CoCache: Query Processing Based on Collaborative Caching in P2P Systems

Weining Qian, Linhao Xu, Shuigeng Zhou, and Aoying Zhou

Department of Computer Science and Engineering, Fudan University

Abstract. Peer-to-peer (P2P) computing is gaining more and more significance due to its widespread use currently and potential deployments in future applications. In this paper, we propose CoCache, a P2P query processing architecture that enables sophisticated optimization techniques. In the scenario of CoCache, a large number of peers, each of which may be attached with a local relational database and a cache, are connected through an arbitrary P2P network. CoCache is different from existing P2P query processing systems in three ways. First, a coordinator overlay network (CON) maintaining the summary of the whole system is constructed by applying DHT technique to query plan trees. CON protocol ensures the efficiency for handling dynamic environments. Second, a preliminary cost-based optimization technique for retrieving appropriate cached copies of data is studied. With the help of CON, we show the possibility of fine optimization in even large scale and dynamic distributed environments. Last but not the least, the collaborative caching strategy is presented, with which even small portion of cache storage on each peer may result in great improvement on query processing performance. Extensive experiments over real-world and synthetic settings show the effective and efficiency of CoCache.

1 Introduction

Peer-to-peer (P2P) computing is now gaining more and more interests in both academia and industry, due to the scalability, self-organization, load-balancing, autonomy and anonymity provided by systems with large amount of resources (e.g., bandwidth, storage and CPU cycles), contents and services shared by various peers. Enabling query processing is a natural extension of key and keyword based search in existing P2P systems [2, 8, 1, 5, 12]. There are several challenges to implement complex query answering functionalities in P2P systems. First of all, as in any P2P system, peers can join and leave the system anytime, anywhere and anyhow, which results in a purely dynamic and ad hoc network environment. Thus, the underlying protocol should be robust enough to handle peer and network failure. Secondly, a full decentralized process must be adopted for query processing. In a dynamic P2P environment, due to the lack of global information, both query execution and optimization become difficult. At last, the collaboration of autonomous peers is essential to fully take advantage of the resources in the system. This usually involves more optimization issues, such as coordination, locality-aware peer clustering, and load balancing. In summary, P2P query processing should be effective and efficient for handling *dynamic* networks, and large scale *distributed* yet *autonomous* peers.

In this paper, we present CoCache, a query processing system with collaborative cache. Caching or replication is widely adopted in centralized and distributed systems. It has several advantages, one of which is that data is available even when the source is temporarily inaccessiable. Furthermore, since the retrieval of cached data is usually much cheaper than that of the source, caching is often used as an optimization technique for decreasing latency. At last, in distributed systems, cached objects become partial copies of source that can serve different requests from different machines at different time. Therefore, caching is also employed in many P2P systems, and studied intensively [4, 14, 3, 12].

CoCache is different from existing P2P systems using cache. First, each peer collaborates with other ones to determine what to be cached. Intuitively, a peer tends to the cache data complementary to cached data in nearby peers. Furthermore, both the caching process and query processing, i.e. the process to find cached or source data, are fully decentralized based on a distributed hash table (DHT) scheme, called CON, for Coordinator Overlay Network. The third difference is the cost-based optimization employed in dynamic environments. Query answering performance is improved greatly with low overhead for maintaining CON. In summary, the contributions of this paper are as follows:

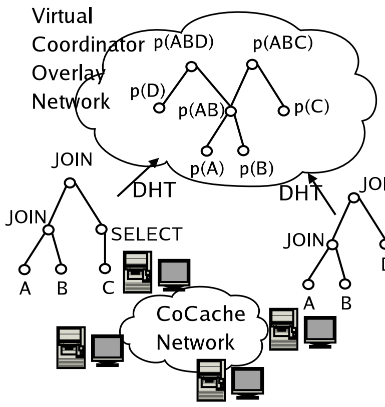
- A P2P query processing framework, CoCache, is presented. One of its main features is that a DHT-based subnet called CON is used to index the resources, i.e. data sources and cache. Analytical results show the ease and efficiency for CON maintenance.
- A cost-based optimization scheme is introduced to CoCache. With low overhead of statistics exchanging, query performance can be improved greatly. Experimental results show that for frequently posed queries, this technique is even more efficient.
- An implementation of the collaborative caching strategy, which is another main feature of CoCache is introduced and empirically studied. Experimental results show that even a small portion of storage devoted for caching on each peer can improve the performance of query processing greatly.

