

Replication, Load Balancing and Efficient Range Query Processing in DHTs

Theoni Pitoura, Nikos Ntarmos, and Peter Triantafillou

R.A. Computer Technology Institute and
Computer Engineering & Informatics Dept.,
U. of Patras, Greece

4th Hellenic Data Management Symposium (HDMS'05)
Athens, Aug 26th, 2005

Overview

- Motivation
- Main ideas & Contributions
- Background: OP-Chord
- The HotRoD architecture
 - Replication and rotation
 - Data placement and definitions
 - Range query processing
- Experimental evaluation and results
- Conclusion

Our Goal

To ensure both **access load balance** and **efficient range query** processing in a DHT-based peer-to-peer network

Related Work/Motivation

□ Motivation:

- The naïve approach to deal with range queries in DHTs (i.e. individually querying each value in a range) is *highly expensive*.
- Other more “clever” approaches lead to access load imbalances in the presence of *skewed distributions*.
- Existing approaches dealing with both load balance and range queries use *“hot data” transfer*; however, data transfer does not solve access load imbalances in skewed access (query) distributions.
 - Andrzejak et al (2002), Gupta et al (2003), Ratnasamy et al (2001), Triantafillou et al (2003) etc: they support range queries, but suffer from access load imbalances
 - Ganesan et al (2004), Godfrey et al (2004), Karger et al (2004): they handle balancing in the presence of range queries, but not in the case of skewed access distributions (LB is based on transferring load).
 - Gopalakrishnan et al (2004): replication-based load balancing algorithm over Chord, using data propagation, which is extremely low; they only deal with exact match queries

Main Ideas - Contributions

- Our approach is based on two key ideas:
 - Use *locality-preserving data placement* (i.e. consecutive values stored on neighboring peers)
 - Use *replication* of popular values/ranges to fairly distribute access load among peers
- Our contributions are:
 - A *novel hash function*, both preserving ordering of values and handling data/range replication in DHT P2P networks
 - A tunable solution allowing for *trading off replication cost* for *fair load distribution* in the case of skewed data access distributions
 - *HotRoD*: A locality-preserving load-balancing DHT-based architecture

08/26/05

4th Hellenic Data Management Symposium

5

OP-Chord: Our Locality-Preserving DHT

- OP-Chord is a Chord-based architecture, extended to support *relations* and *range queries*
- Objects are database *tuples* of a k attribute relation $R(A_1, A_2, \dots, A_k)$, whose domain, DA_i is represented by $(v_{\min}(A_i), v_{\max}(A_i), \text{step}(A_i))$
- It uses an *order-preserving hash function*, $h_i(\cdot)$: for every value $v \in DA_i$

$$h_i(v) = \left(\frac{v - v_{\min}(A_i)}{v_{\max}(A_i) - v_{\min}(A_i)} \cdot 2^m + s_0(A_i) \right) \bmod 2^m$$

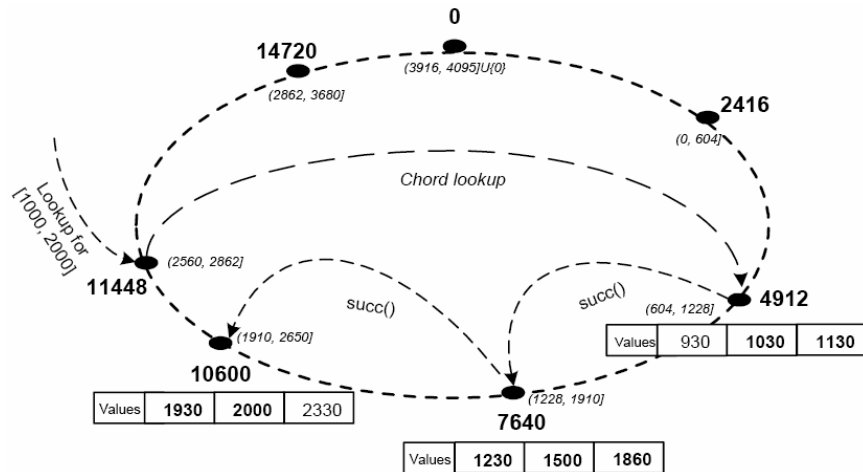
where $s_0(A_i) = \text{hash}(\text{attribute_name}(A_i))$, and $\text{hash}(\cdot)$ the base hash function used by Chord

