Bit Zipper Rendezvous
Optimal Data Placement for General P2P Queries *

Wesley W. Terpstra, Stefan Behnel,
Ludger Fiege, Jussi Kangasharju, and Alejandro Buchmann

Darmstadt University of Technology (TUD),
Department of Computer Science,
D-64283 Darmstadt, Germany,
{terpstra,behnel,fiege}@gkec.tu-darmstadt.de,
jussi@tk.informatik.tu-darmstadt.de, buchmann@informatik.tu-darmstadt.de

Abstract. In many distributed applications, pairs of queries and values are evaluated by participating nodes. This includes keyword search for documents, selection queries on tuples, and publish-subscribe. These applications require that all values accepted by the query be evaluated. To carry out this evaluation we will present the peer-to-peer based Bit Zipper Rendezvous which partitions query-value pairs as opposed to values only. Even for problems that allow an efficient value-based partition, the Bit Zipper complements existing solutions with its generality. Where flooding to N nodes used to be the only fall-back, the Bit Zipper is a replacement needing only $O(\sqrt{N})$. For problems requiring that all pairs be evaluated, we will show that the Bit Zipper Rendezvous is optimal.

1 Introduction

Recently, peer-to-peer (P2P) applications have emerged as an important new class of applications. They provide many attractive properties, such as self-organization and healing, robustness, load balancing, and scalability. These properties make the P2P approach appealing for large distributed applications. P2P systems have been used or proposed for a number of applications such as keyword search for documents, file storage, and publish-subscribe. Now, if we take a broader view on these applications, we can see that many of them contain the rendezvous problem.

The *rendezvous problem* is where several parameters involved in an operation need to be present at the same computer for evaluation. These parameters typically come in pairs and their evaluation takes place at some node in the P2P system. As for the above examples, keywords are searched for in documents and notifications are tested against subscriptions.

We claim that a good data placement strategy can address the rendezvous problem in the above applications. *Data placement strategies* are rules which decide where to place parameter data on the participating nodes.

* This work was partially funded by the German National Science Foundation (DFG) as part of the Graduate Colleges "Enabling Technologies for E-Commerce" and "System Integration for Ubiquitous Computing".

Partitioning parameters by their value is a very efficient rule used for data placement. These *partitioning strategies* place each parameter in exactly one partition; this is the root of their efficiency. For evaluation to be possible, pairs must have their component parameters placed in the same partition. When all required evaluations can be performed, a partitioning strategy is correct. Unfortunately, one cannot always correctly partition by parameter value.

When an efficient, correct parameter partition cannot be found, another strategy must be used. Flooding is the traditional last resort. Naively, flooding applications place all parameters of one type on every participant. This assures a rendezvous with the locally stored parameters of the other type and allows evaluation of all pairs. However, it implies that participating nodes have linear load in the flooded parameter.

In this paper, we present a general-purpose solution with sub-linear load. It is specifically designed to address those situations where potentially all of the pairs of parameters might need to be evaluated. We also show that for rendezvous problems which require all pairs to be evaluated, our strategy is optimal.

The next section provides an overview of the rendezvous problem by presenting concrete examples of applications, and it overviews distributed hash tables (DHTs) as a rendezvous approach. Section 3 presents our general purpose approach, the Bit Zipper Rendezvous data placement strategy, followed by sections 4 and 5 which show its optimality for some important types of applications. We then discuss the implications of our research and prior art in section 6.

2 The Rendezvous Problem

When moved onto a peer-to-peer platform, a number of important applications can benefit from the P2P characteristics mentioned above. The central rendezvous problem is described and exemplified below, and DHTs are introduced as means to implement a rendezvous strategy.

2.1 The Essential Problem

Stated generally, nodes in the rendezvous problem play three roles. In one role, nodes inject queries; in the other, they inject data. The final role is that of an intermediate router. To execute queries, the queries must either be shipped to the data, as in classic database queries, or the data must be shipped to the queries, as in filter and sensor networks. The new option offered by peer-to-peer systems is that the data and queries can *both* be shipped to nodes playing the third role—a rendezvous point where data and queries meet.