The rest part of this paper is organized as follows. In Section 2, the architecture and protocol of CoCache, as well as CON, are introduced. The query processing and optimization scheme is introduced in Section 3. We introduce the details of implementation for collaborative caching in Section 4. In Section 5 extensive empirical study of CoCache is introduced. After the related work introduced in Section 6, Section 7 is devoted for concluding remarks.

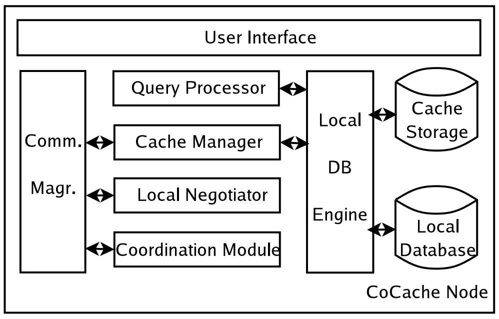
2 The Architecture of CoCache

2.1 The CoCache and CON Networks

A peer may take different roles in CoCache network: a *requester* is a peer who issues one or more queries. A *source peer* is a peer whose database is accessible



(a) CoCache and CON network



(b) Peer architecture of CoCache

Fig. 1. Archiecture of CoCache network and nodes.

by other ones. A *caching peer* is a requester who caches result(s) of its queries or subqueries. Both source peers and caching peers are called *providers*. A *coordinator* is a peer in charge of maintaining information about a specific query expression. The information includes the providers that can provide data to answer the query, the locality information of the providers, and the coordinators corresponding to the sub- and super-expressions. The coordinators are also responsible to coordinate the requesters to determine which part of data to be cached by which peer.

Figure 1 (a) illustrates the architecture of a CoCache network. A CoCache network can be an arbitrary peer-to-peer network[1]. Each peer may share its data with other peers and pose queries. Each node in the query plans is mapped to a specific node via distributed hash table (DHT), who will become the coordinator of the query expression. A coordinator maintains a finger table in which each entry points to the coordinator of a super- or sub-query expression. The coordinators form a *virtual* coordinator overlay network (CON), that is embedded in CoCache. The architecture of a node in CoCache network is shown in Figure 1 (b). Different from other peer data management systems, each node is equipped with two modules called *Local Negotiator* and *Coordination Module*. The former module takes the responsibility of negotiating with coordinators to determine what data should be cached, while the later is in charge of the coordination among the requesters for collaborative caching when the node becomes a coordinator.

A query is represented by a relational calculus expression[2], which can be transformed to a query tree. The peer identifier $p(v)$ of the coordinator corresponding to a node v in the tree is determined by using the following rules:

[1] Currently, CoCache is developed based on BestPeer[7].

[2] In this paper, we do not consider the equivalence of two query expressions.

1. If v is a leaf node, $p(v) = h(v)$, in which $h()$ is a general purpose hash function, such as MD5 or SHA.
2. For node v corresponding to a unitary operator, such as σ or π, $p(v) = p(v')$ in which v' is the child of v.
3. For node v corresponding to a binary operator, i.e. $\bowtie$, $p(v) = p(v_1)|p(v_2)$, in which v_1 and v_2 are the children of v, and $|$ means bitwise OR of two bit strings.

Thus, given a query, the coordinator of each sub-query can be determined by using consistent hashing $h()$. Coordinator $p(v)$ is also called the *host* of node v.

Each coordinator maintains a finger table, in which entries are coordinators corresponding to parent and children nodes in the query tree. Formally, given a query q, v is a node in the query tree, while v', v_1 and v_2 are parent, children nodes of v respectively. Then, $< v', v, p(v') >$, $< v, v_1, p(v_1) >$ and $< v, v_2, p(v_2) >$ are three entries in peer $p(v)$'s finger table. Note that for coordinators corresponding to more than one query expressions, the hosts of parent and children nodes in each query should be included in the finger table. The peers in the finger table are called its *neighboring coordinators*.

