ISSN:2231-0711
www.ijcset.net

Effective Load Balancing in Overlay Chord & P Grid Networks

S.Bhagya Rekha, Dr.M.Vijaya Saradhi

Department of Information Technology
Aurora's Engineering College,
Bhongir. Andhra Pradesh, India

Abstract— **In structured chord & P Grid systems finding the successor nodes, Load balancing and dynamic routing are the challenging issues because nodes are heterogeneous and dynamically nodes may join the network or old node may release from network. We present a general framework, HiGLOB, for global load balancing in structured Chord systems. Current load balancing, finding optimized node and dynamic routing algorithms are based on their own mechanisms they typically adhoc, heuristic based, and localized. Each node in HiGLOB has three key components:**

1) **A Histogram manager maintains a histogram that reflects a global view of the distribution of the load in the system**
2) **A Replication Load Balancing manager redistributes the load whenever the node becomes overloaded or underloaded.**
3) **A Routing Manager finds the best configured and optimized node and also finds the way how nodes are selected to overlay routing tables. We implement this mainly for Chord networks demonstration.**

Keywords- *Histogram, peer-peer networks, Chord networks, overlay networks, P Grid networks, DHT*

I. INTRODUCTION

Load balancing problems in P2P systems come along in many facets. In this paper we report on our results on solving simultaneously a combination of two important load balancing problems with conflicting requirements—storage and replication load balancing–in the construction and maintenance of distributed hash tables [1] (DHTs) to provide an efficient, distributed, scalable, and decentralized indexing mechanism in P2P systems like chord and P Grid. The basic principle of distributed hash tables is the association of peers with data keys and the construction of distributed routing data structures to support efficient search. Existing approaches to DHTs mainly differ in the choice of topology (rings [2], multi-dimensional spaces [3], or hypercubes [4]), the specific rules for associating data keys to peer keys (closest, closest in one direction), and the strategies for constructing the routing infrastructure.

To use the available resources of peer best, a *storage load balancing* approach is applied in all DHTs, i.e., associating keys to peers in a way so that the number of data items each peer is associated with, is uniform in terms of storage consumption. Most existing solutions achieve this by first mapping data keys and peer identifiers into the same space using uniform hashing. Using this approach storage load balancing essentially translates into the classical *balls into bins* problem [5], where peers are the bins (the peer identifier determines the data space) and the data items are the balls. Adapting the classical load-balancing mechanisms in the context of P2P systems, such as load-stealing and load-shedding schemes, in which peers share load with random peers, e.g., [6, 7], or power of two choices [8], lead to the need of redirections which compromise the search efficiency, because keys become increasingly decoupled from the peers associated with the corresponding key space and other structural properties are violated, since routing needs additional redirections. The problem is further aggravated with the growing recognition of the fact that uniform hashing to generate keys which are uniformly distributed on the key space jeopardizes the possibility to do searches on data using the data key semantics, typically the ordering of keys to enable semantically rich queries like range queries.

The approach which we will follow in this paper is to have peers dynamically change their associated key space ("bin adaptation") decoupled from their (unique and stable) identifier, and the routing between peers is based on the associated key space, rather than on the peer identifiers. Following this approach, the partitioning of the key space dynamically adapts to any data distribution, such that uniform distribution of data items over each partition of the key space is achieved. This leads to uneven sizes of the partitions of the key space, which can be viewed in the one-dimensional case analogously to having an unbalanced search tree. This implies a risk of sacrificing search efficiency. However, we show that due to the distributed and randomized routing process we propose (in PGrid), this risk can be contained, such that searches can be performed with communication cost of $O(log(|\Pi|))$ with high probability where $|\Pi|$ is the number of partitions of the key space, irrespective of the key space partitioning. This satisfies the condition of efficient searches in the context of Chord systems under the (standard) assumption that in a P2P network, local resources such as computation and storage are cheap, but communication costs (messages or latency) and network maintenance (routing) are expensive.

Beyond search efficiency, another important issue in Chord systems is resilience against failures. The standard response to this problem is to introduce redundancy. In the context of DHTs this corresponds to associating multiple peers with the same partition of the search space, i.e., peers being replicas of each other. A fair use of resources implies uniform replication of all data partitions, which introduces the *replication load* balancing problem.

II. THE P-GRID DATA STRUCTURE

We use our DHT-based P-Grid P2P system [13, 14] to evaluate the approach described in this paper. We assume that the reader is relatively familiar with the standard distributed hash table (DHT) approach [1] and thus only provide P-Grid's distinguishing characteristics. In P-Grid, peers refer to a common underlying tree structure in order to organize their routing tables (other topologies in the literature include rings [2], multi-dimensional spaces [3], or hypercube [4]). In the following, for simplicity of presentation, we will assume that the tree is binary. This is not a fundamental limitation as a generalization of P-Grid to k-ary structures has been introduced in [15]. Note that the underlying tree does not have to be balanced but may be of arbitrary shape, thus facilitating to adapt the overlay network to unbalanced data distribution [16].

