Scalable Distributed Aggregate Computations through Collaboration

Leonidas Galanis [1], David J. DeWitt [2]

[1] Oracle USA, 500 Oracle Pkwy, Redwood Shores, CA 94065, USA
leonidas.galanis@oracle.com
[2] University of Wisconsin – Madison, 1210 W Dayton St, Madison, WI 53706, USA
dewitt@cs.wisc.edu

Abstract. Computing aggregates over distributed data sets constitutes an interesting class of distributed queries. Recent advances in peer-to-peer discovery of data sources and query processing techniques have made such queries feasible and potentially more frequent. The concurrent execution of multiple and often identical distributed aggregate queries can place a high burden on the data sources. This paper identifies the scalability bottlenecks that can arise in large peer-to-peer networks from the execution of large numbers of aggregate computations and proposes a solution. In our approach peers are assigned the role of aggregate computation maintainers, which leads to a substantial decrease in requests to the data sources and also avoids duplicate computation by the sites that submit identical aggregate queries. Moreover, a framework is presented that facilitates the collaboration of peers in maintaining aggregate query results. Experimental evaluation of our design demonstrates that it achieves very good performance and scales to thousands of peers.

1. Introduction

Peer-to-Peer (P2P) computing has gained both scientific and social importance recently due to the success of systems such as Freenet [3], Gnutella [6] and Napster [12]. Harnessing P2P technology has the potential to produce systems that combine good scalability with minimal infrastructure cost. P2P systems are designed to start out small and seamlessly evolve to very large distributed systems with thousands of participants. The P2P computing paradigm has inspired many research projects to focus on a large variety of open problems. [15], [16], [22] and [25] provide the basis for low-level location services, otherwise known as Distributed Hash Tables (DHTs). [2], [11] and [18] illustrate how to use DHTs to build distributed file systems. [8] and [5] attempt to process complex queries in large P2P systems. The result has been the emergence of a concept known as data centric networking ([7] and [21]). With the advent of P2P systems, finding interesting data efficiently has become a major focus of research in the networking community.

Aggregate computations on data from distributed data sources constitute an important class of queries. P2P tools promise to make such queries feasible and therefore more frequent. If an aggregate computation is interesting to *multiple* peers in the network, the data sources participating in the computation can expect to receive the same query *multiple* times. Thus a new problem arises: The *many-to-many* query

problem (M2M), which places scalability limits on query processing in P2P systems. To better illustrate the M2M problem, consider an application that brings together commodity traders from around the world in a large P2P commodity trading system without a centralized infrastructure. Traders post their sale and bid prices based on information obtained by querying each other. Typically, participants determine their asking price or bid after consulting the maximum bid and minimum sale price for a commodity across all traders. In the absence of a central server, a trader has to query all other traders in order to determine the maximum bid and the minimum sale price. Thus, if m sellers and n bidders are trading on one particular commodity, each trader has to answer $m+n-1$ *identical* queries. Furthermore, the total number of messages that must be exchanged among the participants and the total number of queries executed in the system is $(m+n)\cdot(m+n-1)$. Using a central server instead of a P2P system, each participant would require only 2 queries to retrieve the minimum sale price and the maximum bid, which translates to $2\cdot(m+n)$ messages for $(m+n)$ queries and another $(m+n)$ messages for the $(m+n)$ updates of bids and sale prices. Consequently, any P2P system would still not scale well for this type of application due to the high message traffic and query processing load. On the other hand, a central server can scale by adding additional hardware. The challenge is to make a P2P infrastructure scale gracefully under the M2M query scenario by leveraging existing resources.

This work presents a framework for efficiently processing *many-to-many* aggregate queries over large P2P networks in a scalable fashion. Our approach requires the same number of queries and messages as would be required with a centralized system by leveraging DHT technology and catalog services ([5]). The contributions of this work can be summarized as follows:

- A method for defining the special handling of aggregate computations that follow the *many-to-many* query pattern.
- An efficient query processing strategy that leverages existing P2P technology and allows for scalable processing of *many-to-many* aggregate queries.