The coordinators are logical peers. In a P2P network, it is possible that there is no peer whose identifier is the same as the identifier of a specific coordinator. Different P2P platforms use different ways for handling this kind of problems. Chord, for example, uses the peer whose peer-id is the first one follows a specific identifier in the identifier space to be responsible for the tasks assigned to that specific identifier [13]. Another popular P2P platform CAN uses the closest peer in the torus-like identifier space to take over the tasks for a specific identifier [10]. CoCache is implemented on top of a hybrid P2P system, BestPeer [7], in which superpeers, called LIGLO servers, are responsible for this physical-logical identifier mapping task. It should be noted that although BestPeer is chosen as the bottom platform, CoCache is independant of underground layer. It can be moved to other P2P platforms with few modifications.

Peer Join When joining a CoCache network, a requester first determines the coordinators corresponding to its queries. Then, the queries along with the peer information, such as peer identifier, locality information and other statistics about the peer, is sent to the coordinators. The requester also collects information about the cached data from the coordinators. Thus, a query plan is generated, and evaluated by retrieving data from providers in the query plan[3]. If a requester agrees to cache data, it informs the coordinators of the cache. The coordinators updates their local index when new caching peer's notification is received. When a coordinator has collected a set of updates of caching peers, it initiates a re-caching process, which is introduced in Section 4.

It is possible that a new arrival peer's identifier is more suitable for a logical coordinator than the current one's. The new peer takes over the information of the old coordinator including the finger table. Then, it sends update information to all neighboring coordinators, so that they can update their finger tables.

[3] The details of query processing are introduced in the next section.

Peer Leave For the leave of a requester or provider, or the drop of a query by a requester, the leaving peer informs the coordinators to update their indexed information. If the leaving peer is a coordinator, it contacts the next suitable peer in the system to take over the coordination information. It also informs its neighboring coordinators to update their finger tables to point to the new one.

Failure Handling A peer may leave the system due to power, hardware or network failure. In such cases, a peer may not be able to inform other peers or coordinators. A failure may be detected by various ways. The failure of a coordinator may be found due to a connection failure of a neighboring coordinator or a requester, while the failure of a provider may be detected when a requester tries to retrieve data from the failed peer. In CoCache, each requester sends the information about its queries to the coordinators periodically. When a coordinator fails, the information is routed to the new coordinator, which takes over the work of failed one automatically. When a peer notices the failure of a provider, it informs the corresponding coordinators to update the index. The coordinator then forward this message to all requesters that are registered to retrieve data from the failed peer[4].

3 Query Processing in CoCache

A cache is a binary tuple (v, N), in which v is the logical expression, and N is a multiset of peers, which is called the *container* of the cache. For each occurance $p \in N$, peer p contributes fixed size of storage space to caching data of logical expression v. Note that a source peer can be treated as a special cache, whose v is the data source, and N is the set only having the peer itself.

A *query plan* of a query q is a set of caches $P\{(v_i, N_i)\}$ satisfied that $\bigcup_i R(v_i) \supseteq R(q)$, in which $R(v_i)$ and $R(q)$ mean the relations with logical expressions v_i and q respectively. The cost of a query plan is defined as

$$cost_q(q|P) = \sum_{i, v_i \in P} c(v_i \to p_q) + C(\{v_i\}, q) \tag{1}$$

in which $c(v_i \to p_q)$ is the cost for transmitting cache (v_i, N_i) to requester p_q, while $C(\{v_i\}, q)$ is the computation cost to evaluate query q given cached data.

In implementation, $c(v_i \to p_q)$ is estimated using $|R(v_i)| \times b_{v_i, p_q}$, where $|R(v_i)|$ is the size of the relation, and b_{v_i, p_q} is the cost to transfer one unit of data from peers in the container to the requester. Assuming that peers in a container is close to each other in a subnet, b_{v_i, p_q} can be approximated by using b_{p_j, p_q}, in which p_j is an arbitrary peer in the container. Ping-pang protocol can be used for estimating b_{p_j, p_q}.