The rest of the paper interprets operations and data homogeneously; only a single application-specific operation is assumed to be executed on pairs of data from the first two roles. The two roles are distinguished according to their anticipated dynamism in the peer-to-peer network. The more dynamic parameter is designated *right* and the lesser is *left*.

The principal rendezvous problem is then to intelligently place data items at nodes; this involves choosing rendezvous nodes, and routing to them.

2.2 Examples

Linear Scan. When evaluating database queries, data is matched against queries shipped to the database server. It is sometimes necessary to do a linear scan of a table, e.g., if no index is available. If the data is not located on a central database server, but distributed and replicated in a peer-to-peer system, all involved nodes have to execute the query, which effectively precludes partitioning.

In terms of the rendezvous problem, the tables typically comprise long-lived and rather slowly changing data, and thus are denoted by the left parameter. Queries are instantaneously and frequently injected into the system and thus constitute the right parameter.

Publish-Subscribe. Pub-sub is an interest-oriented communication model [4]. Nodes transmit—or publish—information which is to be received—or consumed—by others. Consumers receive information that matches an expression they have specified; this expression is called a subscription. Thus, the recipient group is self-selecting by virtue of their interests.

In a distributed publish-subscribe network, subscriptions are left parameters located at some nodes. Right parameters are notifications that are forwarded to consumers in case of matching subscriptions. Clearly, the notifications must be selected by subscriptions and thus rendezvous.

Subscriptions and notifications may sometimes be successfully partitioned. For instance, subscriptions to types of information exploit a categorization of published data. However, there are many cases where such partitioning fails.

Distributed Keyword Search. Keyword search involves searching for documents by specifying words they must contain. Here it is assumed that the documents are rarely modified compared to the frequency of the queries. The left parameters are the relatively static documents and the right parameter are the keyword queries. This problem is easily solved by a centralized system using inverted-indexes, but for P2P systems it remains unsolved. After shipping queries to rendezvous nodes, one might copy the publish-subscribe idea and store these queries to get notified about new documents in the future.

2.3 Rendezvous in Distributed Hash Tables

A popular means of addressing the rendezvous problem is distributed hash tables (DHTs) [3, 9, 13, 15, 17, 21]. DHTs form an abstraction to overlay networks that provides a key-based routing scheme. The recipient of a message is determined dynamically by a distributed routing algorithm. However, as opposed to traditional networking, both the delivery path and the destination node can change over time. This easily accommodates topology changes.

By determining (hash-)keys from the parameter values, DHTs provide an obvious lookup mechanism for rendezvous nodes. However, this is partitioning by parameter value, which is not always possible. Nevertheless, DHTs do provide a number of features that make them perfectly suitable as a substrate for our data placement strategy.

DHT routing schemes are based on keys, $\mathscr{K}$. Different DHTs use between 128 to 256 bits for their fixed length keys. For concreteness, we assume that there are 160 bits: 1 to 160. If $k \in \mathscr{K}$, then k_i is defined to be the i-th bit of k.

Every participating node has one randomly chosen identifier, $n \in \mathscr{K}$. $\mathscr{N} \subseteq \mathscr{K}$ is the set of all node keys. $N = |\mathscr{N}|$ is the number of participating nodes. The lookup mechanism is a distributed algorithm implementing the responsibility function $\pi : \mathscr{K} \to \mathscr{N}$. For each key $k \in \mathscr{K}$, π returns the eventual destination node $n = \pi(k)$ of a message sent to k.

DHTs are defined with respect to some distance metric. They route a key, k, progressively closer to $\pi(k)$ by decreasing distance under their routing metric. This implies that all nodes are responsible for their own key; $\pi(n) = n$. When a *Euclidean distance metric* is used in a DHT, $\pi^{-1}(n)$ will be an interval. This property of the responsibility function is required for our data placement strategy to remain performant. Therefore, Kademlia [9] may not be used as substrate.

