Peer-to-Peer Netw Appl
DOI 10.1007/s12083-009-0054-6

Popularity-aware prefetch in P2P range caching

Qiang Wang · Khuzaima Daudjee · M. Tamer Özsu

Received: 1 December 2008 / Accepted: 5 June 2009

Abstract Unstructured peer-to-peer infrastructure has been widely employed to support large-scale distributed applications. Many of these applications, such as location-based services and multimedia content distribution, require the support of range selection queries. Under the widely-adopted query shipping protocols, the cost of query processing is affected by the number of result copies or replicas in the system. Since range queries can return results that include poorly-replicated data items, the cost of these queries is usually dominated by the retrieval cost of these data items. In this work, we propose a popularity-aware prefetch-based approach that can effectively facilitate the caching of poorly-replicated data items that are potentially requested in subsequent range queries, resulting in substantial cost savings. We prove that the performance of retrieving poorly-replicated data items is guaranteed to improve under an increasing query load. Extensive experiments show that the overall range query processing cost decreases significantly under various query load settings.

Q. Wang (✉)
School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L3G1
e-mail: q6wang@uwaterloo.ca, qangw.2000@gmail.com

K. Daudjee · M. T. Özsu
University of Waterloo, Waterloo, Ontario, Canada

K. Daudjee
e-mail: kdaudjee@uwaterloo.ca

M. T. Özsu
e-mail: tozsu@uwaterloo.ca

Keywords Range caching · Prefetching · Unstructured P2P network

1 Introduction

Peer-to-Peer (P2P) infrastructure is being widely employed to support large-scale distributed applications. Besides well-established keyword-based file sharing and content distribution systems (*e.g.,* Gnutella[1] and BitTorrent[2]), many applications require complicated query types such as range selection queries[3] to be supported.

Range queries are used to retrieve all data that satisfy the specified *range constraints*. For instance, in on-demand P2P video systems (*e.g.,* Joost[4]), video clip data within a certain time frame are buffered and shared by some peers, which can be modeled as range data. Peers can pose a range query over any time frame to search the video clips that they intend to play, and the video data satisfying the range constraints are returned to the query issuer. Similarly, in a P2P location-based service system, users may request hotel information within a geographical area around a conference site, which can be modeled as a range query over the corresponding location. Range query processing may also be applied over single values as an advanced functionality of existing systems. For instance, in a music file sharing system,

[1]http://www.gnutella.com

[2]http://www.bittorrent.com/

[3]For conciseness, we use *range query* and *range selection query* interchangeably in the remainder of the paper.

[4]http://www.joost.com

song files are identified through title and release year (*e.g.,* {"*Pink Floyd*", "1982"}); a range query "search all Pink Floyd songs between year 1980 and 1990" will enhance the system with range-aware functionality.

Existing works on P2P range query processing typically rely on structured overlay network protocols (*e.g.,* BATON [11] and P-Tree [5]). While these protocols guarantee range query processing to complete within a bounded number of routing hops, they are not widely adopted in practical systems due to the following drawbacks: (1) the construction and maintenance, particularly during network churn, of structured overlay networks are non-trivial; and (2) these approaches often disregard the potential of data caching during query processing, which may affect query execution performance significantly.

In contrast, unstructured P2P overlay network architecture is widely adopted by practical systems, where constrained flooding mechanisms are usually employed for keyword-based searching (*e.g.,* in Gnutella and Bubblestorm [20]). Since data are cached at peers after retrieval, subsequent queries for the same data can be answered by multiple caches, facilitating the search process. In the case of range query processing, the queries can also be shipped within the network through flooding. Data retrieved after the query execution are easily cached at query issuers, which produce *distributed range caches* that can potentially be used during subsequent range query processing.

Since unstructured P2P overlay networks are built without the knowledge of data placement, average communication cost (*e.g.,* the number of messages or latency) per query is inversely proportional to the number of data copies, or replicas, in the network that satisfy the query [4]. Those data results that are not well-replicated may incur a higher cost to retrieve, delaying the progress of the range query processing. For simplicity, all peers are assumed to be sufficiently powerful to keep data replicas and process related range queries. In practice, peers with heterogeneous storage and processing capacity may act as super-peers. Such a general heterogeneous model is not studied in this dissertation. However, in the proposed approach, peers may impose different constraints on cache storage capacity, which will affect the computation of the data correlation parameter τ, as detailed in Section 3.1.

It is often the case that range query results include data items that are not well replicated. For example, in a location-based hotel reservation system in P2P networks, "popular" hotels, such as those close to a conference site, are usually queried first by conference participants. When these hotels are booked in full, users tend to relax the range constraints (*e.g.,* with respect to geographical proximity) to explore other hotel information, which may not be well replicated in the network yet. For simplicity, we refer to these data items that are not well replicated as *poorly replicated* data items. The approach proposed in this work will predict and prefetch those poorly-replicated data items that may potentially be requested in subsequent range queries and then facilitate the caching of these data to improve overall query execution performance.

Existing approaches are not effective in facilitating the caching of the poorly-replicated data items that may potentially be queried in the future. In unstructured P2P overlay networks, *uniform replication scheme* (deployed in KaZaa) caches each data item at a fixed number of peers so that it preventively excludes poorly-replicated data items from the system. However, this scheme is oblivious to the knowledge of query distribution such that the fixed number of peers that manage the data items covered by "popular" queries may be overloaded. Moreover, the replication scheme requires strong altruistic cooperation from peers in that peers may cache data items regardless of whether they are issuing queries, which may not be feasible in uncooperative P2P environments. In contrast, with the *proportional replication scheme* that is employed in Gnutella,[5] each peer only caches results after query execution such that caching process is triggered by query processing and only query issuers rather than arbitrary peers cache the results. Similarly, *square-root replication scheme* [4] makes the number of cached data items proportional to the square root of the number of the corresponding queries, improving average query processing performance with respect to constrained cache sizes. In comparison to the uniform replication scheme, the last two schemes achieve better load-balancing by considering query distributions and do not rely on altruism from peers. However, they disregard the caching of poorly-replicated data items, which may bottleneck the performance of the range query since these data are part of the range query result.

The key to the problem is to recognize those poorly-replicated data items that may potentially be requested by range queries in the future, and to facilitate the caching of these data items. With respect to unstructured P2P overlay networks, the following design principles are crucial: (1) the approach needs to be purely decentralized; (2) the approach should be efficient, without incurring significant communication or computational overhead; and (3) it is desirable that the approach can be deployed with existing routing protocols

[5]http://www.gnutella.com

and replication schemes that have been developed for unstructured P2P networks.

Using the intuition that well-replicated data are “popular” data cached in the P2P system, we propose an efficient, decentralized *popularity-aware prefetch-based* approach to facilitate the caching of poorly-replicated data items. Prefetching in the context of other application domains (*e.g.,* compiler technology) has been well-studied. Our approach, in the context of prefetching range data in P2P systems, is novel in that it is data-popularity aware: (1) peers independently collect global information about the relationship between poorly-replicated data items and “popular”, well-replicated, ones involved in previously executed queries, enabling an adaptive approach for range query processing (see Section 3 for details); (2) since popular data are easily obtained from peers through flooding or random walk mechanism, simply prefetching poorly-replicated data items can be more cost-effective with respect to bandwidth consumption; and (3) sufficient query issuers will exist over “popular” data items, providing opportunities to piggyback “correlated” poorly-replicated data items onto popular data and thereby facilitating the prefetching process. Briefly, the contributions of this work include the following.

- This is the first work that addresses distributed range caching problems in unstructured P2P overlay networks. In particular, purely decentralized mechanisms are developed to locate poorly-replicated, “correlated”,[6] data items that are potentially queried in the future, and an adaptive prefetch-based approach is proposed to improve performance of the range query processing that involves those poorly-replicated data items.
- The effectiveness of the approach is demonstrated theoretically by proving that, under a specific query distribution model with an increasing query load, the approach can improve overall performance of the queries that retrieve poorly-replicated data items by at least a factor of $\mathcal{O}(\ln m)$, where m is the number of queries, even when network churn and cache expiration exist.
- Through extensive simulations, the effectiveness of the approach is demonstrated under various query load settings.

The organization of the remainder of the paper is as follows. In Section 2 we analyze performance of cache-based range query processing in unstructured P2P overlay networks. Section 3 covers the prefetch-based approach that results in substantial communication cost savings for range query processing. Performance evaluation results are presented in Section 4. We review the related work in Section 5 and conclude this paper in Section 6.