Experimental validation of our approach demonstrates its scalability potential and, at the same time, shows the adverse impact of the M2M query problem on P2P applications. We believe that our design opens up new possibilities for novel distributed applications, since distributed aggregation is going to be increasingly essential for efficiently surveying large amounts of distributed data.

The paper is organized as follows: Section 2 outlines the overall system architecture. Section 3 delves into the detailed design of the distributed aggregate computation layer. Section 4 presents the results of the experiments. The paper ends with related work (Section 5) and concluding remarks (Section 6).

2. System Architecture

The software stack on *each* peer consists of four layers: 1) a **Distributed Hash Table** layer (DHT), 2) the **Catalog Service** (CS) layer, 3) the **Aggregate Computation** (ACL) layer, and 4) a query engine with access to the local data. The design does not dictate a specific data model or query language, but all examples will assume XML [24] data sources and XPath [23] queries.

The **DHT** layer is based on existing technology ([15], [16], [22] and [25]). Its purpose is to support the efficient and scalable location of *keys* or *object identifiers* used by the higher-level layers of the system. In essence, DHTs are fully distributed hash tables that employ protocols for efficiently directing requests for specific *keys* to the nodes in the network that are responsible for those *keys.*

The **Catalog Service** (CS) layer is the data discovery tool. Each node employs a CS that, when given an arbitrary Xpath query, locates the relevant data sources. Subsequently, the query only needs to be submitted to a subset of the peers in the distributed system. To provide this functionality, the CS requires that each data source provide a summary of its data in a special form when it joins the distributed system. The CS is based on the framework presented in [5].

The **Aggregate Computation Layer** (ACL) maintains the registered aggregate queries that have been submitted by the various nodes in the P2P network. The ACL is the focus of this paper and its detailed design is presented in Section 3.

3. Distributed Aggregation

Any node can establish special handling of aggregate queries by requesting the creation of an *aggregation point* (AP) if it discovers that it frequently needs to contact a large number of nodes in order to compute an aggregate or if it becomes overwhelmed with large numbers of identical requests that are part of an aggregation computation. The ACL creates an aggregation point when provided with an *activation record* (AR) that contains the following fields:

Aggregate Function: This field determines the aggregate function (*average*, *minimum*, *maximum* etc) that is applied by the peer responsible for the aggregation computation maintenance to the incoming data.

Target Data: This field contains a query that defines the data needed for computing the aggregation. The target element or attribute of the query determines the catalog service peer ([5]) that is the peer that will create and maintain the AP.

Scope: The scope can be either *global* or *local*. Global scope means that the aggregate function should be computed over all peers with relevant data, while local scope computes one value for each peer.

Group By: The *group by* field refines the aggregation by defining on or more aggregate groups (similar to the SQL "group by" construct). Each group is assigned to a node that maintains the aggregate computation for the specific group.

Table 1 Activation Record for the commodity traders example

Aggregate	MAXIMUM	**Scope**	GLOBAL
Target Data	//bidder/item/current_bid	**Group By**	//bidder/item/@item_id

An aggregation point corresponds to one or more groups depending on the *group_by* field. Each group is assigned to the *Aggregation Point Host* (APH) that is a node in the P2P network. The APH is selected among the peers in the P2P system based on the activation record. A DHT *key* is computed using all the fields of the AR. This key determines which peer will currently serve as the APH and thus assume responsibility for the aggregate computation.

To illustrate the creation of an activation record, consider again the example of commodity traders. Suppose bidders are peers that post their bids online as XML

documents and update them as they trade. The current price a bidder is willing to pay for a commodity item *i* can be accessed using the path p_{bid} = //bidder/item [@item_id = *i*]/current_bid. Sellers naturally want to find the bidder with the maximum bid. Thus, if there are a large number of sellers of commodity *i*, and a large number of bidders, an aggregation point is needed to avoid M2M and make trading more efficient. Table 1 shows what the required activation record looks like. The aggregate function is the *maximum* and the scope is global. The target path is p_{target} = //bidder/item/current_bid. Hence, current_bid is the catalog key that determines the DHT key for catalog information and so the peer that maintains the AR. The group by field is the path $p_{group\_by}(x)$ = //bidder/item/@item_id = x. The p_{group_by} field essentially assigns each traded item to a different DHT key.