A message transported by a *prefix routing* protocol resolves a destination's key one bit at a time for as long as possible. Most DHTs do this either implicitly or explicitly. Our data placement strategy is easy to implement and follows normal routing paths only if a DHT has this property. Unfortunately, CAN [13] does not. Tapestry is actually based on postfix routing, but the difference is negligible for our purpose.

Summarizing, we can say that DHTs provide a way of distributing responsibility for keys uniformly at random. The majority [3, 15, 17, 21] are based on a Euclidean metric and prefix routing. These DHTs are perfect candidates for the Bit Zipper Rendezvous.

3 Bit Zipper Rendezvous

The Bit Zipper approach partitions *pairs* of parameters and distributes them randomly. It places each parameter in a large, but sub-linear number of these partitions. This strategy is applicable even when there is no correct value-based partitioning scheme for parameters separately.

The next subsection presents a method to assign every parameter to a number of pair partitions. Building on a DHT, these parameters are mapped to nodes in section 3.2 and routed in section 3.3.

3.1 Partitioning Pairs

Applications involving the rendezvous problem evaluate pairs of parameters. For some applications, nearly all of the pairs might need to be evaluated. Simply partitioning parameters separately can lead to the component parameters of a pair being placed in separate partitions; this prevents evaluation.

However, partitioning *pairs* of left and right parameters does not prevent evaluation since all pairs remain intact. Each parameter is involved in many pairs and has to be placed in many pair partitions. These pair partitions are

identified by DHT keys, thus mapping every parameter to a large number of DHT keys.

Parameters are assigned a randomly chosen seed key. Taking the seeds of two parameters together, one will be able to determine the DHT key where their pair is placed. Naturally, parameters must be placed at all such possible keys.

The placement follows a zipper-like bit pattern that gives the Bit Zipper its name. It is illustrated in figure 1. The b part is an optimization that will be explained in section 3.4. In this pattern, every even bit in a right parameter's seed, r, is held fixed; whereas, every odd bit in a left parameter's seed, l, is fixed. The key which equals the two seeds for all fixed bits, m, is the partition where their pair is placed.

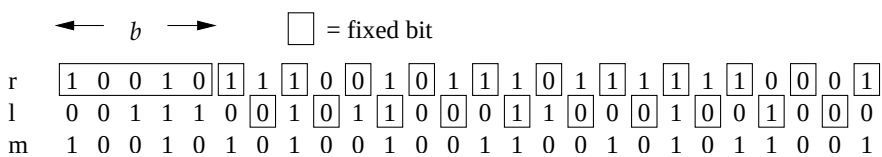

Fig. 1. Partition key m for example seeds r and l

Given a seed, s, we must identify the partitions, m, of all possible pairs involving s. Formally, we define the right and left rendezvous key sets, as

$$\mathscr{R}(s) := \{m \in \mathscr{K} : \forall i \in [1, 160] : ((i \equiv 0) \bmod 2 \text{ or } i \leq b) \implies m_i = s_i\} \quad (1)$$

$$\mathscr{L}(s) := \{m \in \mathscr{K} : \forall i \in [1, 160] : ((i \equiv 1) \bmod 2 \text{ and } i > b) \implies m_i = s_i\} (2)$$

Ignoring b, we mean here that $\mathscr{R}(s)$ is the set of all keys which share all the even bits of the seed s and $\mathscr{L}(s)$ the set of all keys which share all the odd bits.

The correctness of this strategy follows from the fact that $|\mathscr{R}(r) \cap \mathscr{L}(l)| = 1$; that is, there is exactly one partition key for every pair of left and right seeds. This is mathematically clear since the bit selecting predicates in the definitions of $\mathscr{R}$ and $\mathscr{L}$ are logical inverses.