Having received a query, after the host of the query is determined, a requester sends the query along with the peer information to the coordinator, and retrieves the part of index about available caches. In case the caches cannot satisfy the

[4] The details about cache selection and registration is introduced in the next section.

Algorithm 1 Query processing in CoCache

```
Input: query $Q$, cost threshold $t$ Output: query plan $P$
 1: send($p(Q)$,$Q$);$C \leftarrow \{< Q, p(Q) >\}$;$V \leftarrow Q$;{Send query $q$ to coordinator $p(q)$}
 2: $R \leftarrow$retrieve($p(Q)$,“SELECT $C(v_i, N_i)$ FROM CACHE WHERE $v_i = Q$”);
 3: $best \leftarrow$ choose($R$);
 4: while $cost_q(best) > t$ do
 5:   for all $< v_0, c >\in C$ do
 6:     $C' \leftarrow C' \bigcup$retrieve($c$,“SELECT $< v', p >$ FROM FT$(v, v', p)$ WHERE $v = v_0$”);
 7:   end for
 8:   $C \leftarrow C'$;
 9:   for all $< v', c' >\in C$ do
10:     $R \leftarrow R \bigcup$retrieve($c'$,“SELECT $C(v_i, N_i)$ FROM CACHE WHERE $v_i = v'$”);
11:   end for
12:   $best \leftarrow$ choose($R$);
13: end while
14: returen $best$;
```

query processing request, the neighboring coordinators corresponding to the sub-queries are contacted, and more information of caches is collected. This process is iterated until a query plan satisfying the cost request is found. (Algorithm 1).

4 Collaborative Caching

A *cache plan* is a set of valid caches, $P\{(v_i, N_i\}$. Its cost is defined as

$$cost_c(P) = \sum_{v_i \in P} cost_q(v_i | P_{v_i}) \tag{2}$$

in which P_{v_i} is the query plan to answer query (cache) v_i satisfying that $P_{v_i} \in P$. The aim of collaborative caching is to minimize the cost of the cache plan. Note that the purpose of query processing is different from that of caching in that the former aims at optimizing a specific query on a specific peer, while the later tries to coordinate different peers to achieve a globally optimized caching solution.

The caching process is driven by the coordinators. When a coordinator has collected a set of updates about the requesters, it starts a re-caching process. Otherwise, when detecting a new requester that agrees to cache some data, the coordinator assigns it to the cache in the query plan whose $c(v_i \rightarrow p_q)$ is the maximum. The main process of caching can be roughly divided into three phases: cache plan initialization by CON, negotiation, and construction of cache.

4.1 Cache Plan Initialization by CON

A coordinator partitions requesters with common sub-query v, and being willing to cache, into k groups based on their locality information. Thus, peers within the same group can communicate to each other via high-bandwidth connections. Then, the small groups N that are not capable enough to cache the subquery, i.e. (v, N) is not a valide cache, are assigned to nearby groups of peers.

The partitions of requesters are populated in CON with K levels, which is called *coordination level.* Thus, each coordinator has collected a set of candidate caches $P'\{(v_i, N_i)\}$, in which each v_i is a logical expression and N_i is a group of peers. Each coordinator greedily choose caches with maximum cost gain per unit of storage for queries $\{v_i\}$. Here, cost gain of a cache c for a specific query q is defined as $\Delta_q(c) = cost_q(q|P) - cost_q(q|P \cup \{c\})$. Thus, the cost gain is evaluated using $\Delta(c) = \sum_{v_i \in P'} \Delta_{v_i}(c)$. The chosen caches are put into the candidate cache plan P, and the record of free space on a requester is decreased. This process is iterated until no requester has free space or no valid cache remained.

The caches (v_i, N_i) in the candidate cache plan whose logical expression v_i is the same as that of the coordinator are chosen as initial caches. This process is conducted in the spare time of the coordinators.