[6]We define “correlation” in Section 3.1.

2 Cache-based range query processing

Range queries typically involve the range constraints that are defined over numeric-valued data [15]. The processing of a specific range query completes when all distinct result data items (either from original data sources or from caches) that satisfy the range constraints have been retrieved. Range queries can be issued by any peer in the system, and be shipped to other peers through flooding, gossip, or random walk routing mechanisms. Since flooding is no more effective than random walk with respect to routing cost [4], while gossip mechanism is based on random walk in that queries are shipped to randomly chosen peers in the network, we focus on the random walk mechanism in the remainder of this paper without loss of generality.

Based on the random walk mechanism, both the number of messages and the latency to retrieve a specific data item are inversely proportional to the number of the data item replicas in the network. Suppose that the cost function (denoted by *Cost*) of range query processing is defined over $Q \times R$, where Q denotes the set of range queries and R is the real value domain of query processing cost.[7] Given a query $q \in Q$, $Cost(q) \propto \frac{1}{r_q}$ (*i.e., Cost*(q) is proportional to $\frac{1}{r_q}$), where r_q is the lower-bound of the number of data item replicas that can satisfy q. When there exist multiple distinct data items $\{s_1, s_2, ...\}$ in the network that satisfy range query q, $r_q = min(|s_1|, |s_2|, ..., |s_i|, ...)$, where we denote by $|s_i|$ the corresponding numbers of data item s_i replicas.

We are especially concerned about the range queries that include poorly-replicated data items because the overall range query performance is affected by them. We refer to the period that specific data items have not been sufficiently replicated as the *cold period*. Suppose that the results of a range query q include data item s, which initially has a single replica (*i.e.,* the original data item itself) in the network; when there are m subsequent q queries issued, the overall query execution cost during s’ cold period is computed as below, based on the well-established proportional and square-root replication schemes respectively. For simplicity,

[7]In this work, we focus on the query shipping cost to locate query results, ignoring local processing cost.

we suppose that each execution of the query is initiated by a distinct peer.

- *Proportional replication scheme* Each query execution is expected to increase the number of replicas by one, such that the overall query processing cost with respect to m q queries, denoted by $Cost(q)$, is derived as below. Because sequence (3) does not converge, we present an approximate result with respect to a considerably large m.

$$Cost(q) = \sum_{i=1}^{m} \left(\frac{N}{i}\right) \tag{1}$$

$$= N + \frac{N}{2} + \frac{N}{3} + ... + \frac{N}{m} \tag{2}$$

$$\approx N \times \ln m \tag{3}$$

- *Square-root replication scheme* Although the square-root replication scheme performs better than the proportional replication scheme with respect to range query processing performance under constrained overall cache sizes [4], the range query processing involving poorly-replicated data items may incur higher communication cost. The following derivation presents the overall query processing cost, where sequence (5) increases monotonically and never converges. For simplicity, we assume that the number of replicas exactly equals the square root of the corresponding number of queries,[8] which does not affect the validity of the obtained result.

$$Cost(q) = \sum_{i=1}^{m} \left(\frac{N}{\sqrt{i}}\right) \tag{4}$$

$$= N + \frac{N}{\sqrt{2}} + \frac{N}{\sqrt{3}} + ... + \frac{N}{\sqrt{m}} \tag{5}$$

The above analysis shows that range query processing performance is affected when queries retrieve poorly-replicated data items during their cold period. In the next section, we propose the prefetch-based approach, where poorly-replicated data items are prefetched by query issuers that request the well-replicated data items "correlated" to the poorly-replicated ones. This potentially decreases the retrieval cost of the poorly-replicated data items within cold periods and improves the range query execution performance.

3 Prefetch-based caching

In this section, we first define data *correlation*, which materializes the locality concept that is essential to prefetch-based mechanisms [19]. Then we detail the design of the prefetch-based approach.

3.1 Data correlation

To quantify the correlation between poorly-replicated data items and well-replicated ones, we introduce a distance function D. For Euclidean range space, when data items represent point data (*e.g.,* the longitude and latitude information of locations), D can simply be the one-dimensional Euclidean distance function between the point values.[9] Instead, when data items correspond to range segments (*e.g.,* the range span of longitude and latitude information of a region around a specific location), D may be defined over the Euclidean distance between the centroid points (*e.g.,* median points in one-dimensional space) of the corresponding range segments. Other applications may employ their customized distance functions, which does not affect the applicability of the prefetch-based approach.

Based on the distance function D, data *correlation* is defined as follows: *two data items s and s' are correlated if $D(s, s') \leq \tau$*. The correlation threshold τ can be predefined and configured by peers when they join the overlay network, which may not be sufficiently flexible since the threshold may be over or under-valued. For example, with respect to a specific range query load, when τ is set too low, data items that are covered by the same range queries may potentially be regarded uncorrelated; in contrast, when τ is set too high, more irrelevant data items may be regarded correlated even if they are never queried together. Since we are especially interested in the piggybacking of poorly-replicated data items with the well-replicated query results, τ is measured based on the distances between poorly-replicated data items and well-replicated ones from the history of executed range queries.

On one hand, this approach takes the information of range queries into account such that: (1) the inherent correlation of data items within the same range query is captured; and (2) since intuitively the well-replicated data items are "popular" among peers, their correlated (poorly-replicated) data items may also become "popular" with a higher probability, which is recognized by our approach. On the other hand, the approach

[8]The actual number of replicas equals the square root of the corresponding query load size multiplied by a constant factor [4].

[9]This does not conflict with the focus on range query processing since range queries may include multiple point values.

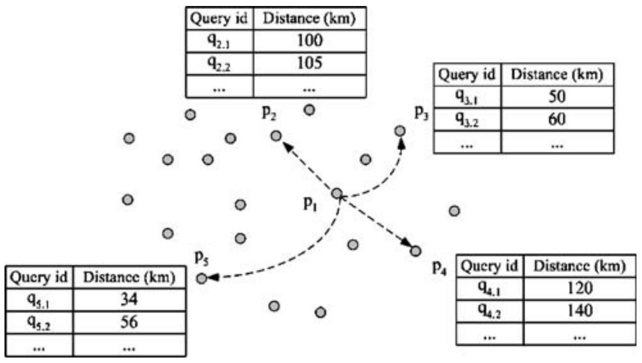
is popularity-aware, enabling *adaptive* data prefetching: the greater the portion of range query workload that retrieves poorly-replicated data items, the more precisely τ captures the expected distances between poorly-replicated data items and well-replicated ones. Conversely, when range queries seldom retrieve poorly-replicated data items, τ tends to be close to zero such that prefetching may not even be triggered.

While there do exist correlations between poorly-replicated data items, or between well-replicated data items, these are not considered in this work because: (1) the proposed approach relies on query issuers requesting well-replicated data items to prefetch poorly-replicated ones; the correlation between poorly-replicated data items is less important because it does not indicate the involved data items will be requested in subsequent queries; and (2) the retrieval of well-replicated data items incurs low processing cost, such that the prefetching of these data items may not be cost-effective with respect to bandwidth consumption. Performance evaluation (Section 4) supports the belief that simply prefetching all correlated data items regardless of data popularity may not be efficient.

Specifically, each peer records the history of the range queries that it issued previously. Each history record contains the maximum distance between any poorly-replicated data items and well-replicated ones that are involved in the range query at query execution time. Each peer then randomly samples a configurable number of other peers in the overlay network and obtains their query processing history records. This captures approximate global information about how poorly-replicated data items affect range query execution. The random sampling in unstructured P2P overlay networks can be realized through existing techniques [7, 12]. Query processing history records are collected by peers periodically such that they learn up-to-date knowledge on executed range queries.

For example, as shown in Fig. 1, peer p_1 randomly samples a number of peers (*i.e.,* p_2, p_3, p_4 and p_5) from the network and obtain their history records.

In addition to collecting history records by sampling the network, peers can alternatively estimate such information locally without consuming extra communication costs. Given a peer p, it observes the query results that are being routed via it. Since the query results that are piggybacked with query are be a subset of the complete results, the history records extracted out of the results may not be accurate. Intuitively, when we can estimate how complete the partial query results are, we can estimate the confidence of the extracted history records. This is feasible for range queries such as in requesting all video fragment within a specific period



Query id	Distance (km)
$q_{2.1}$	100
$q_{2.2}$	105
...	...