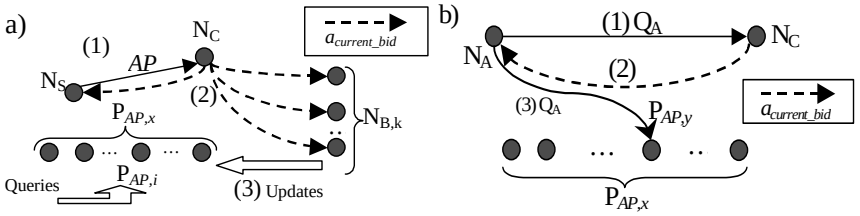

Fig. 1 a) Installation of an Aggregation Point b) Redirection of query Q_A to node $P_{AP,y}$

Under normal operation the catalog service would use *current_bid* to identify the relevant data sources. However, if an *AP* is installed, the mode of query processing changes. Each node uses *Aggregation Key Map* (AKM), which acts as a subscription system that associates catalog keys with their activation records. Fig. 1a outlines the process based on the commodity traders example. Suppose that peer N_S, which hosts sellers of commodity *i* decides to request the creation of the aggregation point *AP* that is defined by the activation record described in Table 1. The request will be forwarded to the peer N_C that holds catalog information for *current_bid* (step 1). N_C will generate the association $a_{current\_bid}$ = *current_bid* $\rightarrow$ $MAX_{GLOBAL}(p_{target}, p_{group\_by}(x))$ and insert it in the AKM. Note that multiple associations for a given key can exist in the AKM and they are selected based on the actual query. Then, N_C forwards $a_{current\_bid}$ to all peers $N_{B,k}$ that host bidders and to N_S (step 2). N_C knows about all such peers since it is hosting catalog information for *current_bid*. Upon receipt of $a_{current\_bid}$ each $N_{B,k}$ is expected to send updates about the current bid for all traded commodities to specific other peers that form the *set of aggregation point hosts* (**APH**) (step 3). The DHT layer, using the information in $a_{current\_bid}$, can uniquely determine each peer in **APH**. For example, for commodity item *i*, the APH $P_{AP,i}$ is determined by hashing the list (MAX, GLOBAL, p_{target}, p_{group_by}(i)) to retrieve a DHT key k_i. Thus $P_{AP,i}$ becomes the maintainer of the requested aggregation for commodity *i*. Aggregation responsibility is tied to the key k_i and not to the peer $P_{AP,i}$. This way if $P_{AP,i}$ leaves, the DHT ensures that k_i points to a different peer.

The question that remains is how peers other than N_S find out about the new aggregation points. This turns out to be straightforward since N_C is the *designated* node for all inquiries regarding *current_bid* (Fig. 1b). Thus, any other node N_A which receives a query such as Q_A = MAX(//bidder/item [@item_id = *y*]/current_bid) will request catalog information from N_C (step 1). N_C will then provide N_A with the

association $a_{current\_bid}$ (step 2). Having this information N_A can identify $P_{AP,y}$ as the peer to visit in order to obtain the answer to Q_A (step 3). Furthermore, N_A locally caches $a_{current\_bid}$ and thus only needs to contact N_C once.

The peers in **APH** can become heavily loaded if they are assigned very popular commodities. In this case load balancing is necessary. We have devised a method that deals with high request rates for both queries and updates. Due to space limitations we do not present the load balancing mechanisms. (see [4] for details).

4. Experimental Evaluation

The experiments are based on a *Distributed Commodity Trading* (DCT) scenario derived from [14]. The potential for eliminating the centralized auctioneer and the "fixed time trading rounds" ([14]) is the motivation to realize DCT. In our scenario traders are users of a large P2P network and buy and sell commodities. Each trader interchangeably follows a *seller session* or a *bidder session.* During a seller session the trader queries for the bidder with the maximum bid for one of its commodities and if successful proceeds with the sale. During a bidder session the trader tries to determine a reasonable bid for a commodity by querying the selling prices in the network. Thus, traders constantly query each other, which leads to the manifestation of the M2M problem. Complete description of the scenario is available in [4].