3.2 Mapping Partitions to Nodes

$\mathscr{R}(s)$ and $\mathscr{L}(s)$ define how to select partitions or rendezvous keys. To find the actual rendezvous nodes, simply apply the DHT responsibility function, π. When $r \in \mathscr{K}$ is the seed used for a right parameter, it is placed on the set of rendezvous nodes $\pi(\mathscr{R}(r))$. Similarly, a left parameter with seed $l \in \mathscr{K}$ is placed on the set of rendezvous nodes $\pi(\mathscr{L}(l))$.

When a new parameter is placed on a rendezvous node, the node pairs the new parameter with all the parameters of the opposite type available on that node. Those pairs that the node is actually responsible for are then evaluated.

Since there is one rendezvous key in common for all pairs (r, l),

$$\emptyset \neq \mathscr{R}(r) \cap \mathscr{L}(l) \implies \emptyset \neq \pi(\mathscr{R}(r) \cap \mathscr{L}(l)) \subseteq \pi(\mathscr{R}(r)) \cap \pi(\mathscr{L}(l))$$

there is at least one rendezvous node in common. Unfortunately, there may in fact be several. Therefore, a rendezvous node, z, should check that it is responsible for the rendezvous key ($z \in \pi(\mathscr{R}(r) \cap \mathscr{L}(l))$) before operating on a pair of parameters. This way, we ensure that a given pair is evaluated at most once.

3.3 Modified DHT Routing

Although the sets $\mathscr{R}(s)$ and $\mathscr{L}(s)$ contain an extremely large number of keys, section 4 will argue that their mapping to rendezvous nodes is radically better than flooding. Specifically, a right parameter is placed on expected $2\sqrt{N}t^{-1}$ nodes and a left parameter is placed on expected $2\sqrt{N}t$ nodes. The t is a tunable constant related to b which trades off load in left parameters against load in left parameters.

In order to build a P2P application, this data placement strategy must be routed over a DHT. Rather than routing to each of the rendezvous keys separately, routing is carried out to all of them simultaneously.

The algorithm is a relatively simple adjustment to prefix routing. Normally, prefix routing DHTs resolve a bit at a time; this leads to a single routing path from source to destination. In our data placement strategy, some of the target key bits are fixed, but others are not. Whenever the predicate in equation 1 does not specify bit equality, the DHT simply routes right parameters in both the direction of a zero and of one. Similarly, the predicate in equation 2 is used for routing left parameters.

This means that delivery will follow the normal routing paths used for the rendezvous keys themselves. However, the shared intermediate nodes are only contacted once. Since every second step doubles the number of reached nodes, the interior nodes grow as two geometric series. By summing these series, the total nodes involved in routing will be $6\sqrt{N}t^{-1}$ for right parameters and $6\sqrt{N}t$ for left.

3.4 Balancing Non-Uniformity

In many applications, the left and right parameters will not be equal in dynamism or size. It is easy to exploit their ratio to further optimize the placement by dividing the nodes into groups. Left parameters are placed in each of these groups; however, right parameters stay within the group that originated them. This can heavily reduce the number of nodes impacted by the dynamism of the right parameters.

As will be explained in section 4, the most heavily loaded node will have expected $\frac{2R}{\sqrt{N}}t^{-1}$ right parameters and expected $\frac{2L}{\sqrt{N}}t$ left parameters placed on it. The constant t is selected to make the trade-off as desired.

Coming back to the balance factor, b, as seen in figure 1, it is defined depending on t as $b = 2\log t$. This balance factor is a global constant for the whole system and fixes the number of groups as $2^b = t^2$. For this reason, it should be selected carefully to reflect the constraints of the application.

A node is in the group identified by the first b bits of its random node id. As right parameters stay in the originating group, the first b bits of their seed should match the origin node. Figure 1 and equations 1 and 2 use b in this way.

4 DHT Routing Bound

Although our data placement strategy selects many rendezvous keys, the actual parameter load for each node is relatively small. By specifying only half of the bits in the routing keys, 2^{80} traditional DHT keys are addressed simultaneously. However, we can show that only $2\sqrt{\frac{N}{2^b}}$ nodes are responsible for these keys. The most loaded node (indeed, every node) has expected $\frac{2R}{\sqrt{N}}t^{-1}$ right and expected $\frac{2L}{\sqrt{N}}t$ left parameters placed on it.