Query id	Distance (km)
$q_{3.1}$	50
$q_{3.2}$	60
...	...

Query id	Distance (km)
$q_{4.1}$	120
$q_{4.2}$	140
...	...

Query id	Distance (km)
$q_{5.1}$	34
$q_{5.2}$	56
...	...

Fig. 1 Sampling-based estimation of τ

of time. Since usually the time period for each video fragment is identical and predefined in advance, it is easy to compute how many video fragments to be contained in final query results. Thus we can easily estimate how complete the partial results are. In more general range query processing cases, without mechanisms to estimate the completeness of partial query results, it is harder to achieve accurate history records. The idea of using data being routed through is similar to that employed in [24], which focused on measuring data popularity instead of collecting history information about query processing.

Once a set of distance values from the history records, denoted by $\{d_i\}$, are obtained, each peer p estimates the distribution of the distance values through kernel estimation technique [18], which is a non-parametric data distribution modeling scheme. Non-parametric modeling schemes are advantageous in P2P environments since no a-priori knowledge about data distribution is required. Then we set τ to be the expected distance value under the distribution model. In this work, the commonly-used Gaussian kernel function $K(x) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}x^2}$ is employed, and the estimated probability density function (PDF) is $PDF_k(x) = \frac{1}{Sh}\sum_{i=1}^{S} K\left(\frac{x-x_i}{h}\right)$, where S denotes the number of history records, x_i denotes the distance value corresponding to the i_{th} sample, and h is a smoothing factor that is configurable with respect to specific applications. The derivation of τ is shown below, where $y_i = \frac{x-x_i}{h}$ for each specific i.

$$\tau = \int_0^{\infty} x \times PDF_k(x)dx = \int_0^{\infty} x \times \frac{1}{Sh}\sum_{i=1}^{S} K\left(\frac{x-x_i}{h}\right) dx$$
$$= \frac{1}{Sh\sqrt{2\pi}} \times \sum_{i=1}^{S}\left(\int_0^{\infty} x \times e^{-\frac{1}{2}\left(\frac{x-x_i}{h}\right)^2} dx\right)$$
$$= \frac{1}{Sh\sqrt{2\pi}} \times \sum_{i=1}^{S}\left(\int_0^{\infty} x \times e^{-\frac{1}{2}y_i^2} dx\right)$$

Define $y_i = x - x_i$,

$$\begin{aligned}
\tau &= \frac{1}{Sh\sqrt{2\pi}} \times \sum\nolimits_{i=1}^{S} \left(\int_0^\infty x \times e^{-\frac{1}{2}y_i^2} \times h dy_i \right) \\
&= \frac{1}{S\sqrt{2\pi}} \times \sum\nolimits_{i=1}^{S} \left(\int_0^\infty (hy_i + x_i) \times e^{-\frac{1}{2}y_i^2} dy_i \right) \\
&= \frac{1}{S\sqrt{2\pi}} \times \sum\nolimits_{i=1}^{S} \left(\int_0^\infty hy_i \times e^{-\frac{1}{2}y_i^2} dy_i \right. \\
&\qquad \left. + \int_0^\infty x_i \times e^{-\frac{1}{2}y_i^2} dy_i \right) \\
&= \frac{1}{S\sqrt{2\pi}} \times \sum\nolimits_{i=1}^{S} \left(h \times \int_0^\infty y_i \times e^{-\frac{1}{2}y_i^2} dy_i + x_i \right. \\
&\qquad \left. \times \int_0^\infty e^{-\frac{1}{2}y_i^2} dy_i \right) \\
&= \frac{1}{S\sqrt{2\pi}} \times \sum\nolimits_{i=1}^{S} \left(h + x_i \times \frac{\sqrt{2\pi}}{2} \right) \\
&= \frac{h}{\sqrt{2\pi}} + \frac{1}{2S} \times \sum\nolimits_{i=1}^{S} x_i
\end{aligned}$$

Observe that the computed expected value of τ is only slightly different from the average $\left(i.e., \frac{\sum_{i=1}^{S} x_i}{S}\right)$ of the sampled distant values. An engineering approach is to replace the expected τ computed through kernel estimation with the average to ease local computation costs. However, this may decrease the accuracy of the data correlation computation, potentially increasing the volume of prefetched data items that are irrelevant to subsequent queries.

With respect to range query processing, since query issuers and peers holding query results may obtain different values of τ, we always choose the query issuer's τ as the correlation threshold, because they can flexibly adjust the value of τ to include other constraints (*e.g.,* local storage constraint). For example, when the query issuer has a storage constraint, denoted by sc, it always sets $\tau = min(correlation\ threshold, sc)$ such that the local storage constraint is never violated.

3.2 Data popularity

The computation of τ requires the knowledge of data *popularity*, which is used to distinguish poorly-replicated data items from well-replicated ones. Although the obtained query history records can be employed to estimate data popularity [24], they may not be sufficiently accurate to reflect the overall data distribution in the overlay network. Thus, we consider a purely decentralized approach to estimate data popularity directly. Each peer evaluates the popularity of the local data through an exploration process. Consider a peer p with a set of data items, denoted by $\mathcal{S} = \{s_1, s_2, ...\}$. p issues an "exploration query" over each $s_i \in \mathcal{S}$, where a configurable Time-to-Live (TTL) counter is attached to each exploration query. During each hop, the TTL counter decreases by one and all appearances of s_i are recorded by p. The shipping of an exploration query terminates when TTL equals zero. Once the exploration process completes for all data items, p figures out the number of cached replicas for each s_i during the exploration process. Those s_i with more replicas (*i.e.,* above threshold T) are regarded as "well-replicated" and the others are considered "poorly-replicated". A similar approach has been employed in PIER to decide data popularity [9]. Both the TTL and T are configurable, where the TTL value is usually set to be small to avoid large explorations.

3.3 Prefetch-based approach

Suppose that peer p receives a range query issued by p' that covers a data item s; p will reply to p' with all its cached data items $\{s_1, s_2, ..., s_i, ...\}$ that satisfy the following conditions: (1) each s_i is correlated to s based on the distance function D and the correlation threshold τ that is attached to the query (*i.e.,* $D(s, s_i) \leq \tau$); and (2) based on the data popularity measurement of p' (*i.e.,* the query issuer), s is well-replicated and s_i is poorly-replicated.

For instance, consider multiple query issuers requesting data items s and s' through queries q_1 and q_2 respectively, as illustrated by white and grey nodes in Fig. 2a. For simplicity, suppose that s and s' are cached at peer p that receives the queries. Then Fig. 2b demonstrates the proportional replication scheme and Fig. 2c shows the proportional replication scheme enhanced with prefetching, where peers denoted by dark nodes eventually cache the poorly-replicated data item s'. Due to the prefetching of s' by query issuers over q_1, s' is cached more quickly in Fig. 2c.

We describe the prefetch-based approach in Algorithm 1 regarding peer p when it receives a query q covering data item s that is issued at p'. The approach

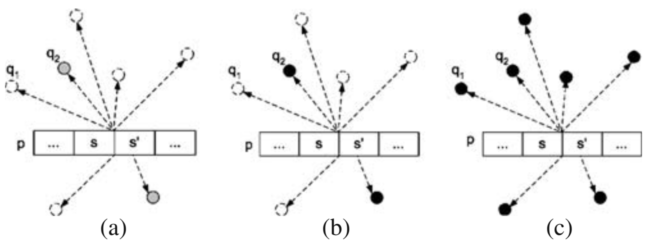


Fig. 2 Caching approaches (**a**–**c**)

has several advantages: (1) it makes slight changes to existing replication schemes without adjusting the overlay network architecture or routing algorithms; all decisions, including the computation of the data correlation threshold and data popularity, are made independently without costly coordination among peers; (2) poorly-replicated data items are piggybacked during the query processing over well-replicated data items, saving communication cost; moreover, peers are not required to be altruistic since only query issuers who utilize the query processing services take on the prefetching overhead cost; and (3) the approach is purely decentralized without relying on any centralized mechanisms, thus providing scalability.