Each peer $p \in P$ is associated with a leaf of the binary tree. Each leaf corresponds to a binary string $\pi \in \Pi$. Thus each peer p is associated with a path $\pi(p)$. For search, a peer stores for each prefix $\pi(p, l)$ of $\pi(p)$ of length l a set of references $\rho(p, l)$ to peers q with property $\pi(p, l) = \pi(q, l)$, where π is the binary string π with the last bit inverted. This means that at each level of the tree the peer has references to some other peers that do not pertain to the peer's subtree at that level. This enables the implementation of prefix routing for search. Each peer stores a set of data items $\delta(p)$. Ideally for $d \in \delta(p)$ the key $\kappa(d)$ of d has $\pi(p)$ as prefix. However, we do not exclude that temporarily other data items are also stored at a peer, that is, the set $\delta(p, \pi(p))$ of data items whose key matches $\pi(p)$ can be a proper subset of $\delta(p)$. In addition, peers also maintain references $\sigma(p)$ to peers having the same path, i.e., their replicas. In a stable state (i.e. where no more maintenance operations are applicable) the set of paths of all peers is prefix-free and complete, i.e., no two peers p and q exist such that $\pi(p) \subset \pi(q)$, i.e., $\pi(p)$ is a proper prefix of $\pi(q)$ and if there exists a peer p with path $\pi(p)$, then there also exists a peer q with $\pi(p) = \pi(q)$. This guarantees full coverage of the search space and complete partitioning of the search space among the peers. All data stored at a peer then matches its path. For search, P-Grid uses a prefix routing strategy. When receiving a search message for key κ from peer p, a peer q checks whether its path is a prefix of κ. If yes, it checks whether it can return a query result from its data store. If not, it randomly selects a peer r having a common prefix of maximal length with κ from its routing table and forwards the request to peer r.

The algorithm always terminates successfully in the stable state: Due to the definition of $\rho(p, l)$, this prefix routing strategy will always find the location of a peer at which the search can continue (use of completeness) and each time the query if forwarded, the length of the common prefix of $\pi(p)$ and κ increases. It is obvious that this search algorithm is efficient ($O(log(|\Pi|))$) for a balanced tree, i.e., all paths associated with peers are of equal length. Skewed data distributions may imbalance the tree, so that it may seem that search cost may become nonlogarithmic in the number of messages. However, in [16] we show that due to the probabilistic nature of the P-Grid approach this does not pose a problem. The expected search cost measured by the number of messages required to perform the the search remains logarithmic, independently how the P-Grid is structured.

Theorem 1. *The expected search cost for the search of a specific key $\kappa(d)$ using a P-Grid network N that is randomly selected among all possible P-Grids, starting at a randomly selected peer p with $\pi(p) \in \Pi$ is less than $log(|\Pi|)$.*
Although this applies to the special case of prefix-free P-Grids, we have shown by simulation that the result also applies to more general cases. A formal proof of this theorem is given in [16]. Due to space limitations we can only provide the intuition which is underlying the proof. Basically we show that the path resolution in the forwarding process normally is not done bit by bit but for longer bit sequences at the processing peers thus keeping the number of messages required in the forwarding process logarithmic. Additionally, [16] shows that the probability that a search does not succeed after k steps ($1 \leq k \leq max(|\pi|, \pi \in \Pi)$) is smaller than $\log(n|\Pi|)k-1$ $(k-1)$!

III. P GRID CONSTRUCTION ALGORITHM

The construction and maintenance of P-Grid is based exclusively on local interactions among peers in order to observe the principle of locality. In this section we give an overview of the *possible* interactions that determine the behavioral options of peers. As peers are autonomous they may use different *strategies* for entering into such local interactions. The choice of concrete strategies will be essential with respect to the global efficiency of the system and discussed later. Interactions among peers are either performed actively by the peers (similar to the peer discovery in Gnutella using the ping-pong messages) or are performed reactively triggered by earlier interactions or search messages. For maintenance purposes, the following interactions occur among two peers p and q: – *balancedSplit*(p, q): The peers check whether their paths are identical. If yes, they extend their paths by complementary bits, i.e., partition (split) the key space they are responsible for. To maintain consistency they exchange their data corresponding to their updated paths and add each other to their routing table. This enables the refinement of the indexing structure into subspaces which are sufficiently populated with data.