The correct calculation of the bottleneck complexity requires a rather technical proof. Though the result presented here is correct, it is a simplification which does not consider the boundary cases. Argumentation similar to [12] would be applied to calculate the expected number of parameters on the most loaded node. This is extremely non-trivial, and requires additionally that L and R be sufficiently large relative to b and N. Instead, this section will count the expected replication of parameters.

It should be clear that a given parameter with a random seed is equally likely to rendezvous at any node; we will discuss balancing later and assume $b = 0$ initially.

First, we consider what bit interleaving does, and how the key sets $\mathscr{L}(s)$ and $\mathscr{R}(s)$ are distributed within the key space $\mathscr{K}$. The sets describe bit patterns of keys with every second bit fixed. With each fixed bit, the possible number of keys is reduced by half. Figure 2 illustrates the key segments selected by either a one or a zero for the first five bit digits. These segments are covers of $\mathscr{L}(s)$ and $\mathscr{R}(s)$

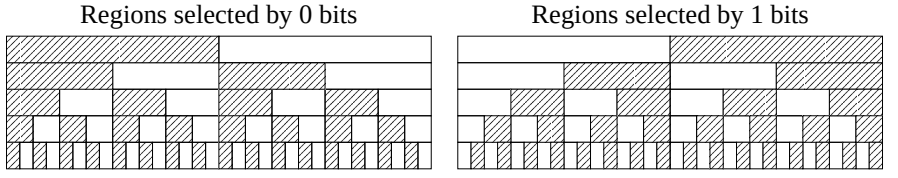

Fig. 2. Cutting up identifier space

and the rendezvous keys are exactly those that remain after intersecting all the areas of even or odd bits, respectively. Stopping this intersection prematurely at bit $d = \log N$ will give us a cover for the keys of sufficient granularity, as is shown below. If this cover is then mapped to rendezvous nodes, it would in turn be a cover for the rendezvous nodes, and thus a bound on the replication.

If the entire key space has area 1, at the d-th row, there are $2^d = N$ segments all of size $\frac{1}{2^d} = \frac{1}{N}$. Since each fixed bit eliminates half of the area, there are only $2^{\frac{d}{2}} = \sqrt{N}$ segments which are in the rendezvous key cover.

Now the involved rendezvous nodes given by the DHT's responsibility function must be counted. Recall from section 2.3 that our data placement strategy requires the DHT to have a Euclidean metric. This implies that nodes are responsible for disjoint intervals, $\pi^{-1}(n) = [a, b)$, which partition the key space $\mathscr{K}$. Although they do not have exactly the same size, in *expectation*, they all have size $\frac{1}{N}$. Considering a specific key covering segment, there are four ways a node's responsibility interval can overlap with it. These are illustrated in figure 3.

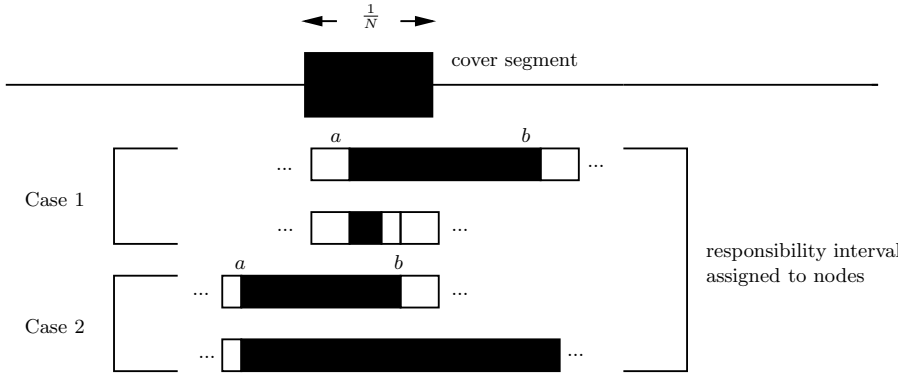

Fig. 3. Responsibility intervals overlapping a rendezvous key cover segment

In the first case, the left responsibility endpoint a is in the segment. Since all the segments have exactly the same size, the likelihood that it is in the considered segment is $\frac{1}{N}$. Thus, we expect only one node ($N\frac{1}{N}$) to have a responsibility interval with a left endpoint in the segment.