Algorithm 1 *prefetch_based_caching*(q, τ)

```
small
1: if $p$ learns that $s$ is well cached then
2:    for all data items $s'$ that are poorly-replicated and $D(s, s') \leq \tau$ do
3:       if $D(s, s') \leq \tau$ then
4:          $p$ replies to $p'$ the data item $s$;
5:       end if
6:    end for
7: end if
```

3.4 Guarantees on performance improvement

With the prefetch-based approach, the population of poorly-replicated data items is affected by the query volume over the correlated well-replicated data items. We now show that, under the following query distribution, our approach guarantees that the overall range query cost over poorly-replicated data items is $N \times \mathcal{O}(1)$, which is at least $\mathcal{O}(\ln m)$ factor less than the counterpart when no prefetching is enforced.

Consider a range query q_1 involving data item s and query q_2 involving data item s', where s is well-replicated while s' is poorly-replicated. Suppose that, initially only one replica of s' exists in the network, while there exist n replicas of s including original and cached ones. Consider a query load consisting of m_1 q_1 queries and m_2 q_2 queries. For simplicity, all q_1 (and q_2) queries are uniform-randomly distributed across a certain period of time; thus it is expected that $\frac{m_1}{m_2}$ q_1 queries are processed when each q_2 is processed. Note that this assumption is not necessary for the following analysis; it will be clear shortly that only if the number of q_1 queries is sufficiently large compared with that of q_2 queries, the analysis will hold. We also assume that the number (n) of s replicas is stable, and will discuss the case when n may change. We now prove Theorem 1, where $\delta > 1$ is a system parameter that affects the constant factor of the average query operation cost (*i.e.,* $\mathcal{O}(1)$), and N denotes the network size. A proof is presented subsequently.

Theorem 1 *When* $\frac{m_1}{m_2 \times n} \geq 2^\delta$, *the overall cost for processing* m_2 q_2 *queries equals* $\mathcal{O}(1) \times N$.

Proof Suppose that initially, there is one replica of s' and it is cached together with s on a peer. This does not affect the generality of the analysis since subsequently prefetched s' will be cached together with s. During each period, one q_2 is processed and $\frac{m_1}{m_2}$ q_1 queries are issued such that the expected number of q_1 queries that visit a peer holding s' equals $\frac{m_1}{m_2 \times n}$. When $\frac{m_1}{m_2 \times n} \geq 2^\delta$, the number of cached s' data item copies increases by 2^δ fold. Recursively, suppose $i > 1$, when there are $(i-1)^\delta$ s' replicas in the network after the execution of the $(i-1)_{st}$ q_2 query, the expected number of peers that cache s' through the execution of query q_1 equals $\frac{m_1}{m_2 \times n} \times (i-1)^\delta$ and is no less than i^δ, as shown in the following derivation.

$$\frac{m_1}{m_2 \times n} \times (i-1)^\delta > (2 \times (i-1))^\delta \tag{6}$$

$$\geq \left(\left(1 + \frac{1}{i-1}\right) \times (i-1)\right)^\delta = i^\delta \tag{7}$$

Consequently, when the execution of the i_{th} q_2 query completes, no less than i^δ peers will cache s'. The overall shipping cost to resolve all q_2 queries is then computed as follows. While q_2 queries are executed sequentially, the above analysis is easily extended to handle concurrent query execution.

$$Cost(q_2) \leq \sum_{i=1}^{m_2} \frac{N}{i^\delta} \tag{8}$$

$$= N + \frac{N}{2^\delta} + \frac{N}{3^\delta} + \ldots + \frac{N}{m_2^\delta} \tag{9}$$

Since sequence (9) is a Riemann-Zeta sequence [6], it converges for all real values $\delta > 1$. Thus the overall query operation performance with respect to the query load of q_2 is $Cost(q_2) = \mathcal{O}(1) \times N$ when m_2 is considerably large. □

For example, when $\delta = 1.5$, $Cost(q_2) \approx 2.612 \times N$. This overall cost is at least $\mathcal{O}(\ln m)$ times less than those achieved by existing replication schemes (addressed in Section 2) that do not employ prefetching.

A sufficient condition of Theorem 1 is that the ratio of the number of q_1 queries over the number of s caches (denoted by n) is no less than 2^δ. However, multiple factors may affect this number in practice: (1) after the execution of q_1 queries, peers are capable of caching s, increasing n; (2) under network churn, peers may fail (or leave) suddenly, decreasing n; and (3) when cache expiry schemes are deployed in P2P systems for data freshness [1, 10], n may also change.

Cache replacement may affect the value of n as well. Specifically, if the cache size is sufficiently large, peers can hold all recently cached data items and cache replacement would happen infrequently. If the cache size is limited, cache replacement behavior can be easily integrated with cache expiry. For example, a widely employed recency-based cache replacement strategy such as Least-Recently-Used can be enforced through the cache expiry process by choosing an appropriate expiry period. This would evict least-recently-used data items from caches after each expiry period.

Next, given a specific query load of q_1, Theorem 1 can be satisfied even when network churn and/or cache expiry occur. We prove this in Theorem 2. Recall that, during each period, one q_2 query is issued. Any peer (and cache) fails (or leaves) with a probability of $0 \leq r \leq 1$ per period, and all cached data items expire after $k > 0$ periods of time. $l(i)$ represents the expected number of q_1 queries during the i_{th} period, $n(i)$ denotes the number of s data item replicas after the i_{th} period, and $n(0) = n$ denotes the initial number of s replicas.

Theorem 2 *When the query load of q_1 satisfies the following distribution,*

$$l(i) = \begin{cases} 2^{\delta+1}(1+2^{\delta+1})^{i-1}(1-r)^{i-1}n, & when\ i < k \\ 2^{\delta+1}(1+2^{\delta+1})^{i-1}(1-r)^{i-1}n & \\ \quad - \mathcal{O}((1+2^{\delta+1})^{i-k}(1-r)^i)), & \\ when\ i \geq k & \end{cases}$$

the number of s' replicas will be no less than i^δ after the i_{th} period, where $\delta > 1$; consequently, the overall cost of processing m_2 q_2 queries is bounded by $\mathcal{O}(1) \times N$, even under network churn and cache expiry.

Proof For brevity, we only prove the case when $i \geq k$. It is easy to apply the same derivation for the case when $i < k$, where no cached data expires.

When $i \geq k$, the number of s replicas (denoted by $n(i)$) after the i_{th} period is computed below, where the first component consists of the accumulated number of s until the $(i-1)_{st}$ period, added by the increase of the s (denoted by $l(i)$) during this period; the second component covers the loss of s replicas due to cache expiry. Both components are multiplied by corresponding damping factors to reflect network churn.

$$\begin{cases} n(i) = (n(i-1) + l(i)) \times (1-r), \quad when\ i < k \\ n(i) = (n(i-1) + l(i)) \times (1-r) - l(i-k) \\ \qquad \times (1-r)^k, \quad when\ i \geq k \end{cases}$$

Consider a more relaxed function $n'(i) = (n'(i-1) + l(i)) \times (1-r) - (1-r)^k$, where $n'(0) = n(0) = n$. Since $(1-r)^k \leq l(i-k) \times (1-r)^k$, it is obvious that $n'(i) \geq n(i)$. Then similarly consider,

$$\begin{cases} n'(i) = (n'(i-1) + l(i)) \times (1-r), \quad when\ i < k \\ n'(i) = (n'(i-1)+l(i)) \times (1-r) - (1-r)^k, \quad when\ i \geq k \end{cases}$$

Suppose $A = 2^{\delta+1}$ and make $l(i) = A \times n'(i-1)$. The following derivation computes $n'(i-1)$.

$$\begin{aligned} n'(i-1) &= (n'(i-2) + l(i-1)) \times (1-r) - (1-r)^k \\ &= (1+A)^{i-1} \times n'(0) \times (1-r)^{i-1} \\ &\quad - \sum_{x=1}^{i-k-1} \left((1+A)^x (1-r)^{x+k}\right) \\ &= (1+A)^{i-1} \times n \times (1-r)^{i-1} \\ &\quad - \sum_{x=1}^{i-k-1} \left((1+A)^x (1-r)^{x+k}\right) \\ &= [(1+A)(1-r)]^{i-1} \times n - \frac{(1+A)^{i-k}(1-r)^i}{(1+A)(1-r)-1} \\ &\quad + \frac{(1-r)^k}{(1+A)(1-r)-1} \end{aligned}$$