– *unbalancedSplit*(p, q): The peers check whether $\pi(p)$ is a proper prefix of $\pi(q)$. In the case $\pi(p)$ is a proper prefix of $\pi(q)$, p extends its path by one bit complementary to the bit of $\pi(q)$ at the same level. The peers exchange their data corresponding to the updated paths and update their routing table. This enables the refinement of the indexing structure into subspaces as in the previous case, but covers the frequently occurring situation that peers have already specialized to different degrees. The case where $\pi(q)$ is a proper prefix of $\pi(p)$ is treated analogously.

– *adoptP ath*(p, q): Peer p becomes a copy (replica) of peer q. In order to avoid data loss peer p attempts to locate peers covering the same subspace and to delegate any non-replicated data items there. If this is not possible it keeps data items not matching the new path to delegate it at a later time.

– *balancedDataExchange*(p, q): The peers check whether their paths are identical. If yes, they replicate mutually all data pertaining to their common path which increases resilience (availability of the data items).
– *unbalancedDataExchange*(p, q): The peers check whether $\pi(p)$ is a proper prefix of $\pi(q)$ (or vice versa). If yes, data of p pertaining to $\pi(q)$ is moved to q.
– *refExchange*(p, q): The peers exchange entries from their routing tables up to the level corresponding to the length of their common prefix randomly. This interaction randomizes the contents of the routing tables which is essential to maintain routing efficiency, in particular in the unbalanced case [16].
– *forwarding*(p, q): If the peers' paths are not in a prefix relationship the peer q provides the peer p with an address of a peer r selected from its routing table which shares a prefix of maximal length with $\pi(p)$ (or vice versa). Then peer p enters into an interaction with peer r. The conditions under which these rules are applied determine the strategies peers pursue in interactions. From these local interaction strategies a global system behavior emerges. The following sequence of actions performed by peers p and q entering into an interaction describes a possible strategy to construct a P-Grid structure from an initial state where all peers store some initial data and have empty paths and routing tables.

Algorithm 1

refExchange(p, q);
if $|\delta(p, \pi(p)) \cup \delta(q, \pi(q))| \leq 2\delta max$**then** *balancedDataExchange*(p, q)
if $|\delta(p, \pi(p)) \cup \delta(q, \pi(q))| > 2\delta max$;**then** *balancedSplit*(p, q)
unbalancedSplit(p, q);
forwarding(p, q);

In this strategy, peers first exchange routing information if possible. Then depending on the relationship among their paths and the current storage load they select one of the four subsequent actions. (Note that we do not explicitly repeat the necessary conditions on the path relationship for executing these actions). We observe that due to the *forwarding* action any initial interaction will eventually lead to the enabling of one of the balanced or unbalanced split or data exchange operations. For a uniform data distribution and provided that the total number of data items is less than $\delta maxn$, where is the total number of peers, this algorithm will end up in a state where each peer carries at most $2\delta max$ data items, the P-Grid structure is (approximately) balanced and all replica peers store the same data.

Theorem 2. *If the total number of data items is less than δmaxn and data keys are uniformly distributed Algorithm 1 results in a steady state in which the PGrid is prefix-free and complete and each peer p with replicas has a data load smaller than 2δmax, all replicas store the same data and in expectation all data items are equally replicated.*

Proof Sketch: First we have to show that the steady state is reached. Prefix freeness follows from the fact that whenever a peer has a path that is a prefix of another peer's path, it eventually will encounter this peer and perform an unbalanced split. Completeness follows from the fact that new paths can only occur as the result of a balanced split. If a peer has a replica and the data load is larger than $2\delta max$, it will eventually perform a split with its replica. If peers with the same path have different data items then they will eventually perform a balanced data exchange. Second, it is easy to see that once the steady state is reached none of the rules can induce further changes to the paths or data associated with the peers.

The problem is that with this strategy peers preferably adapt shorter paths and therefore even though peers try to balance their storage load, the distribution of replicas over the different paths becomes unbalanced in the case of non-uniform distribution of data keys: In a balanced split the same number of peers decide for each side of the data space independent of the actual distribution of data among the two subspaces, and in an unbalanced split peers decide for one side with a probability proportional to the number of peers already specialized for each side of the data space, but independent of the number of data items present in the two subspaces. This has the further effect that fewer peers specialize on paths with higher data load, and sooner end up without replicas. They thus lack the capacity to further refine the path and thus reduce their data load. To address this problem we consider a different strategy to improve replica balancing already during construction of the P-Grid structure.