08/26/05

4th Hellenic Data Management Symposium

6

OP-Chord: An Example



08/26/05

4th Hellenic Data Management Symposium

7

OP-Chord: Conclusions

- Data placement:
 - $O((k+1)\log N)$ routing hops to store a tuple
k: number of index attributes, N: number of peers
- Range query processing:
 - $O(\log N + n')$ routing hops for a range query $[v_{low}(A_i), v_{high}(A_i)]$
n': number of peers lying between peers p_{low} and p_{high} ,
where, $p_{low} = succ(h_i(v_{low}(A_i)))$ and $p_{high} = succ(h_i(v_{high}(A_i)))$
- Load Balancing:
 - Suffers from storage load imbalances in a single-attribute indexing scheme (not the case in multi-attribute indexing)
 - Suffers from access load imbalances in the case of skewed access distributions (like DHTs)

08/26/05

4th Hellenic Data Management Symposium

8

The HotRoD Architecture

- ❑ Built over a locality-preserving DHT, enhanced with a **novel hash function** which:
 - places objects/values in range partitions over the identifier space in an **order/locality-preserving** way, and
 - **replicates** hot (ranges of) objects/values to fairly distribute access load among peers
- ❑ Provides algorithms to detect overloaded peers and distribute access load fairly

08/26/05

4th Hellenic Data Management Symposium

9

Replication and Rotation

Hot Ranges/Rings of Data

- Each peer keeps track of the number of times, α , it was accessed during a time interval, and the average low -- *avgLow* -- and high -- *avgHigh* -- bounds of the respective (range) queries
- A peer is "hot" (or overloaded) when $\alpha > \alpha_{max}$, where α_{max} is the upper limit of its resource capacity
- An arc of peers is "hot" when at least one of its peers is hot
- ❑ "Hot" arcs of peers (or ranges of values) are replicated and rotated over the identifier space
- ❑ The HotRoD identifier space can be visualized as a number of **replicated**, **rotated**, and **overlapping virtual** rings

08/26/05

4th Hellenic Data Management Symposium

10

Replicating Arcs: Implementation Issues

- If a peer is detected “hot”, it starts replication
- Replicating arcs of peers, spec. the successive neighbors of p which correspond to the range [avgLow, avgHigh] to reduce costly jumps during range query processing
- The number of replicas created equals:
 $max(\alpha / \alpha_{avg_max}, \{\rho(v(A_i)), v(A_i) \in [avgLow, avgHigh]\})$
 - $\rho(v(A_i))$ is the current number of replicas of value $v(A_i)$ (replication factor)
 - $\rho_{max}(A_i)$ is the max number of instances of a value of attribute A_i

Multi-Rotation Hash Function

- Multi-rotation hash function (MRHF):

$$mrhf(v, \delta) = (hash(v) + random_i[\delta] \cdot s) \bmod 2^m$$

- Notation:
 - Objects are database tuples of a k attribute relation $R(A_1, A_2, \dots, A_k)$, with domains DA_i , for $i \in \{1, 2, \dots, k\}$
 - $hash(v)$ is the hash function of the underlying DHT
 - $\rho_{max}(A_i)$ is the max number of instances of each value of an attribute A_i
 - $\delta \in [1, \rho_{max}(A_i)]$ is an index variable to distinguish the different instances of A_i 's values
 - $s = (1 / \rho_{max}(A_i)) \cdot 2^m$ is the rotation unit
 - $random_i[1 \dots \rho_{max}(A_i) + 1]$ is a pseudo-random permutation of the integers in $[1, \rho_{max}(A_i)]$ and $random_i[1] = 0$

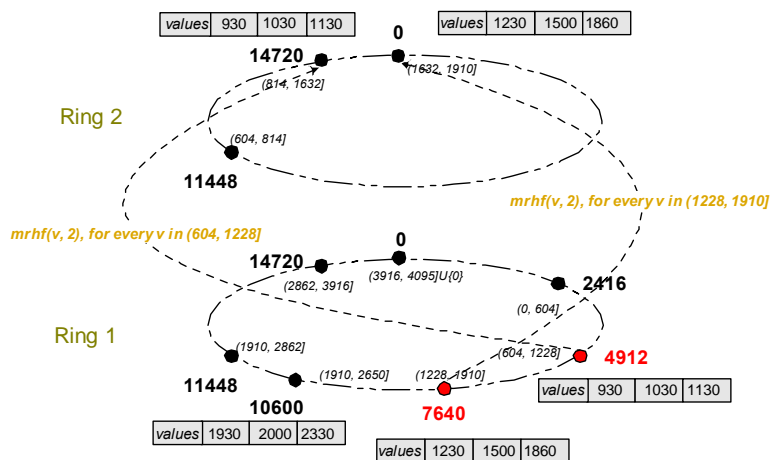
Multi-Rotation Hash Function (cont.)