Table 2 Mean measured CPU and Disk service times of a peer running

Query Load	**XPath**	**Catalog Lookups**
CPU (ms ± ms)	540 ± 57	34 ± 11
Disk (ms ± ms)	1800 ± 324	75 ± 11

4.1. Experimental Methodology

We use simulation and a prototype system (Sect. 4.4) to evaluate our architecture. Simulations follow a two-step methodology that combines system measurements with simulation. First measurements were taken from a system that consists of an XPath query engine, a catalog layer and an aggregation layer (Section 2). This system was loaded with both trading data and catalog information. Then workloads of XPath queries, catalog information lookups and aggregate computations were executed and measurements were collected (Table 2), which yields the nominal peer performance ([4]). The XMark benchmark data generator [20] was used to generate about 1GB of data for auctions that have a structure similar to our trading scenario data. Catalog information lookups were measured using 256 MB of catalog data as described in [5]. The system used for measurements was a 2.4 GHz Pentium 4 PC running Linux with an IDE Hard Disk and 512 MB of RAM. To obtain variance across the peers during the simulations in the P2P network the nominal performance is multiplied by a factor that is uniformly distributed between 0.8 and 1.2. The second step involved building a discrete event simulation model using CSIM [1] consisting of nodes interconnected with a DHT. The nodes in the P2P network are modeled as single CPU, single disk workstations using the measurements from the first step. The prototype system experiments used our departmental cluster with 40 nodes similar to the one used for single node measurements.

4.2. Simulation Setup

Simulations for two peer-to-peer and one central server system were implemented. This section describes their characteristics and their basic differences. Both peer-to-peer systems simulate a DHT that is a generic version of Chord [22]. Following the observation made in [8] we did not use a detailed network model, opting instead for a simple delay model where network delays are exponentially distributed with a mean of 50ms. We assumed that network bandwidth was not a limiting factor since only a small amount of data is transferred in each network message. A catalog service as presented [5] is also present on both P2P systems and is used by traders to locate other traders.

The impact of network volatility on a peer-to-peer system depends on the specific DHT implementation. Therefore our experiments examine stable peer-to-peer systems in order to obtain results that are independent of the underlying DHT implementation and demonstrate the raw impact of the Aggregation Layer framework in improving performance.

The first P2P version that utilizes the aggregation layer has two variants: **AL** (Aggregation Layer) and **LBAL**. The difference is that the second variant employs load balancing (**LB**) [4]. The second P2P setup does not have an aggregation layer and comes in two variants **GC** (General Catalog) and **GCI**. **GC** utilizes the catalog service to discover traders, but directs XPath queries to the traders' peers in order to collect data values and compute the maximum bid and the minimum selling price. These XPath queries are issued *simultaneously* to all traders and the aggregate is computed after the results are retrieved. This setup suffers under the M2M query problem. The **GCI** variant utilizes a local index that makes XPath queries for retrieving bids and sale prices as fast as the aggregate computations and catalog information lookups in the variants **AL** and **LBAL**, and is used for a fairer comparison to **AL** and **LBAL**.

The central server system is intended as a reference point for evaluating the performance of the peer-to-peer variants. It consists of an ideal cluster of with as many nodes as the corresponding P2P system. The traders access their accounts from their workstations connected to the Internet, thus experiencing *network delay* for each query and update request. The central server system also comes in two variants: **CS** and **CSI**. The **CSI** variant employs the same fast local index as **GCI**.

4.3. Performance Results

The goals of this section are to determine the extent to which the aggregation layer improves performance and identify those cases where load balancing is required. A very important parameter in all configurations is the number of unique traded items *T*, which affects both P2P systems similarly. The smaller *T* is, the larger, on average, is the number of traders for a particular item. The consequence for each node in **GC** and **GCI**, and for each aggregation point in **AL** and **LBAL** is more requests on average.