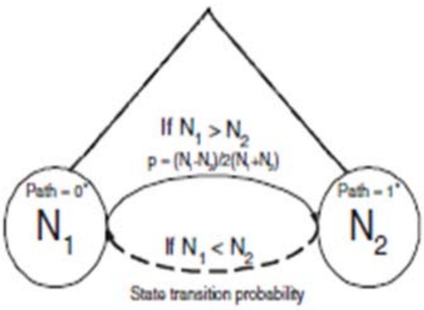
Algorithm 2

refExchange(p, q);
if $|\delta(p, \pi(p)) \cup \delta(q, \pi(q))| \leq 2\delta max \wedge \gamma([0, 1]) < \alpha$**then** *balancedDataExchange*(p, q)
if $|\delta(p, \pi(p)) \cup \delta(q, \pi(q))| > 2\delta max$;**then** *balancedSplit*(p, q)
if $\gamma([0, 1]) < \beta$**then** *unbalancedSplit*(p, q)**else** *adoptP ath*(p, q);
forwarding(p, q);

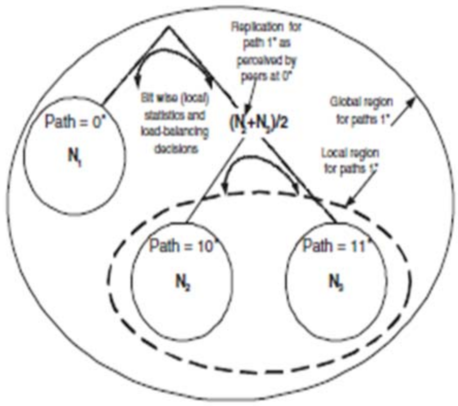
In this strategy two mechanisms work together to improve replica balancing. First, balanced splits are not always performed eagerly, but with reduced probability α, where α may depend on the locally observed load distribution. Thus more unbalanced split situations occur. In those situations peers only either extend their path opposite to the path of the encountered peer or adopt the path. The decision is based on a control parameter β which again may depend on the locally observed load distribution. As a result, if α and β are properly chosen, those subspaces will be populated by more peers that contain more data. Even though, this heuristic approach does not necessarily induce a perfectly uniform replica distribution, it substantially improves the state reached after the P-Grid construction. The remaining balancing is then achieved by the sampling-based replication maintenance algorithm, that we will introduce subsequently. Having a more uniform initial replica distribution substantially reduces the effort required from the maintenance algorithms in order to rectify the distribution. The construction algorithm can be extended to a maintenance algorithm (path retraction). The path retraction is dual to the path extension, such that if two partitions do not have enough data (< $\delta max/2$), then such partitions would be merged.

IV. Replication Maintenance Algorithm

To address the balancing problems discussed in the previous sections, we use a reactive randomized distributed algorithm which tries to achieve globally uniform replication adaptive to globally available resources based on locally available (gathered) information. Before introducing the algorithm we introduce the principles underlying its design. Consider a P-Grid of leaves as shown in Figure 1(a). Let $N1 > N2$ be the actual number of replica peers with paths 0 and 1. To achieve perfect replication balancing $(N1-N2)/2$ of the peers with path 0 would need to change their path to 1. Since each of the peers has to make an autonomous decision whether to change its path, we propose a randomized decision: Peers decide to change their paths with probability $p0\to1 = max(N1-N2/2N1,\ 0)$ (no $0 \to 1$ transition occurs if $N2 > N1$).



A) P-Grid with two leaves



B) P-Grid with three leaves

Figure 1.

Collecting Statistical Information at Peer p**:** In a decentralized setting, a peer p has to rely on sampling to obtain an estimate of the global load imbalance:Upon meeting any random peer q, peer p will gather statistical information for all possible levels $l \le |\pi(p)|$ of its path, and update the number of peers belonging to the same subspace $\Sigma p(l) = |\{q \text{ s.t. } |\pi(p) \cap \pi(q)| \ge l\}|$ and the complimentary subspace $\Sigma p(l) = |\{q \text{ s.t. } \pi(p, l) = \pi(q, l)\}|$ at any level l. When peers p and q interact, statistics gathering is performed as follows

$l := |\pi(p) \cap \pi(q)|$;

$\Sigma p\_q(l) := \Sigma p\_q(l) + 21{+}l{-}|\pi(q\_p)|$;

$\forall 0 \le i < l\ \Sigma p\_q(i) := \Sigma p\_q(i) + 21{+}i{-}|\pi(q\_p)|$;

where the meta-notation $p\ q$ denotes that the operations are performed symmetrically both for p and q.

