session $Uptime_{current}$ is predicted based on the peer's uptime of last complete session $Uptime_{last}$, $AvgRange_{DC}$, and $Ratio_{DC}$ as follow:

$$Uptime_{current} = Uptime_{last} \times (1 - Ratio_{DC} \times AvgRange_{DC}) \qquad (1)$$

The $Uptime_{current}$ is always no more than $Uptime_{last}$ in Eq.1. But in many cases, the actual uptime of current session is longer than $Uptime_{last}$. Hence Eq.1 is a pessimistic prediction.

Based on our trace data, we study the transition of about 190,000 peers and find among all 13,467,602 transitions, over 65% are increase changes, 35% are decrease changes. The distribution of the $Ratio_{DC}$ is shown in Fig.1.

Fig.1 demonstrates that increase changes are more usual than decrease changes. Therefore, the $Uptime_{current}$ predicted in Eq.1 is a very conservative prediction and the probability that the $Uptime_{current}$ is longer than the accurate uptime can be low. Fig.2 shows the distribution of $AvgRange_{DC}$. There are over 30% peers with an $AvgRange_{DC}$ less than 0.3 and the average $AvgRange_{DC}$ is about 0.5.



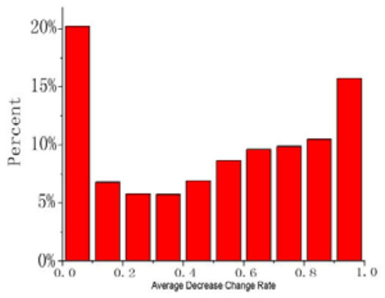


Fig.1. The Distribution of $Ratio_{DC}$

Fig.2. The Distribution of $AvgRange_{DC}$

Furthermore, we can predict the uptime more conservatively as follow:

$$Uptime_{current}' = Uptime_{last} \times (1 - AvgRange_{DC}) \quad (2)$$

We use the real trace data to evaluate our prediction. We predict the uptime of the last 3 days based on the statistics of the uptime of the first 18 days. By using Eq.1, there are 22.7% predictions (totally 37,034,503 predictions) to be false predictions (the predicted uptime is longer than the actual uptime). While using Eq.2, there are only 10.3% predictions to be false predictions. This result proves that our conservative prediction is feasible, especially by using Eq.2. By using Eq.2, we also find that the average of predicted uptime is promising. The average of accurate uptime is 6.56 hours, while the average of predicted uptime is 3.92 hours.

Adaptive Eviction is enforced as follow:

1) When a peer receives a query to be responded, it calculates the time (T_{valid}) for the validation of a query reply according to the predicted $Uptime_{current}$ and the current on-line time $Uptime_{now}$ as $T_{valid} = Uptime_{current} - Uptime_{now.}$ Then it adds T_{valid} to the query reply message.

2) When this query reply message is received, if T_{valid} is beyond a threshold h, the message will be cached. The use of threshold h is to avoid caching those messages with very small value of T_{valid}. Obviously, those query reply messages with a negative value of T_{valid} will not be cached, because their original peers are predicted to have left the network. The default threshold value here is set to half minute.

3) $Peer_j$ periodically examines the cached records. The T_{valid} of each record will be subtracted during every check operation. If T_{valid} is less then the threshold h, the corresponding record will be evicted.

2.3 Exclude List and Cache Transfer

Duplicating query reply messages caused by query-caching may increase the message traffic. So we add an exclude list to the query message. If a query is hit in a peer's cache, before the peer continues to flood the query to its neighbors, it adds the peers with corresponding records in the cache into the exclude list. Thus when the query is forwarded to next peer, the peers in the exclude list will be excluded from the searching scope. Although this approach could not solve the message duplication completely, it does provide an effective and light-weighted way to reduce a large fraction of duplication. Exclude List eliminates the message duplication in a depth-first way. To reduce duplication in a breadth-first way, random k-walker can be used together with Exclude List.

When a peer is about to leave the network, the information in its cache may still be valuable. So we can transfer the information of cache to some other peers that are still online in order to prolong their contribution. However we must take care of the choice of the destination of the transfer. We choose peers with a longer uptime. Because when the cache information is transferred to a peer that is ready to leave, we will face next transfer soon. Besides we should choose a peer with more connections. A peer with more connections usually has more queries through it. This enables the information to serve more queries. When most peers leave the network gracefully, that is, their leaves are under controlled and not caused by power failure, the Cache Transfer approach is a good supplement to query-caching.

2.4 Theoretical Analysis of Performance

We build a model to analyze the performance of query-caching. We use average search length to evaluate the performance. The average search length also influences the amount of message generated. We only consider a stable Gnutella network. We do not consider the initial stage of the establishment or the final stage of the demise of a Gnutella network. Due to the paper limitation, we only give the conclusions directly.