Now suppose the left responsibility endpoint is outside of the segment. The only way the interval can overlap is if the right endpoint reaches to the start of the segment. There can be at most one such responsibility interval, and therefore, node.

Taking these two cases together, we can say that no more than two nodes are expected to have a responsibility interval which intersects the segment. Since there are $\sqrt{N}$ segments, $2\sqrt{N}$ is the expected number of parameter replications in the system.

These numbers have to be adapted if grouping is applied (cf. section 3.4). Recall that the N nodes are conceptually broken into 2^b groups of roughly $\frac{N}{2^b}$ size. Left parameters are placed in all groups and require the aforemention load. Right parameters stay within their group by virtue of the construction of $\mathscr{R}$. The above arguments still hold, only the considered key space for each group is now $\frac{1}{2^b}$. So, the segmentation of the key space is stopped at $d = \log N - b$ and every N is substituted by $\frac{N}{2^b}$. Taken together with the tradeoff parameter, t, set to ideally $b = 2 \log t$, the load is as follows.

For right parameters, there are R parameters placed this way over N nodes. Dividing the load evenly between nodes yields

$$\frac{2R\sqrt{\frac{N}{2^b}}}{N} = \frac{2R}{\sqrt{N2^b}} = \frac{2R}{\sqrt{N}}\,t^{-1} \quad \text{per node.}$$

The left parameters are replicated between all 2^b groups, so the expression is

$$\frac{2L2^b\sqrt{\frac{N}{2^b}}}{N} = \frac{2L\sqrt{2^b}}{\sqrt{N}} = \frac{2L}{\sqrt{N}}\,t \quad \text{per node.}$$

5 Optimality Sketch

For applications which require that all pairs of right and left parameters be operated upon, this data placement strategy is optimal. This includes linear scans and difficult variants of publish-subscribe. We present here a brief and informal sketch of our optimality.

Theorem 1. *In any routing algorithm over N nodes which evaluates RL pairs of right and left parameters, there exists a node which receives at least $\frac{R}{\sqrt{N}}t^{-1}$ right and $\frac{L}{\sqrt{N}}t$ left parameters for some t.*

Proof. There are RL pairs of right and left parameters to be operated on. Therefore, there exists a node which evaluates $P = \frac{RL}{N}$ pairs. Suppose this node has X right parameters placed on it. Choose the balance factor $t = \frac{R}{X\sqrt{N}}$. Then $X = \frac{R}{\sqrt{N}}t^{-1}$.

The node can only operate on the product of all the right and left parameters placed on it. Therefore, there must be at least

$$\frac{P}{X} = \frac{\frac{RL}{N}}{\frac{R}{\sqrt{N}}t^{-1}} = \frac{L}{\sqrt{N}}t$$

many left parameters placed on this node.

6 Related Work

Distributed Keyword Search. There have been a number of proposals for implementing distributed keyword search. The majority of them are based on inverted indexes. An inverted index stores a list of documents for every possible keyword. To execute a query, the lists for involved keywords are intersected. The problem in a distributed network is that transferring these lists means a high network overhead for potentially uninteresting provisional results.

Gnawali [5] proposes a "Keyword-Set Search System" (KSS) that uses keyword sets rather than single keywords. Their intention is to reduce the number of lists that must be retrieved. On the other hand, keeping all pairs squares the number of lists per document and thus the cost for updates. All triples cubes the

cost. Reynolds and Vahdat [14] evaluated about 100,000 WWW queries during a 10-day period and found that 71.5% of them consisted of two or more keywords and about 40% contained three or more words. This puts the Gnawali [5] approach in a bad light.