Choosing Migration Path for Peer p**:** A path change of a peer only makes sense if it reduces the number of replicas in an underpopulated subspace (data). Therefore, as soon as a minimum number of samples have been obtained, the peer tries to identify possibilities for migration. It determines the largest $lmax$ such that $\Sigma p(lmax)$

$\Sigma p(lmax) > \zeta$ where $\zeta \ge 1$ is a dampening factor which avoids migration if load-imbalance is within a ζ factor. We set $lmax := \infty$ if no level satisfies the condition.If all peers try to migrate to the least replicated subspace, we would induce an oscillatory behavior such that the subspaces with low replication would turn into highly replicated subspaces and vice versa. Consequently, instead of greedily balancing load, peers essentially have to make a probabilistic choice proportional to the relative imbalance between subspaces. Thus $lmigration$ is chosen between $lmax$ and $|\pi(p)|$ with a probability distribution proportional to the replication load-imbalance $\Sigma p(i)\ \Sigma p(i)\ ,\ |\pi(p)| \ge i \ge lmax$. Thus the migrations are prioritized to the least populated subspace from the peer's current view, yet ensuring that the effect of the migrations is fair, and not all take place to the same subspace. There are subtle differences in our approach to replication balancing in comparison to the classical balls into bins load balancing approach, because in our case there are no physical *bins*, which would share load among themselves, and it is rather the *balls* themselves, which need to make an autonomous decision to migrate. Moreover, the load sharing is not among bins chosen uniformly, but is prioritized based on locally gathered approximate global imbalance knowledge.

To further reduce oscillatory behavior, the probability of migration is reduced by a factor $\xi \le 1$. As migration is an expensive operation—it leads to increased network maintenance cost due to routing table repairs, apart from the data transfer for replicating a new key space—it should only occur if long-term changes in data and replication distribution are observed and not result from short term variations or inaccurate statistics. The parameters ζ and ξ are design parameters and the impact of their choice on the system behavior will be further explored in Section 5.

1) ***Migrating Peer*** p*: The last aspect of replication load balancing is the action of changing the path. For that, peer* p *needs to find a peer from the complimentary subspace and thus inspects its routing table* ρ*(*p, lmigration*) (s.t.* π*(*p*)* ∩ π*(*q*)* = lmigration*). After identifying a peer* q*,* p *clones the contents of* q*, including data and routing table, i.e.,* δ*(*p*) :=* δ*(*q*) and* ρ*(*p, ∗*) =* ρ*(*q, ∗*), and the statistical information is reset in order to account for the changes in distribution*

V. Simulation Results

This section highlights some of the many experiments we performed using a simulator implemented in Mathematica to evaluate the construction and maintenance algorithms. The simulations aim at verifying the load balancing characteristics of the algorithms, and do not model aspects related to the physical runtime environment with different network topologies, communication latencies, or heterogeneity of resources of nodes. Unless mentioned otherwise, simulations were performed with 256 peers. This relatively low number was chosen to keep simulation time manageable. From the design of the algorithms it is clear that the results will scale up to larger populations. To support this, we will give one result for the complete maintenance algorithm with changing peer population at the end. The data was chosen from a Zipf distribution with parameter $\theta = 0.8614$ such that the frequencies of keys were monotonically increasing with decreasing size of the key. We set $\delta max = 50$.

Replication Load Balancing Throughout Construction: In Section 3 we discussed a possibility to maintain better replica load balancing while establishing storage load balance during P-Grid construction, by reducing the probability α of balanced splits of the key space (while choosing $\beta = 1$). In Table 1 we show the results of an experiment in which each peer initially holds 15 data items.We see how a reduction of α reduces both the variance $R\sigma 2$ in the replication factors for the key space partitions and the maximum replication factor $Rmax$, where $R\mu$ is the average replication factor with an expected value of 3.33.2 With lower probabilities more interactions occur to reach a steady state.

TABLE I. INFLUENCE OF SPLITTING PROBABILITY *A* ON DISTRIBUTION OF REPLICATION FACTOR

α	Interactions	R_μ	R_{σ^2}	R_{max}
0.05	40,000	3.32	1.82	10
0.1	35,000	3.20	1.99	9
0.5	20,000	3.55	3.39	21
1.0	20,000	3.28	3.94	23

Replication Load Balancing Throughout Maintenance: Given a P-Grid that partitions the data space such that the storage load is (approximately) uniform for all partitions, migrations are used to establish simultaneous balancing of replication factors for the different partitions without changing the data space partitioning.

Figure 2 shows the reduction of the variance of the distribution of replication factors compared with the initial variance as a function of the number of key space partitions. The simulation was starting from an initially constructed, unbalanced P-Grid network with replication factors chosen uniformly between 10 and 30 for each of the key space partitions. We compared the effect of an increasing number of key space partitions ($p = \{10, 20, 40, 80\}$) on the performance of the replication maintenance algorithm. One observed that the reduction of variance increases logarithmically with the number of partitions. For example, for $p = 80$ the initial variance is reduced by approximately 80%. We conducted 5 simulations for each of the settings. The error bars give the standard deviation of the experimental series. The right part of Figure 2 shows the rate of the reduction of variance of replication factors as a function of different numbers of peers associated with each key partition. We used a P-Grid with $p = 20$ partitions and assigned to each partition uniformly randomly between k and $3k$ peers, such that the average replication factor was $2k$. The other settings were as in the previous experiment. Actually variance reduction appears to slightly improve for higher replication factors. This results from the possibility of a more fine-grained adaptation with higher replication factors.

