

Adlib: A Self-Tuning Index for Dynamic Peer-to-Peer Systems

Prasanna Ganesan

Qixiang Sun

Hector Garcia-Molina

Stanford University

{*prasannag, qsun, hector*}@cs.stanford.edu

Abstract

Peer-to-peer (P2P) systems enable queries over a large database horizontally partitioned across a dynamic set of nodes. We devise a self-tuning index for such systems that can trade off index maintenance cost against query efficiency, in order to optimize the overall system cost. The index, Adlib, dynamically adapts itself to operate at the optimal trade-off point, even as the optimal configuration changes with nodes joining and leaving the system. We use experiments on realistic workloads to demonstrate that Adlib can reduce the overall system cost by a factor of four.

1 Introduction

A peer-to-peer (P2P) system consists of a large, dynamic set of *nodes* spread over a wide-area network. The system's scale and dynamism dictate that inter-node communication takes place on a low-degree *overlay network*, with only neighboring nodes allowed to exchange messages directly. We focus on systems where nodes contain "related" data, i.e., we can view each node as "owning" some tuples in a global, horizontally partitioned relation R .

A fundamental operation in such a P2P system is a selection query that requires all, or some, tuples that have a "key" attribute A equal to a given value. Such a query may be answered efficiently using a distributed index that maps each possible value of attribute A to the set of nodes that contain tuples with that value. Different indexes offer different trade-offs between the index maintenance cost and the querying benefit obtained.

To illustrate, consider two common indexing structures proposed in past literature. In Gnutella [1], each node constructs a local "index" over its own tuples. Therefore, a query for all tuples with a given attribute value needs to be sent to all nodes in the system, resulting in a high query cost. On the other hand, index maintenance is free. When a new node joins, or an existing node leaves, the indexes of other nodes are completely unaffected.

A second approach is to construct a distributed global index, partitioned across nodes by attribute values; for each value, some one node in the system is designated to manage and store the entire index entry – a list of all nodes with tuples containing that value. The assignment of which nodes manage which values may be done in different ways, for example, by hash partitioning [6] or by range partitioning [2]. Such a global index offers efficient querying; a query for a given value is answered simply by contacting the node managing the index entry for that value. On the flip side, every time a node N joins or leaves the system, all nodes that manage index entries for values owned by N need to be notified, in order to update the index appropriately.

Many P2P systems are characterized by query rates that are comparable to the rates at which nodes join and leave [5, 7]. In such systems, the index-maintenance cost of the global-index approach can be very high, and dwarf the benefit obtained for queries. Worse still, as the systems scale up in terms of the amount of data owned by each node, the index maintenance cost grows much faster than the cost of queries, rendering the global index expensive.

Our objective is to devise a self-tuning index that can dynamically trade off index-maintenance cost against the benefits obtained for queries, in order to minimize the total cost of index maintenance and query execution. We also desire organic scaling with the data volume per node, which we expect to grow rapidly over time. In this brief announcement, we describe some of the key ideas behind our solution, Adlib. More details can be found in [4].

2 A Two-Tier Approach

The key to Adlib's control of the query vs. maintenance cost trade-off lies in its two-tier structure. Nodes in Adlib are partitioned into k independent equal-sized *domains* where k is a tunable parameter controlled by Adlib. Nodes within each domain build a distributed "global" index *over the content stored in that domain*.

Queries We consider two different types of queries. A *partial-lookup* query requires a fixed number of results

matching the specified key. Such a query is answered by first executing the query within the domain in which it is posed. If the number of answers prove insufficient, the query is forwarded to more and more domains. (We may also imagine a staged partial-lookup query, where the user may press a “Get more results” button to get more and more results on demand.) A *total-lookup* query, on the other hand, requires all available answers, and thus the query needs to be executed in every domain.

For a fixed number of nodes n , it turns out that the cost of a total lookup increases almost linearly as the number of domains k increase. The cost of partial lookups is somewhat more complicated, as it depends on query selectivities, but for small domain sizes, it also increases with k .