The first series of experiments involved peer-to-peer networks from 100 to 100,000 nodes. The numbers of bidders and sellers in the P2P network are approximately equal and each node hosts one trader. The number of items assigned to each trader is uniformly distributed between 5 and 15. The popularity of the traded items follows the 80/20 rule (a.k.a. Pareto's principle) observed in many real world settings: 20% of

the items are chosen by traders 80% of the time. The centralized versions of the system have exactly the same trader and commodity distributions. The configuration with 10,000 nodes is presented first to demonstrate our key findings when varying the number of unique traded items (commodities).

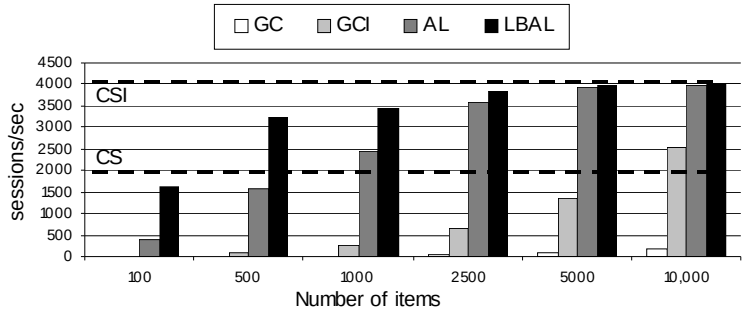

Fig. 2 Seller session throughput for 10,000 nodes

The throughput of seller sessions (Fig. 2) as a function of the different number of items traded is the first set of results presented (relative error at most 1% with 95% confidence). Bidder sessions follow a similar trend [4]. The throughput for **CS** and **CSI** is insensitive to the number of unique traded items (dashed lines). For both P2P systems a small variety of items has an adverse effect as expected. Nevertheless, the impact of a small number of items is more significant on **GC** and **GCI** than on **AL**. For 1000 items, the throughput of **AL** is 9.6 times better than **GCI** while for 10,000 items **AL** is 1.5 times better. At the same time as the number of items decreases, the need for load balancing becomes apparent: for over 2500 items **AL** and **LBAL** have similar performance. In the case of 100 items, however, **LBAL** is over 4 times better than **AL**.

Table 3 Average trader session durations for 10,000 nodes

#items	***CS***	***CSI***	***GC***	***GCI***	***AL***	***LBAL***
100	3.13 s	1.48 s	N/A	N/A	15.60 s	2.04 s
500			N/A	47.1 s	4.09 s	1.84 s
1000			236.1 s	22.9 s	2.61 s	1.79 s
2500			153.4 s	9.2 s	1.73 s	1.58 s
5000			72.0 s	4.41 s	1.56 s	1.52 s
10,000			30.2 s	2.24 s	1.51 s	1.49 s

Table 3 shows the average duration of traders' sessions. Session durations are indicative of the usability of each configuration. Short sessions imply short query response times, which in turn imply more accurate values for minimum sale price and maximum bid. The high session durations for **GC** and **GCI** show that they are not usable. As expected, **LBAL** does a very good job by keeping the duration of sessions below 2 sec. Without load balancing, the average session duration for **AL** with 100 items climbs to 15.6 sec. The throughput and response time numbers show that **GC** is virtually unusable. **GCI** is viable in the 5000 and 10,000 item setups but still lags far behind both **AL** and **LBAL** in terms of system throughput The absence of data for **GC** (for 100 and 500 items) and **GCI** (for 100 items) was the result of event backlogs

in the simulator, which led to high memory image sizes, forcing the simulations to abort.