With reasonable network churn rate, $(1+A)(1-r)$ will be larger than 1. By ignoring the last component $\frac{(1-r)^k}{(1+A)(1-r)-1}$, which is a small constant when k is fixed, $n'(i-1)$ is bounded by $[(1+A)(1-r)]^{i-1} \times n - \mathcal{O}((1+A)^{i-k})(1-r)^i)$. Consequently, $l(i) = A \times n'(i-1) = 2^{\delta+1} \times ([(1+A)(1-r)]^{i-1} \times n - \mathcal{O}((1+A)^{i-k})(1-r)^i)$ is a sufficient condition for $l(i) \geq A \times n(i-1)$ because $n'(i-1) > n(i-1)$. Then the following analysis about the query processing cost holds based on mathematic recursion, where $i > 1$.

$$\begin{aligned} l(i) \times (i-1)^\delta &\geq 2^{\delta+1} \times (i-1)^\delta \times n(i-1) && (10) \\ &\geq 2 \times \left(\frac{i}{i-1}\right)^\delta \times (i-1)^\delta \times n(i-1) && (11) \\ &\geq (i^\delta + (i-k)^\delta) \times n(i-1) && (12) \\ &\rightarrow \frac{l(i) \times (i-1)^\delta \times (1-r)}{n(i-1) \times (1-r)} \\ &\quad -(i-k)^\delta \geq i^\delta && (13) \end{aligned}$$

Equation 13 shows that under the setting that the network churn rate equals r and the cached data items are evicted after k periods, the number of s' replicas (including the original and cached ones) is no less than i^{δ} after the i_{th} period. It is direct that, the overall cost for processing m_2 q_2 queries is no more than $\sum_{i=1}^{m_2} \frac{N}{i^{\delta}} = N + \frac{N}{2^{\delta}} + \frac{N}{3^{\delta}} + ... + \frac{N}{m_2^{\delta}} = \mathcal{O}(1) \times N$, even under network churn and cache expiry. □

3.5 Proactive preprocessing

The prefetch-based approach discussed above assumes that a cache contains both rare and popular data, which makes it direct for the rare data to be prefetched during the execution of queries that request the popular data on the same peer. However, the rare data that are correlated with popular data may initially be cached alone at some peers, such that peers making earlier queries involving rare data may experience a much longer delay than other peers that issue the queries later.

To alleviate this problem, we employ a light-weight proactive preprocessing. Suppose that, initially peer p caches only rare data item s_i; it computes all correlated data items $\mathcal{S} = \{s, s', ..., s_j, ...\}$ based on the distance function D and the data correlation threshold τ; then p issues an independent query over each s_j, and caches those popular data items s_j together with s_i. Consequently, both rare and the correlated popular data items are cached at the same peer p, such that Algorithm 1 can be applied.

The proactive preprocessing is easy to realize since it relies on existing query shipping protocols; moreover, the operation to retrieve popular data items is inherently inexpensive since these items have already been well replicated in the network. For clarity, the proactive preprocessing is sketched in Algorithm 2.

Algorithm 2 *proactive_preprocessing*(q)

```
1: for each peer $p$ that caches only rare range data $s_i$ do
2:     $p$ computes the set of correlated data items $\mathcal{S} = \{s, s', ..., s_j, ...\}$ based on $D$ and $\tau$;
3:     $p$ issues a query over each $s_j \in S$ and caches those well-replicated ones;
4: end for
```

3.6 Utilization of prefetched data

The volume of prefetched data is decided by threshold τ and the preference of peers (*e.g.,* regarding local storage constraints). This is based on the assumption that queries are coherent regarding the ranges (*e.g.,* over temporal and spatial ranges). For example, in the P2P location-based service system introduced in Section 1, when certain hotel rooms are booked in full, peers tend to consider other vacancies that are located closer to the rooms in previous queries. This assumption usually holds and is widely exploited in multiple scenarios such as operation system [19] and media streaming systems [3].

However, this assumption may also not hold, such that the utilization of the prefetched data items in future query processing is not guaranteed. Specifically, in our approach, prefetching decision is decided by the value of correlation threshold τ, which is computed based on the distance between well-replicated and poorly-replicated data items, orthogonal to the utilization of prefetched data items in future query processing. For instance, when queries migrate back and forth frequently, prefetched data items might be cached out before being utilized to answer queries. This leads to lower utilization rate of prefetched data items, compromising the cost-effectiveness of our approach.

To ease this problem, we introduce a mechanism that adaptively adjusts the volume of prefetched data based on the utilization rate of the data. During each specific period of time, each peer p approximates the utilization rate $0 \leq ur \leq 1$ of prefetched data items over the queries issued by p in previous periods. Specifically, consider a set S of all k queries where $S = \{q_1, q_2, ..., q_k\}$ that are issued by p during previous periods. Denote by function $utilized : S \times \{false, true\}$ such that, given a query $q_i \in S$, $utilized(q_i)$ equals to true if the query results of q_i contain prefetched data items, while $utilized(q_i)$ equals to false if query results do not exploit prefetched data items. Then ur is simply defined as $ur = \frac{\sum_{i=1}^{k} utilized(q_i)}{k}$. Note that each peer decides the value of ur independently since utilization rate is largely affected by the characteristics of the query load issued by each individual peer.

The adjustment of volume of prefetched data might affect query processing performance. To balance between the utilization of prefetched data and query processing performance, we consider query processing performance loss due to the shrink of prefetched data. Denote by t the period of time of adjustment of prefetched data volume. During each $t > 1$, peer p collects the query processing performance $hopNum_t$ by averaging the number of hops per query. Query processing performance loss pl_t is then defined as $\frac{hopNum_t}{hopNum_{t-1}}$. Then we denote by f_t the adjustment ratio with respect to the t_{th} period, where $f_t = min(1, ur \times pl_t)$.

During the t_{th} period, f_t will be piggybacked with all the queries issued by p. All data items that are

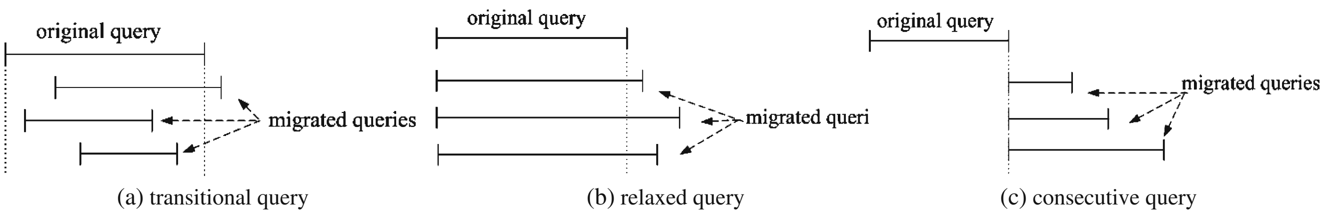


Fig. 3 Synthesized migrated queries (**a**–**c**)

correlated to query results are prefetched with a probability linearly proportional to f_t. Since the computation of f_t already takes the performance loss into account, when f_t is still low, it indicates that data items may be over-prefetched. Thus by reducing the prefetched data volume proportionally, the utilization rate of prefetched data is guaranteed to increase.

4 Performance evaluation

4.1 Experimental methodology

To study range query processing performance under the prefetch-based approach, we use the discrete event simulator p2psim.[10] We generate a network topology by randomly mapping peers to coordinates in a square area using a uniform distribution (which also generates local skew effects in the mapping). We consider a network of up to $N = 3000$ peers. Both a static network setting and a dynamic setting with network churn are simulated. For clarify, the configurable system parameters and their values are shown in Table 1.

We consider one-dimensional range queries within the domain of $R = [0, 10000]$. Initially each peer holds $i = 10$ items that are randomly chosen within the range. The range query load is synthesized as follows. The number of range queries within each *epoch*[11] follows Poisson distribution with mean $m = 100$; the query execution process runs for 20 epochs. To simulate the Power-law characteristics of query load that is common in real world [16], the range domain is evenly partitioned into a configurable number (*i.e.,* 10) of range segments; the start point (*i.e.,* lower bound) of each range query is generated over a randomly chosen segment based on the principles of *growth* and *preferential attachment*, which produce Power-law query distribution [2]. When a query is issued more than a number of times and becomes "popular", peers issuing it start to migrate the query.

[10]p2psim:http://pdos.csail.mit.edu/p2psim/

[11]In this experiment, each epoch lasts 5×10^6 ms.

To cover various peer behaviors, three migration patterns are considered (in Fig. 3), producing Transitional Queries, *relaxed queries* and *consecutive queries*. For each original query Q_p, a configurable number of migrated queries Q_r are generated and executed subsequently. The number (m) of the original queries and the number (m') of migrated queries per original query will be configured shortly.