- Multi-rotation hash function (MRHF):

$$mrhf_i(v, \delta) = (hash(v) + random_i[\delta] \cdot s) \bmod 2^m$$

- $\delta=1$ (no replicas):
 - v is placed on peer $p = mrhf_i(v, \delta) = hash(v)$ according to the underlying DHTs hash function
- $\delta > 1$:
 - The δ^{th} instance of v is created (i.e. the $(\delta-1)^{\text{th}}$ replica)
 - Placed on the peer whose identifier is "closer" to $hash(v)$, shifted by $\delta \cdot s$ clockwise
 - δ also viewed as the number of rotations, with s being the displacement of every rotation

Data Placement: An Example



Implementation Issues

- **GetRho($v(A_i)$)** gets the current number of replicas of a value $v(A_i)$, i.e. the replication factor $\rho(v(A_i))$
- **PutRho($v(A_i)$, ρ)** sets the replication factor of $v(A_i)$ equal to ρ
- Functions **GetRho($v(A_i)$)** and **PutRho($v(A_i)$, ρ)** manipulate $\rho(v(A_i))$ over the network using the underlying DHT

08/26/05

4th Hellenic Data Management Symposium

15

Range Query Processing

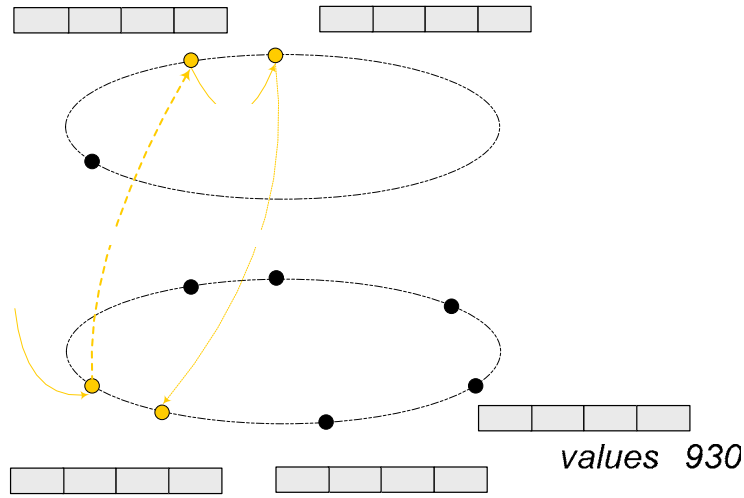
- Let a range query $R = [v_{\text{low}}(A_i), v_{\text{high}}(A_i)]$ initiated at peer p_{init}
 - p_{init} randomly selects a number, r from 1 to $\rho(v_{\text{low}}(A_i))$, which is the current number of replicas of $v_{\text{low}}(A_i)$
 - p_{init} forwards the range query to $\text{succ}(\text{mrhf}_i(v_{\text{low}}(A_i), r))$, say p_l
 - p_l searches for matching tuples in R and forwards the query to its successor, p ;
 - p repeats similarly as long as it finds replicas of values of R at the current ring
 - Otherwise, p forwards the range query to a randomly selected lower-level ring, and repeats
 - Processing is finished when all values of R have been looked up

08/26/05

4th Hellenic Data Management Symposium

16

Range Query Processing: An Example



08/26/05

4th Hellenic Data Management Symposium

17

Ring 2

Experimental Evaluation

- Experiments
 - We implemented HotRoD using the Chord simulator
 - We compared HotRoD against:
 - Plain Chord (PC)
 - An imaginary enhanced Chord (EC)
 - OP-Chord
- Results are presented in terms of:
 - Efficiency of query processing (#of hops)
 - Access load balancing (cumulative access load distribution curves, Gini coefficient)
 - Overhead costs (#of peers, #of tuple replicas)
- Simulation model
 - N=1,000 peers, 10 (=logN) fingers, 10 immediate successors, 10,000 values in DA_i , 5,000 tuples
 - 20,000 range queries
 - Range span ranging from 1 to 400 values (i.e. 0.01%-4% of DA_i)
 - Zipf query distribution over DA_i with $\theta=0.5, 0.8, 1.0$
 - Number of HotRoD rings ranging from 2 to 150

08/26/05

4th Hellenic Data Management Symposium

18

Performance Evaluation Results: Efficiency of Range Queries

- ❑ HotRoD is more expensive against OP-Chord, due to GetRho() operations
- ❑ HotRoD ensures savings from 4% to 78% (in #of hops) against EC for different range spans
- ❑ As the range span increases, OP-Chord / HotRoD improve their performance against PC / EC

r	50	100	200	400
PC	123	246	489	898
EC	25	48	87	190
OP-Chord	18	20	25	33
HotRoD ($\rho_{max}=30$)	24	27	31	41