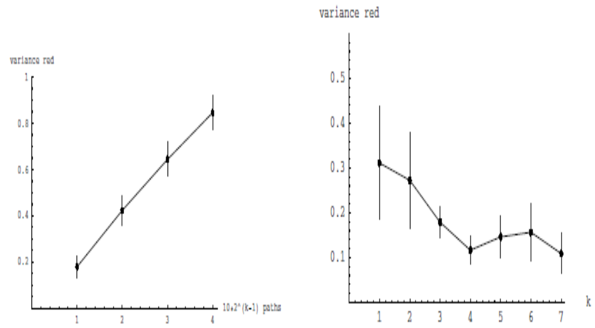


Figure 2. Maintenance of replication load-balance

Simultaneous Balancing of Storage and Replication Load in a Dynamic Setting: In this experiment we studied the behavior of the system under dynamic changes of the data distribution. Both storage load balancing by restructuring the key partitioning (i.e., extending and retracting paths) and replication balancing by migration were performed simultaneously.We wanted to answer the following two questions: (1) Is the maintenance mechanism adaptive to changing data distributions? (2) Does the combination of restructuring and migration scale for large peer populations?

Table 2 shows the results of our experiments. We executed an average of 382 rounds in which each peer initiated interleaved restructuring and maintenance operations, which was sufficient for the system to reach an almost steady state. $R\sigma^2$ is the variance of the replication factors for the different paths and $D\sigma^2$ is the variance of the number of data items stored per peer.

TABLE II. REPLICATION FACTOR RESULTS OF SIMULTANEOUS BALANCING

Number of peers	Number of paths		R_{σ^2}		D_{σ^2}	
	initial	final	initial	final	initial	final
219	10	43	55.47	3.92	180,338	175
461	20	47	46.30	10.77	64,104	156
831	40	50	40.69	45.42	109,656	488
1568	80	62	35.80	48.14	3,837	364

The experiments show that the restructuring of the network as well as replication balancing was effective and scalable: (1) In all cases the data variance dropped significantly, i.e., the key space partitioning properly reflects the (changed) data distribution. Because of the randomized choices of the initial P-Grid structure and the data set, the initial data variance is high and varies highly. It actually depends on the degree to which the randomly chosen P-Grid and the data distribution already matched. From the case $p = 40$ (number of initial paths), we conclude that this has also a substantial impact on the convergence speed since more restructuring has to take place. Actually, after doubling the number of interactions, the replication variance dropped to 20.93, which is an expected value. (2) With increasing number of replicas per key partition the replication variance increases. This is natural as fewer partitions mean higher replication on average and thus higher variance. (3) With increasing peer population the final data variance increases. This is expected as we used a constant number of interactions per peer and the effort of restructuring grows logarithmically with the number of key partitions.

The algorithms do not require much computation per peer hence have a low overhead. Simulating them, however takes considerable effort: A single experiment with $3 * 10^5$ interactions for the results in this section took up to 1 full day. Thus we had to limit the number and size of the experiments. Nevertheless they indicate the feasibility, effectiveness and scalability of the algorithms.

VI. RELATED WORK

For data replication in P2P systems we can distinguish six different methods (partially according to the classification from [17]): *Owner replication* replicates a data object to the peer that has successfully located it through a query (Napster, Gnutella, Kazaa). *Path replication* replicates a data object along the search path that is traversed as part of a search (Freenet, some unstructured P2P networks). *Random replication* replicates data objects as part of a randomized process. [17] shows that for unstructured networks this is superior to owner and path replication. *Controlled replication* replicates a data object a pre-defined number of times upon insertion (Chord [2], CAN [3], and Pastry [9]). This approach does not adapt replication to the changing environment with variable resource availability. The replication balancing mechanism proposed in this paper (and as used in P-Grid) is *adaptive to the available resources* in the system. This mechanism tries to uniformly exploit the storage resources available at peers, and thus achieve uniform distribution of the replicas of data objects. In addition, *query adaptive replication* [11] can be used in various structured overlays, complementing controlled or available resource adaptive replication.Replication of index information is applied in structured and hierarchical P2P networks. For the super-peer approach it has been shown that having multiple replicated super-peers maintaining the same index information increases system performance [18]. Structured P2P networks maintain multiple routing entries to support alternative routing paths if a referenced node fails. With respect to load balancing in DHT based systems only a few recent works have been reported. The application of uniform hashing and its limited applicability have already been discussed in the introduction.