4.2 Negotiation for Caching

A coordinator informs the peers in the containers of initial caches. A requester may receive several notifications from different coordinators. It chooses the caches with logical expressions closest to that of the query of the requester to cache, until no free space is left. The requester sends the feedback to the coordinators. The coordinator removes the peers do not agree to cache from the containers, and checks if the cache is still a valid one. The requesters in the valid caches are informed, so that they retrieve the data for caching. For those invalid caches, a coordinator informs the requesters in the containers, so that the requesters can reassign the spaces left for the invalid cache for other caches.

4.3 Cache Construction

When a requester receives is notified for caching by a coordinator, it begins the cache construction process. If the free storage space is large enough for caching all result of the logical expression, the requester evaluates the query corresponding to the expression, and caches the result. Otherwise, the requester chooses the portion of result that are not cached by other peers in the container. In CoCache implementation, data are partitioned into chunks with identifiers. The identifiers are assigned to the peers in the container via hashing. Furthermore, each caching peer multicast its cached chunks to those peers in the same container. Each peer in the container is responsible for maintaining the cache, and serve other requesters. Since only nearby peers are put into the same container in cache plan initialization process, the peers in the container may serve each other efficiently.

4.4 Query Processing Implementation

Each CoCache peer is equiped with a query engine[5]. Queries are written in standard SELECT-FROM-WHERE form without aggregation. Some peers may share the common schema of data. Each requester knows the schema of data to

[5] Current implementation of CoCache uses IBM DB2 UDB 7.1 as query engine.

be queried. We believe that this setting is common in many applications, such as information management in large enterprises. The problem of schema discovery can be solved using information retrieval style method introduced by PeerDB[8].

The logical query expressions are translated to SQL queries for query processing. For retrieval of data in caches, a query is transformed to a set of SQL queries for chunks respectively, and then sent to the caching peers in the container. In case that a caching peer fails, the corresponding subquery is sent to data source to retrieve the missing chunks. The caches are retrieved simultaneously. After obtaining all the data cached (or from data sources), the query is evaluated, and the result is returned to users. Cache construction is similar to query processing. A requester or caching peer periodically re-evaluate its query. The problem of ensures consistency of query result in P2P systems is left as open problems for further study.

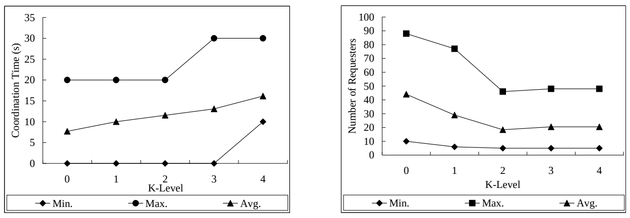
5 Empirical Study of CoCache

5.1 Simulation Experiments

For synthetic data, we generate a table with 32 columns and 128 rows blocks, in which each block is in the same size, and is marked with coordinate (rid, cid). Thus, given a quadruple $(top, left, bottom, right)$, the blocks whose coordinate satisifies $left \leq cid \leq right$ and $top \leq rid \leq bottom$ are determined. Each peer generates such a quadruple to determine the data to be stored, while each query is also a quadruple. The data overlapped by a query is the answer to the query. If any blocks on any single peer only overlaps part of the blocks of a query, the data on different peers must be joined together. The query with maximum number of joins involves data on sixteen peers to be joined. The synthetic data set is tested in a P2P system simulator. One thousand peers are simulated. The peers are distributed in a network with topology generated by the program downloaded from `http://www.cc.gatech.edu/fac/Ellen.Zegura/gt-itm/gt-itm.tar.gz`,

In addition to CoCache, PeerDB [8] without caching (PDB-NC) is used as baseline, while PeerDB with caching (PDB-C) is compared with CoCache. Note that in the later case, each peer devotes the same size of storage space for caching.