Average number of hops per query for different range spans r
(Zipf parameter $\theta=0.8$)

08/26/05

4th Hellenic Data Management Symposium

19

Performance Evaluation Results: Access Load Balancing

- ❑ Results are presented in terms of:
 - Lorenz curves: functions of the cumulative proportion of peers, ordered by their load (i.e. hits), mapped onto the corresponding cumulative proportion of their load.
 - ❑ If all peers have the same load, the curve is a straight line, the line of uniformity
 - The Gini coefficient: a summary of the total amount of load imbalances, expressed by:

$$G = \sum_{i=1}^n (2i - n - 1) \cdot l_i / n^2 \cdot \mu,$$

- ❑ μ is the mean load
- ❑ G takes values from 0 to 1; 0 when all peers in the network have equal load; 1 when all peers except one have zero load
- ❑ In real terms, in a fair load distribution, G ranges from 0.5 to 0.65, whereas in a very unfair load distribution, it ranges from 0.85 to 0.99

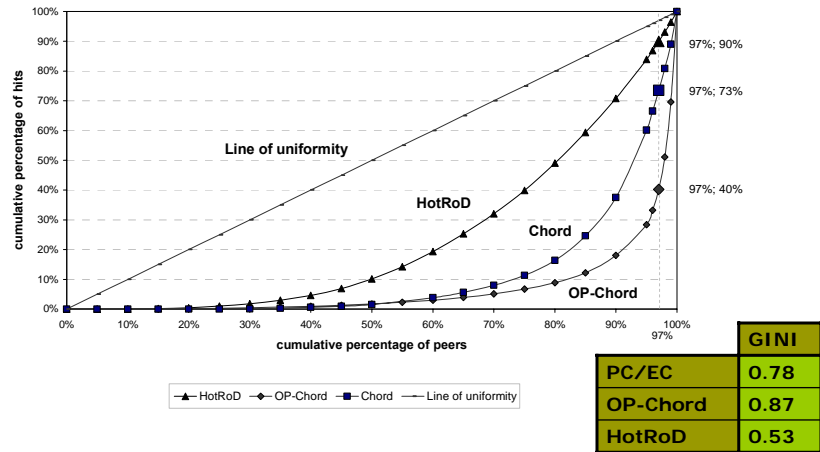
08/26/05

4th Hellenic Data Management Symposium

20

Performance Evaluation Results: Access Load Balancing (cont.)

Lorenz Curves for Access Load Distribution
($r = 200, \theta = 0.8$)



08/26/05

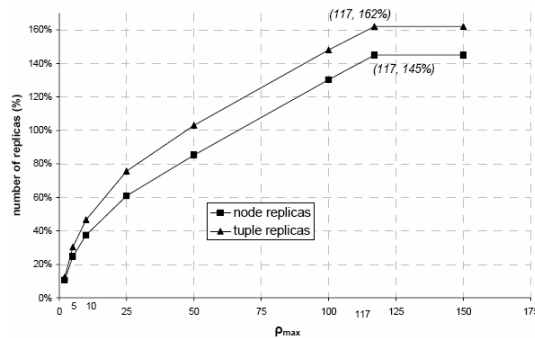
4th Hellenic Data Management Symposium

21

Performance Evaluation Results: Overhead Costs

Tuning replication degree by ρ_{\max}

Number of Replicas per ρ_{\max}
($r=50, \theta=0.8, \alpha_{\max}=100$)



- As ρ_{\max} increases, the number of peers' and tuples' replicas are increased till an upper bound
- High values of ρ_{\max} provide diminished returns in load balancing, although the degree of replication is very high
- Therefore, we can achieve a very good access LB with low values of ρ_{\max}

08/26/05

4th Hellenic Data Management Symposium

22

Conclusions

- The proposed approach concurrently attacks two key problems:
 - Efficient range query processing
 - Access load balance, through replication
- Our achievements:
 - A significant hop count saving in range query processing with different range query spans, ranging from 5% to 80% comparing with standard DHTs
 - A data replication less than 100% leads to significant more fairly distributed access load
 - Greater benefits for greater range query spans, and/or higher access skew
- HotRoD achieves very good hop count efficiency coupled with a significant improvement in the overall access load distribution among peers