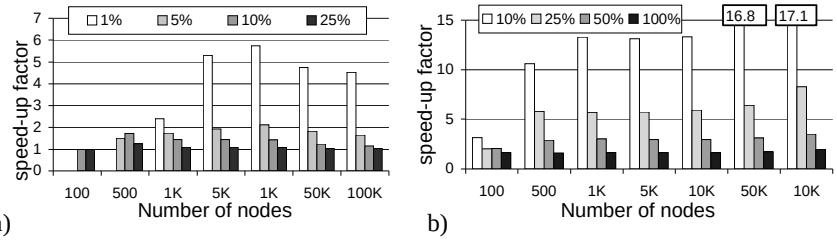

Fig. 3 a) Speed-up of LBAL over AL. b) Speed-up of LBAL over GCI

While we have obtained results for a variety of network configurations, due to space limitations scalability results are summarized in Fig. 3a and Fig. 3b. Fig. 3a shows the speed-up of **LBAL** over **AL** in trader session throughput. The percentages in the legend denote the number of items in each network as a percentage of the total number of nodes. The graph, in essence, shows which combinations of network size and traded items make load balancing a necessity. For instance, in the 100 node network load balancing is not necessary. In the 500 node network the load balancing benefits are observable. In the larger networks load balancing of aggregate computations becomes a necessity as demonstrated by the achieved speed-up. Let *r = number of unique traded items / number of nodes*. In large networks (number of nodes > 1000) the smaller *r* is, the larger is the speed-up of **LBAL** over **AL** due to load balancing. These results suggest that if $r \geq 25\%$ the speed-up achieved using load balancing is not significant.

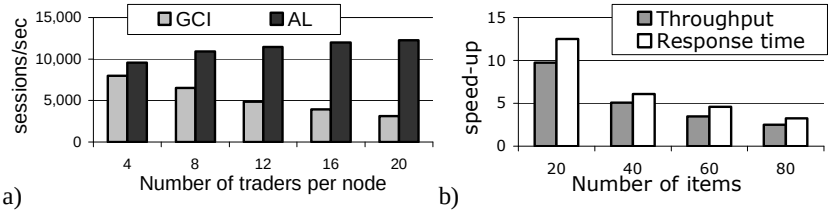

Fig 4a) Combined trader throughput for 10,000 nodes, 50,000 unique items and varying number of traders b) Speed-up of PAL over PGC

Fig. 3b shows the trader session throughput speed-up of **LBAL** over **GCI**. The percentages have the same meaning as in Fig. 3a. The percentages are now larger as **GCI** would not work on large networks (>10,000) with a small variety of traded items. **GCI** appears somewhat usable when the number of traded items is large (>50% of total number of nodes): With $r = 50\%$ **GCI** is about two to four times slower than the systems with the aggregation layer (**AL**, **LBAL**). For $r = 10\%$ or 25% **GCI** is clearly not scalable, which is indicated by the increasing speed-up.

Fig 4a shows how the combined session throughput varies with the number of traders per node in **GCI** and **AL**. The network has 10,000 nodes and 50,000 unique items, which favors **GCI**. While **AL** starts out slightly better than **GCI** it becomes 4 times better (20 traders).

4.4. Prototype Experiments

We implemented a prototype system to confirm the simulation results. The system is written in Java and uses Pastry [16] as the DHT and Berkeley DB XML as the storage and query engine layer, which is accessed through the Java native interface. For the experiment we used 40 machines from our departmental Linux cluster. A trader with 15 commodities, on average, is emulated on each machine. The non-aggregation layer configuration **PGC** corresponds to **GCI** in our simulations and the aggregation layer configuration **PAL** corresponds to **AL**. Due to space constraints we only present Fig 4b that shows the speedup of trader sessions of **PAL** over **PGC** achieved in the system with 40 nodes and a varying number of unique commodities (items). The aggregation layer achieves significant speed-up in a working prototype system and confirms the simulation results.

5. Related Work

Related work in P2P architectures has been mentioned in Sect. 1. Here we present work more closely related to distributed aggregation, which is a relatively new subject in the context of P2P systems. Willow [16] organizes nodes in a single tree. Aggregate computations percolate automatically up the tree whenever there are data changes or new aggregate queries are installed. However, these updates are not instantaneous and converge eventually. The Aggregation Layer presented here follows a best-effort approach by having a flat structure. SOMO [25] similar to our approach layers on top of a DHT and, like Willow, organizes the aggregate computations in a tree. SOMO has a generic gathering procedure that can be programmed to perform aggregate computation. This procedure is invoked periodically, in contrast to the updates and requests to the aggregation points of the Aggregation Layer, which are on demand. The aforementioned projects are a sample of many similar ongoing projects addressing distributed aggregation in P2P systems.