Li, Loo et al. [7] like Reynolds and Vahdat [14] also aim at fast distributed list intersection. However, they start by comparing two partitioning techniques: by document and by keyword. Keyword partitioning requires transmitting the document lists, whereas document partitioning involves merely asking each partition to execute the query. According to their analysis, keyword partitioning in a DHT requires bandwidth that is about two orders of magnitude higher than in document partitioning.

Gnutella [8] is a well known decentralized system using document partitioning. However, it uses flooding to query all the partitions. The next version of Gnutella introduced ultra peers [8] to control the flooding, but this approach only reduces the number of flooded nodes. Harren et al. [6] propose to cut keywords into n-grams (fixed size substrings) and to index these. Their approach allows substring search and reduces the impact of typos. This will result in longer document lists, exacerbating the problem.

Like Gnutella, we propose to partition by document. However, instead of flooding, we propose to use the Bit Zipper to place the document meta data on $2\sqrt{N}$ nodes, and have queries pass through only $6\sqrt{N}$ nodes. On each node, a local inverted index could be used to execute the query, or one could use the technique of Harren et al. for more generality. With a network of four million users, we would require $\frac{1}{166}$-th of the bandwidth of Gnutella per node for an exhaustive search while distributing the computational load randomly over all nodes. If the work of Li, Loo et al. [7] is correct, this would place us over four orders of magnitude beyond the reach of distributed inverted indexes.

Publish-Subscribe. Subject-based publish-subscribe (SBPS) partitions notifications into hierarchically organized categories. This partitioning reduces the problem size immensely. Current SBPS systems like Scribe [16] and Bayeux [22] try to distribute their message load over large distributed systems. Coupled with the reduced problem size, they become highly scalable. Newsgroups show that SBPS scales quite well up to certain limits. However, newsgroup cross-posting shows that there are cases where the artificial hierarchy is too restrictive, or simply not relevant. On the other hand, high volume newsgroups have the problem of being too broad.

Pietzuch [11] and JMS [18] follow an approach that adds attribute filtering to the subject-based routing. This maintains the scalability attained from subject-based problem size reduction. Furthermore, it addresses the issue of high volume newsgroups by allowing filtering to be applied to reduce the noise. However, overly restrictive topics and conflicting categorizations remain a problem.

Content-based pub-sub (CBPS), on the other hand, uses filters over the content of notifications as its primary routing policy. Therefore, it reduces the load on subscribers by allowing them to specify their interests very precisely, without the problem of conflicting categorization. Rebeca [10] exemplifies pure CBPS,

routed over a distributed broker tree. The filters are used to determine whether or not to forward a notification. The problem with REBECA is that although clients are not heavily loaded, central brokers often carry linear load. Our previous approach [20] addresses this concern by eliminating the tree-like structure and using conceptually many trees. Unfortunately, this approach worked as attenuated broadcast. When perfect attenuation is too costly, this degenerates to flooding.

Tam et al. [19] partition based on the content of notifications. This indexing approach is commonly known from database systems. It combines high selectivity with good performance. Unfortunately, for some subscriptions, there can be no matching index. In databases, these queries would be solved by a linear scan. As mentioned in section 5, the Bit Zipper is the optimal solution for linear scans. Thus, our strategy provides an efficient fall-back mechanism which corresponds to the database table itself. In this manner, all queries can be dealt with, and the administrator can make index choices solely for performance gains.

7 Conclusion

In distributed rendezvous problems which frustrate partition by value, flooding to all N nodes was the best known technique. Our general-purpose data placement strategy can address these same problems with a cost of only $O(\sqrt{N})$ messages. Furthermore, when these rendezvous problems exhibit all-pairs evaluation, our strategy is asymptotically optimal. This includes important problems such as distributed linear scan and general content-based publish-subscribe.