The load balancing strategy for Chord proposed in [7] uses multiple hash functions instead of only one to select a number of candidate peers. Among those the one with the least load stores the data item and the others store pointers to it. This scheme does not scale in the number of data items due to the effort incurred by redirection pointer maintenance. Moreover, using a predetermined number of hash functions do not give any adaptivity according to the systems requirement. Also Chord's original search no longer works and essentially multiple Chord overlays have to be maintained which are interconnected among themselves in a possibly unpredictable manner.Another scheme for load balancing for Chord is suggested in [19] based on virtual servers. Nodes are responsible to split the data space to keep the load of each virtual server bounded. The splitting strategy is similar to the splitting used in our storage load balancing strategy, however, this work does not consider the effects on replication nor on search efficiency. Online load-balancing has been a widely researched area in the distributed systems domain. It has often been modeled as balls into bins [5]. Traditionally randomized mechanisms for load assignment, including load-stealing and loadshedding and power of two choices [8] have been used, some of which can partly be reused in the context of P2P systems as well [7, 6]. In fact, from storage load balancing perspective, [6] compares closest to our approach because it provides storage load-balancing as well as key order preservation to support range queries, but in doing so, they no more provide any guarantee for efficient searches of isolated keys. As mentioned earlier, load-balancing in DHTs poses several new challenges, which call for new solutions. We need to deal with the dynamic membership (off-online behavior of peers) and dynamic content, and there is neither global coordination nor global information to rely on, and the load-balancing mechanism should ideally not compromise the structural properties and the search efficiency of the DHT, while preserving the semantic information of the data. In [20], storage load-balancing is achieved by reassignment of peer identifiers in order to deal with network churn, but this scheme is designed specifically for uniform load distribution only.

The dynamic nature of P2P systems is also different from the online load-balancing of temporary tasks [21] because of the lack of global knowledge and coordination. Moreover, for replication balancing, there are no real bins, and actually the number of bins varies over time because of storage load balancing, but the balls (peers) themselves have to autonomously migrate to replicate overloaded key spaces. Also for storage load balancing, the balls are essentially already present determined by the data distribution, and it is essentially the bins that have to fit the balls by dynamically partitioning the key space, rather than the other way round. Substantial work on distributed data access structures has also been performed in the area of distributed databases on scalable data access structures, such as [22, 23]. This work is apparently relevant, but the existing approaches apply to a different physical and application environment. Databases are distributed over a moderate number of fairly stable database servers and workstation clusters. Thus reliability is assumed to be high and replication is used only very selectively [24] for dealing with exceptional errors. Central servers for realizing certain coordination functions in the network are considered as acceptable and execution guarantees are mostly deterministic rather than probabilistic. Distributed search trees [25] are constructed by a full partitioning, not using the principle of scalable replication of routing information at the higher tree levels,as originally published in [1] (with exceptions [26]). Nevertheless, we believe that at the current stage the potential of applying principles developed in this area to P2P systems is not yet fully exploited.

VII. CONCLUSIONS

Existing uncoordinated online load-balancing mechanisms do not address the requirements of DHT-based P2P networks. In this paper we compared the new load-balancing problems of such systems with the standard model so that wherever possible we can apply existing solutions. But more importantly, we identified the new and specific requirements of this family of Chord and P Grid systems, and proposed new algorithms to efficiently achieve simultaneous storage and replication load-balancing relying only on local information. Some of the important novelties of our solution in comparison to other proposed Chord load balancing mechanisms are: Our mechanism allows the access structure to adapt and restructure dynamically, but preserves its structural properties, unlike other mechanisms which require extrinsic mechanisms like redirection pointers, that make queries inefficient. The effort incurred by our load-balancing approach is low because it requires no extra communication but we gather statistic data from normal interactions and "piggy-back" the load-balancing into the standard information exchanges required by the DHT. We also preserve key ordering, which is vital for semantically rich queries like range queries. Using randomized routing choices, search efficiency is guaranteed with high probability, irrespective of key distribution. Additionally, unlike some other proposals, our solution does not require the peers to change identity which allows us to retain existing knowledge and semantics, that may be exploited by higher level applications

VIII. REFERENCES

[1]. Plaxton, C.G., Rajaraman, R., Richa, A.W.: Accessing Nearby Copies of Replicated Objects in a Distribute d Environment. In: Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA). (1997)

[2] Stoica, I., Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H.: Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In: Proceedings of the ACM SIGCOMM. (2001)

[3]. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A Scalable Content- Addressable Network. In: Proceedings of the ACM SIGCOMM. (2001)

[4]. Schlosser, M., Sintek, M., Decker, S., Nejdl, W.: Hypercup–hypercubes, ontologies, and efficient search on peer-to-peer networks. LNCS **2530** (2003)

[5]. Raab, M., Steger, A.: "Balls into Bins" - A Simple and Tight Analysis. In: RANDOM. (1998)

[6]. Karger, D.R., Ruhl, M.: New Algorithms for Load Balancing in Peer-to-Peer Systems (2003) IRIS Student Workshop (ISW).