The average search length for fixed time eviction is:

$$L = 2 - \frac{T}{\Delta T} \tag{3}$$

The average search length for adaptive eviction is:

$$L = \frac{1}{(N+1)\Delta T}\int_{t_0}^{t_0+(N+1)\Delta T}\left[P_t\{L=1\}+2P_t\{L=2\}\right]dt$$

$$= \frac{1}{(N+1)\Delta T}\int_{0}^{(N+1)\Delta T}\left[P_t\{L=1\}+2P_t\{L=2\}\right]dt \tag{4}$$

ΔT is the interval between the entering to a peer cache of two consecutive query reply messages containing the same keyword. N is a value determined by $N\Delta T \le \max\limits_{P_{K1},P_{K2},\cdots,P_{K_m}} T_{\text{valid}}(K_j) < (N+1)\Delta T$, where P_{ki} stands for the peers whose shared files contain keyword K and T_{valid} is the time a record being cached. $P_t\{L=1\}$ stands for the probability that the search length is 1.

It can be observed that when the size of Gnutella network is big enough, $P_t\{L=1\}\uparrow\to 1$, $P_t\{L=2\}\uparrow\to 0$. While for Eq.3, $L_0\uparrow\to 2$.

Therefore, Adaptive Eviction mechanism achieves considerable improvement over the performance of fixed time eviction. The larger the Gnutella network is, the more obvious the improvement is. However, the upper bound of the improvement is:

$$(2-\frac{T}{\Delta T}-1)/(2-\frac{T}{\Delta T})\uparrow \xrightarrow{M\to\infty} 50\%$$

3. Simulations

In order to further evaluate our algorithm, we have designed a simulation. We set up a network with 600 peers. 100 peers are ultra-peers and the rest are leaf nodes. The ultra-peers are organized in a random graph. This setting is very similar to the real setting of Gnutella network, the only deference is that all the parameters are reduced by a ratio. Therefore, though a small peer size, our simulation can also represent the condition of a network with a larger size.

Other parameters are set as follow: the number of files of a peer is conformed to the *Zipf(1.5, 25)*; the number of queries a peer issued is conformed to *Zipf(2.0, 20)*; the class level of keywords is conformed to *Zipf(1.8, 40)*; the uptime of peers is conformed to *Zipf(1.2, 86400)*.

The simulation is set to last for a virtual time of 24 hours. During the simulation, when a peer leaves, we add a new peer at the position where the previous peer was. This provides a simple way to keep the network with a stable peer size, meanwhile reflecting the dynamic of the Gnutella work. When a query has received 50 query reply messages from 50 different peers, we think the query is satisfied and then stop flooding of the query. Otherwise we continue flooding the query until the TTL of the query becomes zero.

We have done six different experiments based on six different situations. They are: 1) Basic (without query-caching); 2) Fixed Time Eviction with a 5 minutes threshold; 3) Fixed Time Eviction with a 10 minutes threshold; 4) Adaptive Eviction; 5) Adaptive Eviction plus Exclude List; 6) Adaptive Eviction plus Exclude List plus Cache Transfer.

As shown in Fig.3, the average search length of the basic algorithm (without query-caching) is 3.114 hops. When adopting the fixed time eviction with a 5 minutes threshold, the average search length drops to 1.749. However, when the threshold is set to 10 minutes, the average search length decreases to 1.614, which is only a 7.7% improvement. So for fixed time eviction, the performance bonus decrease with the increasing of the threshold. After the threshold reaches a certain value, continuous increasing it will not improve the performance much. This is because that when increasing the threshold, though the cache can store more query reply messages, it suffers more from the penalties which is to compensate the invalid records in the cache. With Adaptive Eviction, we can achieve an average search length of 1.128, which is a 30.1% reduction from the fixed time eviction with a 10 minutes threshold. This result demonstrates that Adaptive Eviction has considerable advantages over fixed time eviction. When adopting Cache Transfer, the average search length can be further reduced, while the reduction is not obvious, only 3.8%.