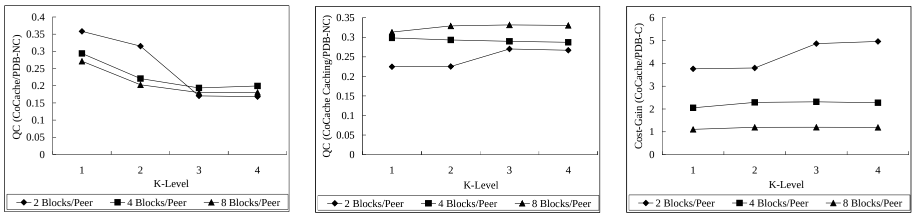
In Figure 2 (a), the minimum, average and maximum runtime of cache plan intialization is shown. It is obvious that the runtime is ascendant with the increasing of coordination level K. For some coordinators with few corresponding requesters, the process may be very fast. Furthermore, it is shown that even K is set to 3, the average runtime of coordinators does not increase much. Since in worst case, the candidate cache plans collected by a coordinator is explosive to K, the performance goes bad when K is larger than 4. However, the average runtime is linear to K in our experiments. In Figure 2 (b), the requesters a coordinator should negotiate with is shown. The larger K is used, the less requesters a coordinator should negotiate with. The reason is that by exchanging candidates caches, a coordinator may drop a lot of caches that are not preferred for



(a) Runtime for cache plan initialization.

(b) Number of requesters to be noticed per coordinator.

Fig. 2. Workload on coordinators with different coordination levels.



(a) Query processing elapse time comparison of CoCache and PDB-NC.

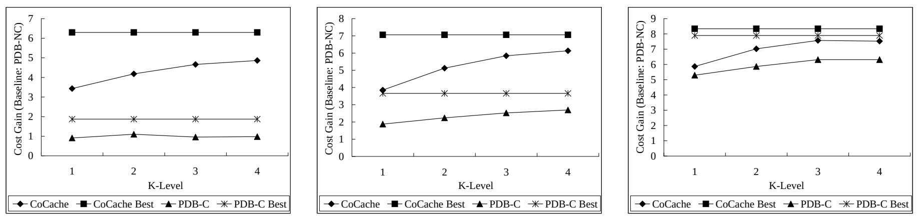
(b) Cache construction elapse time compared with that of query processing in PDB-NC.

(c) Cost gain of CoCache compared with PDB-C.

Fig. 3. The cost of query processing, view construction compared with PDB-NC and PDB-C, with different settings of caching space and coordination level.

their limited contribution for increasing cost gain. It is shown that when K is equal or larger than 2, the view candidate refinement process eliminates a large amount of view candidates. Thus, setting $K = 2$ may save both computation and communication cost.

Figure 3 (a) shows the query processing cost comparison between CoCache and PDB-NC, in the condition that caches are constructed, while the cache construction cost is shown in Figure 3 (b). It is shown that both query processing and cache construction cost is quite small when compared with the query processing cost of PDB-NC. In Figure 3 (c), CoCache is compared with PDB-C on query processing cost. CoCache outperforms PDB-C when each peer contributes limited storage for caching (2 blocks/peer). Even when the storage for caching is large enough for the whole query result (8 blocks/peer), CoCache is slightly better than PDB-C on average. If collaborative caching is less frequent than query processing, such as applications of continuous query processing, CoCache is more efficient than PDB-NC and PDB-C, and its advantage is much more obvious when cache space is limited.



(a) 2 blocks for caching per peer (b) 4 blocks for caching per peer (c) 8 blocks for caching per peer

Fig. 4. Cost gain comparison of CoCache and PDB-C, divided by that of PDB-NC.

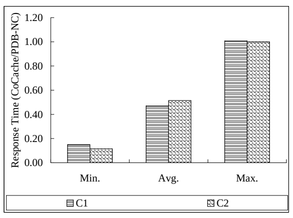
The cost gains under different K's and different block-size settings are shown in Figure 4. Here, CoCache-Best means the cost gain obtained when each coordinator knows the status of the whole system, which is an ideal condition and is impossible to be reached in applications, while PDB-C Best means the cost gain obtained for all the peers while PDB-C means the cost gain obtained only for those peers participating in collaborative caching. The figures show that the cost gain increases along with K. In any cases, the cost gain of CoCache is at least half of that obtained under ideal environment. When it is impossible to store the required data locally, CoCache always outerperforms PDB-C. Even all query results can be cached, CoCache is still a little better than PDB-C, since only raw data are cached in PeerDB.