SCRIBE [19] is an application layer multicast publish/subscribe system that uses a PASTRY [16] to define rendezvous points for managing group communication on a specific topic. It uses topic identifiers to assign topics to peers similarly to our use of catalog keys to assign aggregation points to peers. A basic difference between our approach and SCRIBE is that aggregation point hosts do not implement publish/subscribe functionality, and are thus much simpler. Their purpose is to passively collect data and maintain an always up-to-date aggregate. Distributed aggregation methods in [10] presents distributed aggregate computations using gossip-based protocols in P2P networks. Its focus, however, is on how quickly aggregate computations converge to the actual value and not how to facilitate large volumes of aggregate queries over distributed data sets.

6. Conclusions

In this paper we presented the case for the many-to-many query problem that is bound to be a concern in very large distributed systems where queries require data from multiple data sources. Using existing technology we developed a framework that can

solve this problem for a broad class of important queries by harnessing the resources of the peers in the distributed system. Our experimental evaluation using both simulations and a real working prototype shows how severe the M2M problem can be and how our architecture efficiently solves it in a P2P environment.

References

1. CSIM Development Toolkit for Simulation and Modeling. http://www.mesquite.com.
2. F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica. Wide-area cooperative storage with CFS. SOSP 2001.
3. FreeNet. http://www.freenetproject.org
4. L. Galanis. Towards a Data-Centric Internet. Ph.D. Thesis. 2004 Univ. of Wisconsin.
5. L. Galanis, Y. Wang, S. R. Jeffery, D. J. DeWitt. Locating Data Sources in Large Distributed Systems. VLDB 2003
6. Gnutella Resources. http://gnutella.wego.com/
7. J. M. Hellerstein. Toward Network Data Independence. SIGMOD Record, Vol. 32, No. 3, Sept. 2003.
8. R. Huebsch, J. M. Hellerstein, N. Langam, B. T. Loo, S. Shenker, I. Stoica. Querying the Internet with PIER. VLDB 2003
9. Kazaa. http://www.kazaa.com.
10. D. Kempe, A. Dobra, J. Gehrke: Gossip-Based Computation of Aggregate Information. FOCS 2003
11. J. Kubiatowicz et al. OceanStore: An Architecture for Global-Scale Persistent Storage. In Proc. ASPLOS 2000.
12. Napster. http://www.napster.com
13. Network Time Protocol (NTP). http://www.ntp.org.
14. E. Ogston, S. Vassiliadis. A Peer-to-Peer Agent Auction. AAMAS'02.
15. S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker. A Scalable Content-Addressable Network. in Proc. of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications.
16. R. van Renesse and A. Bozdog. Willow: DHT, Aggregation, and Publish/Subscribe in One Protocol. International Workshop on Peer-to-Peer Systems (IPTPS) 2004.
17. A. Rowstron, P. Druschel, Pastry. Scalable, distributed object location and routing for large-scale peer-to-peer systems. IFIP/ACM Intl. Conference on Distributed Systems Platforms.
18. A. Rowstron, P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. SOSP 2001.
19. A. Rowstron, A. Kermarrec, M. Castro, P. Druschel. Scribe: The design of a large-scale event notification infrastructure. Intl. Workshop on Networked Group Communication 2001.
20. A. Schmidt, F. Waas, M. Kersten, M. J. Carey, I. Manolescu, R. Busse. XMark. A Benchmark for XML Data Management. VLDB 2002.
21. S. Shenker. The Data-Centric Revolution in Networking. Keynote VLDB 2003.
22. I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. SIGCOMM 2001.
23. XML Path Language (XPath) 2.0 http://www.w3.org/TR/xpath20/
24. XML Extensible Markup Language. http://www.w3.org/XML.
25. Z. Zhang, S. –M. Shi and J. Zhu. SOMO: Self-Organized Metadata Overlay for Resource Management in P2P DHT. International Workshop on Peer-to-Peer Systems (IPTPS) 2003.
26. B. Y. Zhao, J. Kubiatowicz, A. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. UCB Tech. Report UCB/CSD-01-1141.