The Bit Zipper Rendezvous fits naturally on most distributed hash tables. The additional routing cost in such a self-maintaining configuration is only a small multiplicative factor. Through partitioning *pairs* of left and right parameters, it achieves randomized load distribution. Hence, the load distribution will be asymptotically homogeneous.

For problem domains like publish-subscribe, where efficient parameter partitioning techniques exist, our work is complementary. When there are no matching indexes, our strategy provides a fall-back. This work paves the way for general purpose systems like databases where unanticipated queries may be posed.

The question of whether distributed keyword search requires the same message load remains open. However, since flooding (aka. partition by document [7]) was previously the fastest decentralized technique, our new result should in theory perform radically better. An implementation will show whether our predictions are correct.

References

1. *The 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, San Diego, California, USA, 2001.
2. *The 1st International Workshop on Peer-to-Peer Systems (IPTPS02)*, MIT Faculty Club, Cambridge, MA, USA, Mar. 2002.

3. K. Aberer. P-Grid: A Self-Organizing access structure for P2P information systems. In *Proc. of the 6th Intl. Conference on Cooperative Information Systems (CoopIS 2001), Trento, Italy*, 2001.
4. P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2), 2003.
5. O. D. Gnawali. A keyword set search system for Peer-to-Peer networks. Master's thesis, Massachusetts Institute of Technology, June 2002.
6. M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, and I. Stoica. Complex queries in DHT-based Peer-to-Peer networks. In *Proc. of the 1st Intl. Workshop on Peer-to-Peer Systems (IPTPS02)* [2].
7. J. Li, B. T. Loo, J. Hellerstein, F. Kaashoek, D. R. Karger, and R. Morris. On the feasibility of Peer-to-Peer web indexing and search. In *Proc. of the 2nd Intl. Workshop on Peer-to-Peer Systems (IPTPS03)*, Berkeley, CA, USA, Feb. 2003.
8. Q. Lv, S. Ratnasamy, and S. Shenker. Can heterogeneity make gnutella scalable? In *Proc. of the 1st Intl. Workshop on Peer-to-Peer Systems (IPTPS02)* [2].
9. P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proc. of the 1st Intl. Workshop on Peer-to-Peer Systems (IPTPS02)* [2].
10. G. Mühl. *Large-Scale Content-Based Publish/Subscribe Systems.* PhD thesis, Darmstadt University of Technology, Sept. 2002.
11. P. Pietzuch and J. Bacon. Hermes: A distributed event-based middleware architecture. In *Proc. of the 1st Intl. Workshop on Distributed Event-Based Systems (DEBS'02)*, Vienna, Austria, July 2002.
12. M. Raab and A. Steger. Balls into Bins - a simple and tight analysis. In *Proc. of the 2nd Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM'98)*, volume 1518 of *LNCS*, Barcelona, Spain, Oct. 1998.
13. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proc. of the 2001 ACM SIGCOMM Conference* [1].
14. P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. Technical report, Duke University, Sept. 2001.
15. A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale Peer-to-Peer systems. In *Middleware 2001*, Nov. 2001.
16. A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In *Proc. of the 3rd Intl. COST264 Workshop on Networked Group Communication (NGC 2001)*, London, UK, 2001.
17. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proc. of the 2001 ACM SIGCOMM Conference* [1].
18. Sun Microsystems Inc. Java message service specification 1.1, 2002.
19. D. Tam, R. Azimi, and H.-A. Jacobsen. Building Content-Based Publish/Subscribe systems with distributed hash tables. In *1st Intl. Workshop on Databases, Information Systems, and P2P (DBISP2P)*, Berlin, Germany, Sept. 2003.
20. W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. Buchmann. A Peer-to-Peer Approach to Content-Based Publish/Subscribe. In *Proc. of the 2nd Intl. Workshop on Distributed Event-Based Systems (DEBS'03)*, San Diego, CA, USA, June 2003.
21. B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, Computer Science Division, U. C. Berkeley, Apr. 2001.
22. S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Intl. Workshop on Network and OS Support for Digital A/V (NOSSDAV'01)*, June 2001.