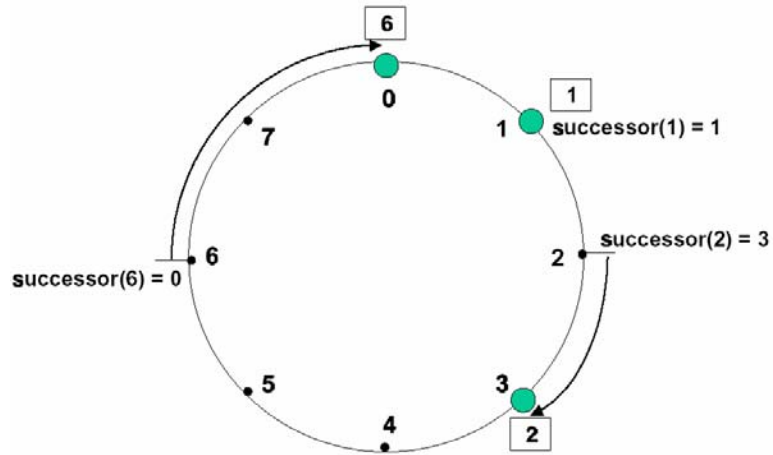
Thank you!

Backup slides

Background: Chord

- Chord is a scalable distributed look-up service:
 - It uses an m -bit identifier space ordered in a mod- 2^m circle, the **Chord ring**
 - It maps peers and objects to identifiers in the Chord ring, using the hash function *SHA-1*
 - It uses **consistent hashing**:
 - an object with identifier id is placed on the successor peer, $succ(id)$, s.t. its identifier is $\geq id$, on the Chord ring
 - Each peer n maintains routing information about peers that lie on the Chord ring at **logarithmically increasing distance** from them
 - Routing a message among two peers requires $O(\log N)$, N the number of peers

Background: Chord (2)

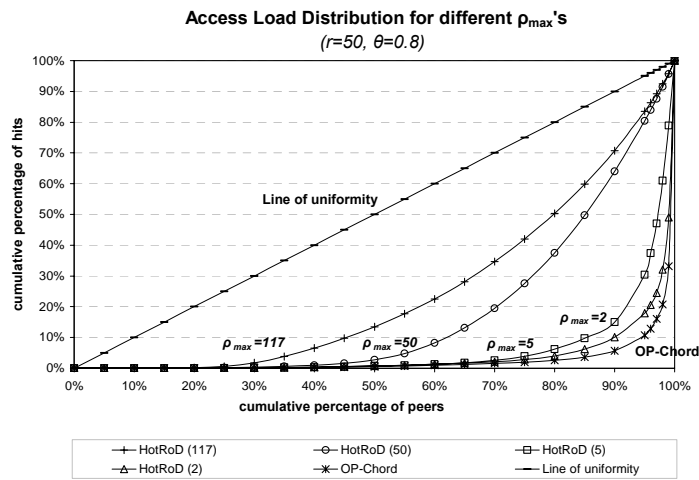


08/26/05

4th Hellenic Data Management Symposium

27

Results: the role of ρ_{max}



08/26/05

4th Hellenic Data Management Symposium

28

Replication and Rotation (cont.)

- $\delta=1$ (no replicas):
 - v is placed on peer $p=mrhf_i(v, \delta) = h_i(v)$ according to the underlying DHTs hash function
- $\delta>1$:
 - The δ^{th} instance of v is created (i.e. the $(\delta-1)^{\text{th}}$ replica)
 - Placed on the peer whose identifier is "closer" to $h_i(v)$ **shifted by $\delta \cdot s$ clockwise**
 - δ also viewed as the number of rotations, with s being the displacement of every rotation

$$mrhf_i(v, \delta) = \begin{cases} \left(\frac{v - v_{\min}(A_i)}{v_{\max}(A_i) - v_{\min}(A_i)} \cdot 2^m + s_0(A_i) \right) \bmod 2^m, \delta = 1 \\ \left(\frac{v - v_{\min}(A_i)}{v_{\max}(A_i) - v_{\min}(A_i)} \cdot 2^m + \text{random}_i[\delta] \cdot s \right) \bmod 2^m, \delta > 1 \end{cases}$$

08/26/05

4th Hellenic Data Management Symposium

29

Related Work

- Range Query Processing & Load Balancing
 - Andrzejak et al (2002), Gupta et al (2003), Ratnasamy et al (2001), Triantafillou et al (2003) etc: they support range queries, but suffer from access load imbalances
 - Ganesan et al (2004), Godfrey et al (2004), Karger et al (2004): they handle balancing in the presence of range queries, but not in the case of skewed access distributions (LB is based on transferring load).
 - Gopalakrishnan et al (2004): they use a replication-based load balancing algorithm over Chord, through propagation, which is extremely low; they only deal with exact match queries

08/26/05

4th Hellenic Data Management Symposium

30