[7]. Byers, J., Considine, J., Mitzenmacher, M.: Simple Load Balancing for Distributed Hash Tables. In: 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03). (2003)

[8]. Mitzenmacher, M.: The power of two choices in randomized load balancing. IEEE Transactions on Parallel and Distributed Systems **12** (2001) 1094–1104

[9]. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. Lecture Notes in Computer Science **2218** (2001)

[10]. Alima, L.O., El-Ansary, S., Brand, P., Haridi, S.: DKS(N,k,f): A Family of Low Communication, Scalable and Fault-Tolerant Infrastructures for P2P Applications. In: 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID). (2003)

[11]. Datta, A., Aberer, K., Nejdl, W.: Principles of Query-Adaptive Optimal Replication in DHTs. Technical Report IC/2004/110, Ecole Polytechnique Fdrale de Lausanne (EPFL) (2004)

[12]. Renesse, R.V., Birman, K.P., Vogels, W.: Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. ACM Trans. Comput. Syst. **21** (2003) 164–206

[13]. Aberer, K.: P-Grid: A self-organizing access structure for P2P information systems. In: Proceedings of the Sixth International Conference on Cooperative Information Systems (CoopIS). (2001)

[14]. Aberer, K., Hauswirth, M., Punceva, M., Schmidt, R.: Improving Data Access in P2P Systems. IEEE Internet Computing **6** (2002)

[15]. Aberer, K., Punceva, M.: Efficient Search in Structured Peer-to-Peer Systems: Binary v.s. k-ary Unbalanced Tree Structures. In: International Workshop On Databases, Information Systems and Peer-to-Peer Computing. Collocated with VLDB 2003. (2003)

[16]. Aberer, K.: Efficient Search in Unbalanced, Randomized Peer-To-Peer Search Trees. Technical Report IC/2002/79, Swiss Federal Institute of Technology, Lausanne (EPFL) (2002)

[17]. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S.: Search and replication in unstructured peer-to-peer networks. In: International Conference on Supercomputing. (2002)

[18]. Yang, B., Garcia-Molina, H.: Improving Search in Peer-to-Peer Networks. In: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02). (2002)

[19]. Rao, A., Lakshminarayanan, K., Surana, S., Karp, R., Stoica, I.: Load Balancing in Structured P2P Systems. In: Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03). LNCS, Springer (2003)

[20]. Manku, G.S.: Randomized ID Selection for Peer-to-Peer Networks. Technical report, Stanford University (2004) http://dbpubs.stanford.edu:8090/aux/ index-en.html.

[21]. Azar, Y., Kalyanasundaram, B., Plotkin, S., Pruhs, K., Waarts, O.: On-line load balancing of temporary tasks. Journal of Algorithms

[22] (1997) 93–110 22. Litwin, W., Neimat, M., Schneider, D.A.: RP*: A Family of Order Preserving Scalable Distributed Data Structures. In: VLDB. (1994) 342–353

[23]. Litwin, W., Neimat, M., Schneider, D.A.: LH* – A Scalable, Distributed Data Structure. ACM Transactions on Database Systems (1996) 480–525

[24]. Litwin, W., Schwarz, T.: LH*RS: A High-Availability Scalable Distributed Data Structure using Reed Solomon Codes. In: SIGMOD Conference. (2000) 237–248

[25]. Kroll, B., Widmayer, P.: Distributing a Search Tree Among a Growing Number of Processors. In: ACM SIGMOD Conference. (1994) 265–276

[26]. Yokota, H., Kanemasa, Y., Miyazaki, J.: Fat-Btree: An Update-Conscious Parallel Directory Structure. In: International Conference on Data Engineering. (1999) 448–457

IX. AUTHORS

S. BhagyaRekha is currently pursuing M.Tech from Aurora's Engineering College, Bhongir. You may contact her at bhagyarekha2001@gmail.com

Dr.M.V.Vijaya Saradhi is Currently Professor in the Department of Information Technology (IT) at Aurora's Engineering College, Bhongir, Andhra Pradesh, India, where he teaches Several Courses in the area of Information Technology. You may contact him at meduri_vsd@yahoo.co.in