- *Transitional queries*. All points in the data domain are potentially chosen as the start point of Q_r with a probability proportional to the Euclidean distance away from the start point of Q_p. The range span of Q_r follows Poisson distribution with mean $span = 100$. This migration pattern may simulate the scenario that, after finding that the hotel rooms around a specific city are all booked, peers may turn to check other proximate cities.
- *Relaxed queries*. The start point of each Q_r equals to that of Q_p, denoted by *start*. Then the range span of Q_r is relaxed to follow Power-law distribution between ($span$, $|D| - start$). This migration pattern covers those relaxed queries over different radius of the region around a specific location (*e.g.,* a conference site).
- *Consecutive queries*. The start point of each Q_r adopts the end point of Q_p. The range span of Q_r follows Poisson distribution with mean $span = 100$. This migration pattern captures the setting when peers adjust the search scope by retrieving the data that are disjoint but contiguous to initial geographical range constraints.

Since the random walk mechanism requires a randomization mechanism, all experimental results are averaged over three runs, each with a different random seed. Recall that both the number of messages and the query routing latency are inversely proportional to the number of query result replicas in either flooding or random walks; we only report the number of messages consumed. Since only the reply message sent back to query issuers carry query results and prefetched data while all other messages only carry the query itself (*e.g.,* range constraints), the amortized message size is small.

Table 1 Configurable system parameters

Parameter	Meaning	Value assignment
N	Network size	3000
i	The number of data items per peer	10
DR	Data range domain	[0, 10000]
$period$	Each simulation period	5×10^6 s
e	The number of periods for cost measurement	20
m_1	The number of popular queries per period	Poisson distribution ($mean = 100$)
m_2	The number of migrated queries per popular query	1, 5, 10
pr	Popularity ratio of data	$50\% - 80\%$
$1 - r$	Failure rate of peers per period	$10\% - 40\%$
d	Data update rate per period	1, 10, 100, 1000
s	Cache size (*i.e.,* the number of cached data items per peer)	500, 1000, 1500, 2000
k	Cache expiry periods (simulation seconds)	5×10^6, 1×10^7, 5×10^7, 1×10^8
h	Smoothing factor for kernel estimation	1
ps	Proactive preprocessing sampling number	$0.001 \times N$
pp	Proactive preprocessing period	2, 4, 8, 16 *periods*

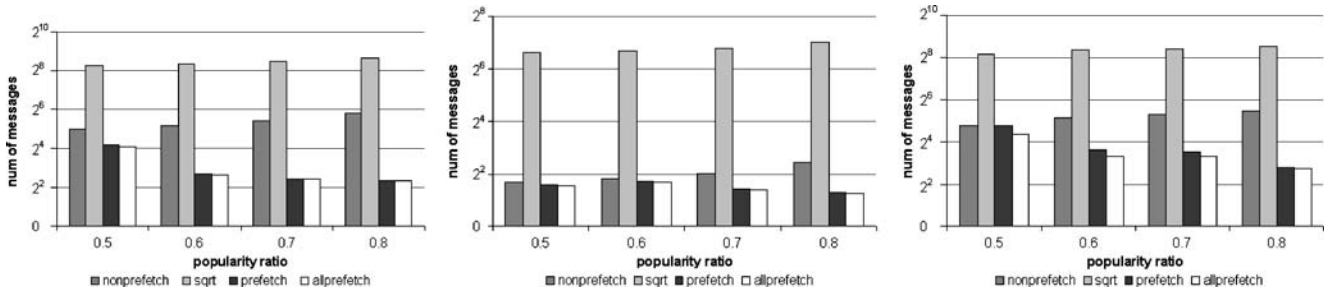
Thus, message size is not studied in the performance evaluation. In this simulation, the proportional and square-root replication schemes are considered as the baseline approaches. No overall storage constraints are assumed in this evaluation. Thus, the query processing costs incurred by the square-root replication scheme are even higher than those incurred by the proportional replication scheme, which supports the theoretic performance analysis presented in Section 2.

4.2 Evaluation of query shipping

For simplicity, this experiment assumes a static environment, where no network churn or data insertion occur. We will consider network churn shortly in Section 4.4. Initially, $m = 100$ queries are generated during each epoch by following Power-law distribution. When a query becomes "popular", $m' = 5$ migrated queries (respectively transitional, relaxed and consecutive queries) are generated based on each original query.

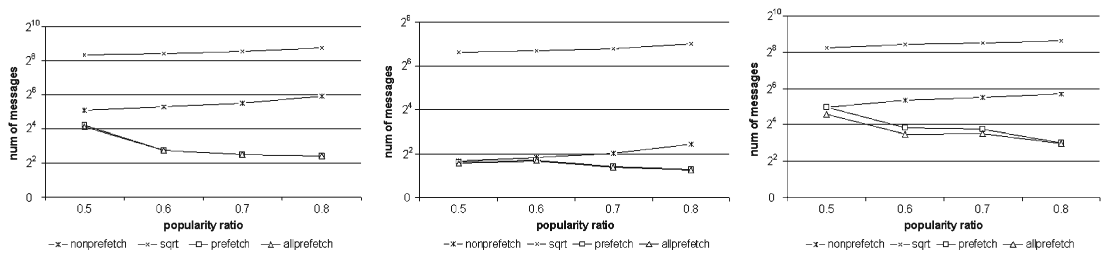
The average number of messages per query is shown in Fig. 4. In addition to the *nonprefetch* option that acts the same as the proportional replication scheme and the *prefetch* option that corresponds to our prefetch-based approach, we also consider *allprefetch* approach, which prefetches not only correlated poorly-replicated data items but well-replicated ones during the execution of range queries. For presentation, the results are illustrated with a logarithmic scale with base of 2.

The experimental results show that the prefetch-based approach effectively decreases the overall number of messages per query with respect to all migration patterns. In this experiment, the ratio (denoted by pr) corresponding to the x-axis of Fig. 4 may trigger the query migration: when an original query is issued by peers for a sufficiently large number of times (*i.e.,* $c \times pr \times N$), migrated queries are generated for this query, where N denotes the network size and $c = 0.6$



(a) overall with transitional queries (b) overallwith relaxed queries (c) overall with consecutive queries

Fig. 4 Average shipping cost per query (**a–c**)



(a) transitional query only (b) relaxed query only (c) consecutive query only

Fig. 5 Average shipping cost per query (**a**–**c**)

for this simulation. When *pr* is higher, the number of original queries is expected to increase, such that more migrated queries are generated, potentially including more poorly-replicated data items; consequently the average query execution performance of the *non-prefetch* option goes up. In contrast, the average query execution cost under the *prefetch* option is lower under all settings, showing the effectiveness of our approach. In Fig. 5a–c we also present the shipping cost consumed by only migrated queries, indicating that the performance improvement of our approach is primarily due to the cost savings over migrated queries. These figures also illustrate that the benefit of simply prefetching all correlated data items including well-replicated ones (under the *allprefetch* option) is not significant. We measure the bandwidth consumption of the *prefetch* option versus that of the *allprefetch* option, which can be 60% more. This confirms our belief that data popularity should be considered in prefetching.

Since "relaxed queries" are generated by following Power-law distribution and a small set of migrated queries are issued with an exponentially higher probability, the (initially) poorly-replicated data items covered by corresponding query results become well replicated very quickly. Thus the number of messages per query is significantly lower than that of the queries generated under the other two migrated patterns. This holds for other experimental results over "relaxed queries" in the remainder of the paper.

With respect to the load of migrated queries, we evaluate different numbers of migrated queries (*i.e.,* $m' = 1$, 5 and 10), with $pr = 0.7$. As shown in Table 2, the average number of messages decreases because a relatively larger number of queries (over poorly-replicated data items) may benefit from the prefetching of poorly-replicated data.

4.3 Evaluation with cache constraints

In practical systems, peers usually have caches with limited storage sizes. Moreover, cached data items may expire after a period of time, as discussed in Section 3. In this simulation, we measure the query shipping cost with respect to these cache constraints.