With the same parameters, we use Eq.3 and Eq.4 to calculate the theoretical value of average search length. We get ΔT=1904.4 seconds. Therefore when adopting the fixed time eviction with a 5 minutes threshold:

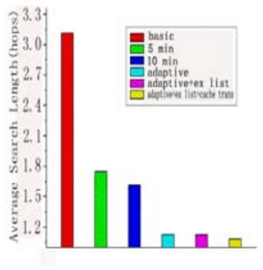


Fig.3. Average Search Length

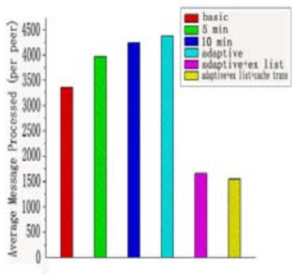


Fig.4. Average Messages Processed

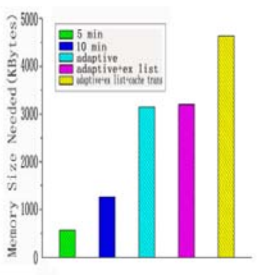


Fig.5. Average Memory Size

$$L_{5\min} = 2 - \frac{T}{\Delta T} = 2 - \frac{300}{1904.4} = 1.84$$

When the threshold is 10 minutes,

$$L_{10\min} = 2 - \frac{T}{\Delta T} = 2 - \frac{600}{1904.4} = 1.67$$

When adopting Adaptive Eviction, we get N as 4, and $L \approx 1.1172$. The improvement rate here is about 33.10%. The experimental data is very close to the theoretical value.

Fig.4 shows the messages processed per peer for different algorithms. The message here only includes query request and query reply messages. When adopting the basic algorithm, the average query messages processed per peer Avg_{QM} is 3352.2. When adopting the fixed time eviction, without using of Exclude List, Avg_{QM} increases to 3970.8 with a 5 minutes threshold. The Avg_{QM} continues increasing to 4245.5 with a 10 minutes threshold. However, when adopting the technique of Exclude List, the Avg_{QM} reduces to 1658.0. This is a 50.5% reduction from the basic algorithm and a 60.9% reduction from the fixed time eviction with a 10 minutes threshold, which indicate that Exclude List is very efficient in the reduction of message traffic.

Fig.5 shows the average memory a cache consumed. When adopting the fixed time eviction with a 5 minutes threshold, the average memory needed is about 575KB. With a 10 minutes threshold, the memory needed increases to 1265KB. When adopting Adaptive Eviction, the memory needed increases to 3146KB. Though Adaptive Eviction needs more memory for caching, a 3~4MB memory demand can be fulfilled easily for a state-of-art PC.

4. Conclusions and Future Works

Query-caching is an effective way to reduce the average search length and overall message traffics. However fixed time eviction mechanism has drawbacks and can not take full advantages of query-caching. By considering the heterogeneity of uptime, Adaptive Eviction is a more efficient approach to provide better performance. With the support of Exclude List and Cache Transfer, Adaptive Eviction is a good solution to query-caching.

In the future, we are looking forward to adopt our design in real environment in order to evaluate the practical effect to Gnutella. We are also trying to optimize our algorithm in order to make more accurate prediction of a peer's uptime, which will eventually give further improvement to the performance of system.

References

1. M. Ripeanu, "Peer-to-Peer Architecture Case Study: Gnutella", In *Proceedings of the International Conference on Peer-to-Peer Computing (P2P2001)*, Linkoping, Sweeden, Aug. 2001, pp.99-100
2. A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems", In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, Nov. 2001, pp.329–350
3. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications", In *Proceedings of ACM SIGCOMM 2001*, San Diego, CA, Aug. 2001
4. Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks", In *Proceedings of the 2002 International Conference on Supercomputing*, June 2002, NY, USA, pp.84-95
5. A. Crespo and H. Garcia-Molina, "Routing Indices for Peer-to-Peer Systems", In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, July 2002, Vienna, Austria, pp.23-34
6. B. Yang and H. Garcia-Molina, "Efficient Search in Peer-to-Peer Networks", In *Proceedings of the 22nd International Conference on Distributed Computing Systems(ICDCS'02)*, July 2002, Vienna, Austria, pp.5-14
7. Y. Chawathe, S. Ratnasamy, L. Breslau, and S. Shenker, "Making Gnutella-like P2P Systems Scalable", In *Proceedings of ACM SIGCOMM 2003*, Aug. 2003, pp.407–418
8. E. Cohen and S. Shenker, "Replication Strategies in Unstructured Peer-to-Peer Networks", In *Proceedings of the ACM SIGCOMM 2002*, Aug. 2002, PA, USA, pp.177-190
9. K. Sripanidkulchai, "The Popularity of Gnutella Queries and Its Implications on Scalability", http://www-2.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html
10. E. P. Markatos, "Tracing a Large-Scale Peer to Peer System: An Hour in the Life of Gnutella", In *Proceedings of 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2002)*, May 2002, Berlin, pp.65-74
11. K. P. Gummadi, R. J. Dunn, S. Saroiu, S. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, Modeling and Analysis of a Peer-to-Peer File-Sharing Workload", In *Proceedings of the 19th ACM symposium on Operating Systems Principles (SOSP03)*, pp.314-329
12. Limewire, http://www.limewire.org/