Index Maintenance The Adlib index also needs to be maintained as nodes join and leave the system (or as data is updated). Note that the departure (and similarly, the arrival) of a node A has two separate effects: (a) the index entries that were stored by A disappear and need to be re-inserted, and (b) the index entries corresponding to the A 's *content* need to be eliminated from the system. Let us focus on (b). If the number of tuples at A is reasonably large, the corresponding index entries are going to be spread over a large number of nodes, limited by the total number of nodes in the domain. For small domain sizes, the most efficient way to achieve (b) is to simply broadcast the fact of A 's departure to all other nodes in the domain; the cost of this operation increases with the domain size.

Thus, for a fixed number of nodes, query costs increase with k while the maintenance cost decreases. The optimal value $k = k_I$, which minimizes the overall system cost (the sum of the query and maintenance costs), can be shown to be proportional to \sqrt{n} for total-lookup queries and proportional to n for partial lookups, under simple models of node lifetimes, query rates and data distribution.

We verified the above results experimentally using realistic data sets derived from real P2P system measurements. Operating Adlib at this optimal value provided at least a factor of four improvement in costs compared to operating at one of the end-points, i.e., using a global index or using Gnutella, showing the value of the two-tier architecture.

A real P2P system, however, does not have a fixed number of nodes. As n changes over time, so does k_I ; thus, we need Adlib to dynamically change the number of domains it uses to stay optimal. (Other system parameters such as query rates also change over time and affect k_I ; we could adapt to such changes too.)

3 Making Adlib Self-Tuning

The intuition behind Adlib's adaptation is to split an existing domain into two when the current number of domains k is less than the ideal value k_I for the current configura-

tion, and to merge two domains into one when k is greater than k_I . (Note that there is no central node to compute k_I and decide when and what to split or merge.) Such domain splitting and merging introduces three challenges:

The Overlay Problem: Nodes should not be required to abandon their existing overlay links and set up new links on a domain split, since re-linking can be expensive. Query routing and broadcast must still be efficient under splits.

The Re-indexing Problem: The splitting or merging of domains requires a corresponding splitting and merging of indexes. Such re-indexing can prove expensive and must be made as efficient as possible.

The Atomic-Split Problem: Domain splits and merges must not require all nodes in the domain to synchronize, since this may prove impossible in a dynamic P2P system. Queries must continue to succeed even when only a fraction of the nodes have split, while others have not.

In [4], we show how all these problems may be tackled by careful construction of the overlay network using a hierarchical technique introduced in [3]. We show that re-indexing requires each node to only communicate with one other node in expectation, and that queries can be answered successfully even when nodes have “inconsistent” views about what the domains are.

Adlib also adapts well to network heterogeneity, where different pairs of nodes experience different communication latencies. We can exploit Adlib's two-tier structure to ensure that nodes within a domain are also physically close, leading to low-cost index maintenance and efficient lookup of “local” content in preference to content that is further away. In summary, Adlib is a distributed indexing mechanism that is self-tuning and adapts very well to growing volumes of data and changing numbers of nodes.

References

- [1] Gnutella. Website <http://gnutella.wego.com>.
- [2] P. Ganesan, M. Bawa, and H. Garcia-Molina. Online balancing of range-partitioned data with applications to p2p systems. In *Proc. VLDB*, 2004.
- [3] P. Ganesan, K. Gummadi, and H. Garcia-Molina. Canon in g major: Designing DHTs with hierarchical structure. In *Proc. ICDCS*, 2004.
- [4] P. Ganesan, Q. Sun, and H. Garcia-Molina. Adlib: A self-tuning index for dynamic p2p systems. Technical report, Stanford University, 2004.
- [5] S. Saroiu, K. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. Multimedia Computing and Networking (MMCN)*, 2002.
- [6] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. ACM SIGCOMM 2001*, 2001.
- [7] B. Yang and H. Garcia-Molina. Comparing hybrid peer-to-peer systems. In *Proc. VLDB*, 2001.