We consider 500, 1000, 1500 and 2000 cache entries (*i.e.,* the cached data items per peer). The experimental results are shown in Table 3, which indicate that when the cache size is larger, the average query shipping cost tends to be lower. This is primarily due to the fact that larger caches can hold more prefetched data, facilitating query execution. We also evaluate the query processing performance under various cache expiry periods (*i.e.,* 5×10^6, 1×10^7, 5×10^7 and 1×10^8 simulation milliseconds). The results (shown in Fig. 6) demonstrate that when the lifespan of cached data items increases, the average number of messages per query decreases. This is because longer cache life means that data expire less frequently, leading to better use of cached data items.

4.4 Evaluation under dynamic settings

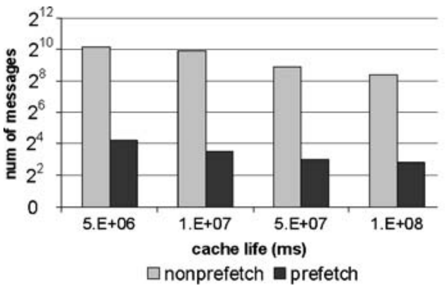
In P2P networks, peers may leave and fail arbitrarily, affecting the number of cached data items in the network. Moreover, peers may join the network with new

Table 2 Query execution cost with different query load

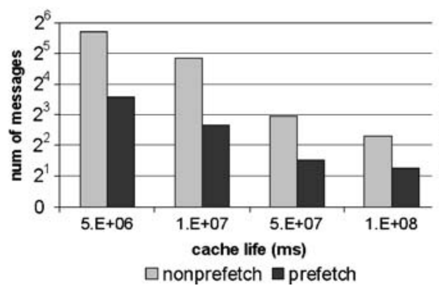
	m'		
	1	5	10
Transitional	10.47	5.39	4.61
Relaxed	2.66	2.31	2.30
Consecutive	9.93	7.59	7.59

Table 3 Query execution cost under various cache sizes

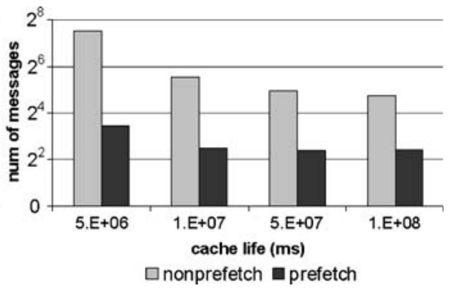
	Cache size			
	500	1000	1500	2000
Transitional	357	321	255	241
Relaxed	2.73	2.11	2.07	2.07
Consecutive	72	24	16.8	12.19



(a) transitional query



(b) relaxed query



(c) consecutive query

Fig. 6 Query execution cost per query under cache expiry (**a**–**c**)

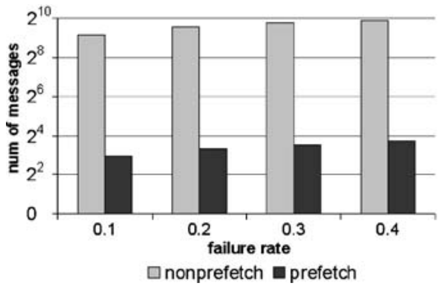
data items and data insertion manipulation may also be issued by existing peers.

We simulate the network churn by allowing each peer to leave with a probability of r during each epoch, where r varies from 0.1 to 0.4. To make the network stable, peers join the network at the same rate. The number of messages per query is shown in Fig. 7. The experimental results indicate that when failure rate increases, the number of messages per query goes up, which is due to more random walks being required for completing the query processing during network churn.

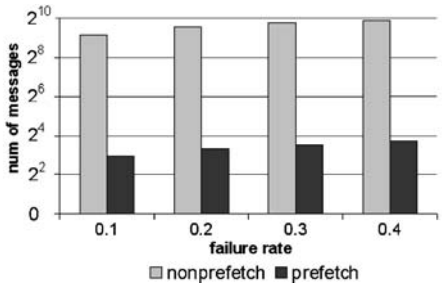
To measure range query processing performance when data insertion occurs, peers generate a configurable number (*i.e.,* 1, 10, 100 and 1000 per epoch) of new data items that are randomly chosen within the data domain. In this experiment, no migrated queries are generated. As shown in Fig. 8, the average shipping cost per query increases steadily during the insertion of new data items. This is not surprising, because when peers keep issuing the same range queries, they still need to locate the new data items that initially are poorly-replicated . Similar to data insertion, the deletion of data items may also incur poorly-replicated data items, affecting range query execution performance in a similar way. Thus, data deletion will not be studied separately. Moreover, since data items are assumed to be read-only (*e.g.,* song files or hotel address information), data insertion or deletion does not incur inconsistency among cached data replicas. Thus data consistency is not an issue in this context.

4.5 Evaluation of bandwidth and adaptivity

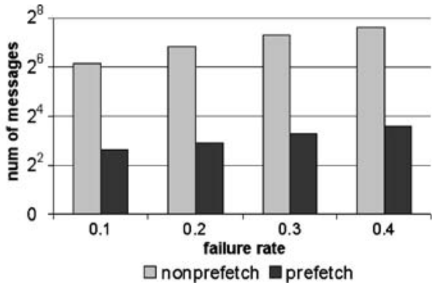
The correlation threshold τ is computed adaptively based on the knowledge of executed range queries. In our performance studies, we compare the average bandwidth costs of data prefetching per query with respect to a query load consisting of the same number of queries but with different migration query loads: either $m' = 1$ migrated query is generated for a "popular" original query or $m' = 10$ migrated queries are generated. The results shown in Table 4 indicate that the bandwidth consumed by the prefetch-based approach with a relatively larger number of migrated queries is higher. This is because the execution of more queries is affected by poorly-replicated data items and τ tends to be larger, increasing the volume of prefetched data. In contrast, when migrated queries are fewer, the value of τ is smaller. Thus the volume of prefetched data is lower, demonstrating the adaptivity of our approach.



(a) transitional query



(b) relaxed query



(c) consecutive query

Fig. 7 Query execution cost per query under network churn (**a**–**c**)

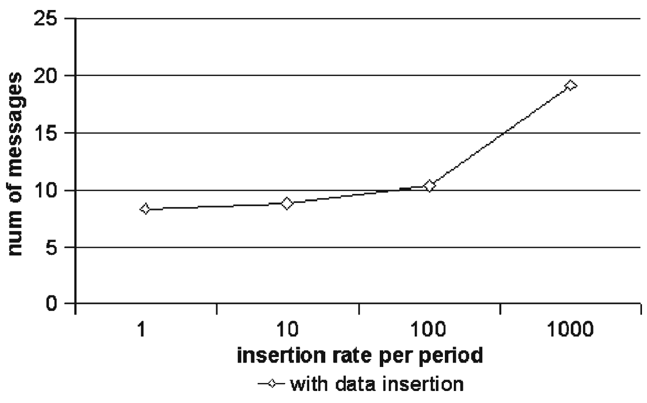


Fig. 8 Query execution cost per query with data insertion

4.6 Adjustment of prefetched data volume based on utilization rate

We evaluate the approach to adjust prefetched data utilization rate addressed in Section 3.6. Without loss of generality, we consider consecutive query type. During the running of queries for five periods, where each period lasts 10^7 simulation milliseconds. Since the query processing cost does not change significantly during these periods, as shown in Table 5, the prefetched data volume is adjusted exclusively based on the current utilization rate.

It is not surprising that, the volume of overall prefetched data items decreases, as shown in Fig. 9a. Correspondingly, the utilization rate of prefetched data items increases, as presented in Fig. 9b.

5 Related work

Various architectures have been proposed to support large-scale P2P applications, including unstructured, super-peer-based, and structured architectures [21]. In this work, we focus on unstructured P2P architecture, which imposes little constraints on the overlay network structure and has been widely employed in practical systems such as Gnutella, KaZaa, and BitTorrent.

Multiple approaches have been proposed to support range query processing in P2P networks. Structured overlay network protocols, such as BATON [11], PHT [14], P-Tree [5] and others, employ tree structures to interconnect peers and manage distributed data. Based on efficient distributed range indexes, range queries can be solved within a scalable number of hops. However, these protocols require non-trivial construction and maintenance cost; moreover, they do not exploit the cached data that are enabled during the query execution, which may affect query processing performance.

Caching techniques have been employed in P2P systems [22], focusing on the routing efficiency rather than the range query processing that is addressed in this work. Distributed range caching mechanisms have also been developed for P2P networks [13, 17]. Kothari et al. propose a distributed tree-based index to manage all range caches to facilitate the search of range data in P2P networks [13]. Sahin et al. employ a hyper-rectangle-based overlay network to index multi-dimensional range data to support efficient range query processing [17]. These works use structured overlay networks in indexing range caches, which may potentially be affected by their non-trivial construction and maintenance cost. In contrast, our approach is focused on range caching in unstructured P2P overlay networks.