5.2 Experiments in a Real P2P Environment

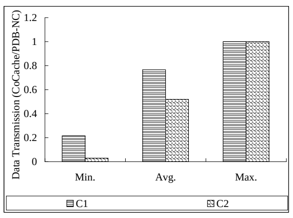
DBLP data set is transformed from DBLP XML records[6], in which the total number of tuples are more than 600,000, and the corresponding storage space is more than 200MB. The data set is partitioned and assigned to the peers. Furthermore, the quries are generated by a generator. Totally 144 queries are generated to be used in experiments, in which the number of joins varies from zero to five. The details of the data set partition and query generation are introduced in [9].

The DBLP data set is tested in a LAN environment with 40 peers, each of which is a PC with Pentium 1.4 GHz processor and 128MB RAM. The peers are divided into four groups. Within each group, ten peers are connected with one hub, and the hubs are connected by campus network with each other. CoCache is developed using Java, and running under Microsoft Windows 2000 Workstation. IBM DB2 UDB 7.1 is used as database engine. In experiments on DBLP data set, two schemes of block sizes are tested. In C1 scheme, each peer contributes a large storage space for caching (512KB per query), while in C2 scheme, only a small storage space is devoted for caching on each peer (128KB per query). Furthermore, the scheme of PDB-NC is used as baseline.

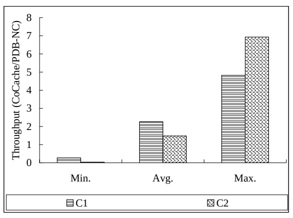
[6] `http://dblp.uni-trier.de/xml/`



(a) Response time: summarization.



(b) Volume of data trnasfered: summarization.



(c) Throughput: summariation.

Fig. 5. Experiments on DBLP dataset in a real P2P environment: CoCache *vs.* PDB-NC.

The response time, volume of data transfered, and throughput for 144 queries are recorded and shown with their summaries in Figure 5. It is interesting that the volume of data transfered in C1 scheme is more than that in C2 scheme. This is because that in C1 scheme, more cachess are established, which presumes more overhead and cache-to-cache data transfer. However, it is shown that the response time of C1 is better than C2, since cache-to-cache data transmission is usually cheap. Even in C2 scheme, the performance is not far worse than that in C1 scheme. The result is quite consistent with that obtained in simulation. It can be concluded that by collaborating, few contribution on each peer can gain much improvement on performance. The throughput of C2 scheme does not win PDB-NC scheme much. Since only part of data with maximum cost can be stored in caches, requesters still need to obtain data from some data sources that are not very far away.

6 Related Work

There are several popular peer-to-peer platforms that support key or identifier based search, such as Chord [13], CAN [10], Pastry [11], and BestPeer [7]. Caching is supported by some such platforms [13, 10]. However, the key-based caching scheme is usually too coarse to support complex query processing.

Providing query processing functions in P2P systems is a hot research topic. Several prototype systems are developed. They have different focuses, including interactive query processing (PeerDB [8]), DHT-based query execution (PIER [2]), schema mapping based query rewriting and processing (PIAZZA [1]), and data mapping based query processing (Hyperion [6]). As in PIER, we assume that schema information can be obtained in advance by requesters [2]. However, CoCache has a different focus for providing a mechanism for sharing data, processing and partial results among nearby peers with common subqueries.

Caching in P2P systems is extensively studied for its advantages for performance improvement. SQUIRREL [3] and BuddyWeb [14] are two prototype systems that allows each peer to share its Web cache, so that it can be shared by other peers in the same community. PeerOLAP, another cache sharing P2P system is designed for online analytical query processing [4]. The data are partitioned into aligned chunks which become the objects to be cached. Different caching strategies are studied, and the efficiency is shown under the setting of a self-configurable P2P network. The work closest to our research is range query result caching [12]. The ranges in a one-dimensional space is mapped to a two-dimensional CAN. Efficient search algorithm is developed to find the cached ranges. Above methods have the same characteristic in that each peer caches data blindly with other peers, while CoCache uses collaborative caching based on information collected via CON. Different from the CAN in range query caching, that is used to index data, CON is used to index combinations of dimensions and peers.

7 Conclusions and Future Work

We propose a query processing framework called CoCache in this paper, which is designed to utilize the limited storage devoted by various peers. The collaborative caching scheme adopted is a natural extension of key-based caching. With the help of DHT-based coordinator overlay network, peers can obtain summary information of the related queries and providers' information. Thus, collaborative caching can serve the queries more efficiently than existing caching schemes in P2P systems. Furthermore, the coordinator overlay network enables the cost-based optimization with low maintenance overhead. Experimental results show that CoCache is especially effective when each peer has limited storage for caching, which is a great challenge in real-life applications.

The authors would like to extend the research work in several directions. First, more query optimization techniques are to be studied based on the framework of CoCache. Furthermore, consistency maintenance and cache replacement strategies in CoCache system are to be explored. Last but not the least, we are working on more complex caching schemes for heterogeneous peers who do not share the common schema.

Acknowledgement

This work is supported by Infocomm Development Authority of Singapore (IDA). The authors would like to thank Jianfeng Yin, Wenyuan Cai and Tian Xia for their help in implementing the previous version of CoCache, called PeerView, and Dr. Wee Siong Ng for providing the source code of PeerDB.

References

1. A. Halevy, Z. Ives, P. Monk, and I. Tatarinov. Piazza: Data management infrastructure for semantic web applications. In *Proceedings of the 12th World-Wide Web Conference (WWW'2003)*, 2003.

2. R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the internet with pier. In *Proceedings of the 29th International Conference on Very Large Databases (VLDB'2003)*, 2003.
3. S. Iyera, A. Rowstron, and P. Druschel. Squirrel: A decentralized, peer-to-peer web cache. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC'2002)*, 2002.
4. P. Kalnis, W. S. Ng, B. C. Ooi, D. Papadias, and K.-L. Tan. An adaptive peer-to-peer network for distributed caching of olap results. In *Proceedings of ACM SIGMOD 2002 International Conference on Management of Data (SIGMOD'2002)*, 2002.
5. A. Kementsietsidis, M. Arenas, and R. J. Miller. Managing data mappings in the hyperion project. In *Proceeding of IEEE Conference on Data Engineering (ICDE'2003)*, 2003.
6. A. Kementsietsidis, M. Arenas, and R. J. Miller. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In *Proceedings of ACM SIGMOD 2003 International Conference on Management of Data (SIGMOD'2003)*, 2003.
7. W. S. Ng, B. C. Ooi, and K.-L. Tan. Bestpeer: A self-configurable peer-to-peer system. In *Proceedings of IEEE Conference on Data Engineering (ICDE'2001)*. IEEE Press, 2002.
8. W. S. Ng, B. C. Ooi, K.-L. Tan, and A. Zhou. Peerdb: A p2p-based system for distributed data sharing. In *Proceedings of IEEE Conference on Data Engineering (ICDE'2003)*. IEEE Press, 2003.
9. W. Qian, L. Xu, S. Zhou, and A. Zhou. Peerview: View selection for query processing in p2p systems. Technical report, Dept. of Computer Science and Engineering, Fudan Univeristy, Available at `http://www.cs.fudan.edu.cn/wpl/memeber/wnqian/`, 2004.
10. S. Ratnasamy, P. Francis, K. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'2001)*, 2001.
11. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware'2001)*, pages 329–350, 2001.
12. O. Sahin, A. Gupta, D. Agrawal, and A. E. Abbadi. A peer-to-peer framework for caching range queries. In *Proceedings of the 20th IEEE International Conference on Data Engineering (ICDE'2004)*, 2004.
13. 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 the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'2001)*, pages 149–160. ACM Press, 2001.
14. X. Wang, W. S. Ng, B. C. Ooi, K.-L. Tan, and A. Zhou. Buddyweb: A p2p-based collaborative web caching system. In *Proceedings of Peer-to-Peer Computing Workshop (Networking 2002)*. IEEE Press, 2002.