In unstructured P2P networks, effective search strategies are developed based on constrained flooding [23] and random walks [7]. A data replication scheme is employed in Bubblestorm [20], which does not take query distribution into consideration, potentially leading to load-balancing problems. In contrast, proportional and square-root replication schemes [4] are devised that consider query distribution. In this work, we consider a prefetch-based approach to facilitate the caching of poorly-replicated data items to improve range query processing performance, which can be directly deployed with these replication schemes. Moreover, our approach may prefetch different volumes of data items according to the knowledge of executed range queries, enhancing the adaptivity of the approach. An adaptive replication scheme has been studied in P2P networks [8], which however is focused on employing server load measurements to reduce replication cost.

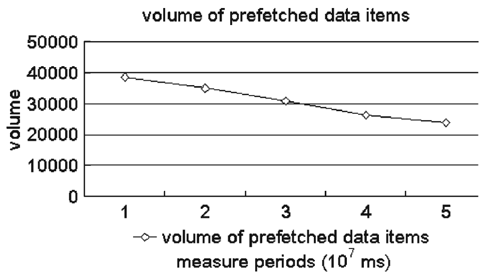
Table 4 Bandwidth consumption per query with different migration queries

	m'	
	1	10
Transitional (byte)	87500	72981
Relaxed (byte)	18	2
Consecutive (byte)	40652	38274

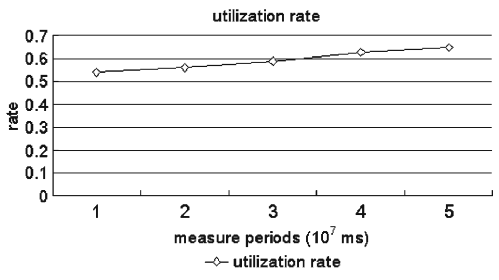
Table 5 Average number of messages per query

Period (10^7 ms)	Average number of messages per query
1	36.19
2	36.35
3	36.07
4	36.23
5	36.25

Fig. 9 Adjustment of prefetched data volume based on utilization rate (**a**, **b**)



(a) prefetched data volume



(b) utilization rate

Prefetching has been used in operating systems to facilitate instruction feed to CPU by fetching all possible instructions related to a branch conditional test in advance [19]. Our approach handles P2P range query processing and exploits data popularity to improve effectiveness of the prefetching. P2P media streaming systems also employ prefetching techniques to buffer upcoming stream data for smooth playback of the stream [3], which focus on adjustment of the volume of prefetched data and are specific to streaming systems.

6 Conclusion

Under unstructured P2P overlay network architecture, query processing performance is usually decided by the number of query result replicas in the network. When range queries involve poorly-replicated data items, query execution performance degrades. In this work, we propose a popularity-aware prefetch-based caching approach that effectively facilitates the caching of poorly-replicated data items that are correlated with well-replicated ones, resulting in cost savings for future queries that access the poorly-replicated data. Our approach does not require strong altruistic cooperation from peers since only query issuers that use the query processing services incur prefetching overhead. Under various query load settings, we prove that the performance of range queries involving poorly-replicated data is guaranteed to improve. Experimentally, we also show that our proposed prefetch-based approach delivers substantial query processing cost savings.

References

1. Balke W, Nejdl W, Siberski W, Thaden U (2005) Progressive distributed top-k retrieval in peer-to-peer networks. In: Proc int conf on data engineering, pp 174–185
2. Barabási A-L, Albert R (1999) Emergence of scaling in random networks. Science 286:509–512
3. Cheng B, Liu X, Zhang Z, Jin H (2007) A measurement study of a peer-to-peer video-on-demand system. In: Peer-to-peer systems, first international workshop
4. Cohen E, Shenker S (2002) Replication strategies in unstructured peer-to-peer networks. In: Proc ACM SIGCOMM, pp 177–190
5. Crainiceanu A, Linga P, Gehrke J, Shanmugasundaram J (2004) Querying peer-to-peer networks using P-Trees. In: Proc 7th int workshop on the world wide web and databases (WebDB), pp 25–30
6. Edwards HM (1974) Riemann's zeta function. Academic, London
7. Gkantsidis C, Mihail M, Saberi A (2004) Random walks in peer-to-peer networks. In: Proc 23rd annual joint conference of the IEEE computer and communications societies
8. Gopalakrishnan V, Silaghi B, Bhattacharjee B, Keleher P (2004) Adaptive replication in peer-to-peer systems. In: Proc 24th int conf on distributed computing systems, pp 360–369
9. Huebsch R, Hellerstein JM, Lanham N, Loo BT, Shenker S, Stoica I (2003) Querying the internet with PIER. In: Proc 29th int conf on very large data bases, pp 321–332
10. Iyer S, Rowstron AIT, Druschel P (2002) Squirrel: a decentralized peer-to-peer web cache. In: Proc ACM SIGACT-SIGOPS symp on principles of dist comp, pp 213–222
11. Jagadish HV, Ooi BC, Vu QH (2005) BATON: a balanced tree structure for peer-to-peer networks. In: Proc 31th int conf on very large data bases
12. Jelasity M, Voulgaris S, Guerraoui R, Kermarrec A-M, van Steen M (2007) Gossip-based peer sampling. ACM Trans Comput Syst 25(3)
13. Kothari A, Agrawal D, Gupta A, Suri S (2003) Range addressable network: a P2P cache architecture for data ranges. In: Peer-to-peer computing, pp 14–22
14. Ramabhadran S, Ratnasamy S, Hellerstein JM, Shenker S (2004) Brief announcement: prefix hash tree. In: Proc ACM SIGACT-SIGOPS symp on principles of dist comp
15. Ramakrishnan R, Gehrke J (2002) Database management systems. McGraw-Hill, New York
16. Ramasubramanian V, Sirer EG (2004) The design and implementation of a next generation name service for the internet. In: Proc ACM SIGCOMM, pp 331–342
17. Sahin OD, Gupta A, Agrawal D, Abbadi AE (2004) A peer-to-peer framework for caching range queries. In: Proc 20th int conf on data engineering, pp 165–176
18. Scott D (1992) Multivariate density estimation: theory, practice and visualization. Wiley, New York
19. Stallings W (2004) Operating systems: internals and design principles. Prentice Hall, Englewood Cliffs
20. Terpstra WW, Kangasharju J, Leng C, Buchmann AP (2007) Bubblestorm: resilient, probabilistic, and exhaustive peer-to-peer search. In: Proc ACM SIGCOMM, pp 49–60

21. Valduriez P, Pacitti E (2004) Data management in large-scale P2P systems. In: High performance computing for computational science—VECPAR 2004, 6th international conference, pp 104–118
22. Wang C, Xiao L, Liu Y, Zheng P (2006) DiCAS: an efficient distributed caching mechanism for P2P systems. IEEE Trans Parallel Distrib Syst 17(10):1097–1109
23. Yang B, Garcia-Molina H (2002) Improving search in peer-to-peer networks. In: Proc 22nd int conf on distributed computing systems, pp 5–12
24. Zhang R, Hu YC (2005) Assisted peer-to-peer search with partial indexing. In: The 24st annual joint conference of the IEEE computer and communications societies, pp 1514–1525

Qiang Wang obtained Ph.D. from Computer Science department of University of Waterloo, Canada in July 2008. My supervisor is Dr. M. Tamer Özsu. I graduated from Computer Science department at Nanjing University of China with Master's (2001) and Bachelor's degree (1998); I was working as a full-time software engineer at Motorola Software Center, China from June, 2001 to July 2002.

M. Tamer Özsu is Professor of Computer Science and University Research Chair at the University of Waterloo and Director of the David R. Cheriton School of Computer Science. Dr. Özsu's current research focuses on Internet-scale data distribution, multimedia data management, and XML query processing and optimization. He is a Fellow of ACM, a Senior Member of IEEE, and a member of Sigma Xi.

Khuzaima Daudjee is a faculty member in the David R. Cheriton School of Computer Science at the University of Waterloo. His research interests are in distributed and peer-to-peer systems, database systems and software engineering. He holds a Ph.D. in computer science from the University of Waterloo.