casa Engineering Research Center for
Collaborative Adaptive Sensing of the Atmosphere

Exploiting Communities for Enhancing Lookup Performance in Structured P2P Systems

H. M. N. Dilum Bandara and Anura P. Jayasumana

Colorado State University

Anura.Jayasumana@ColoState.edu

University of Massachusetts Amherst

University of Oklahoma

Colorado State University

University of Puerto Rico Mayaguez

CASA is primarily supported by the Engineering Research Centers Program of the National Science Foundation under NSF award number 0313747.

Contribution

Community-aware caching scheme to enhance lookup performance in structured P2P systems

1. Build sub-overlays among community members while preserving overlay properties
2. Weighted least frequently used caching based on local statistics

- Enhances both communitywide (23-51%) & system-wide lookup (40%) performance
- Works with structured P2P systems that provide alternative paths to a given destination
- Works with any skewed popularity distribution
- Adaptive to changing popularity
- Need small caches

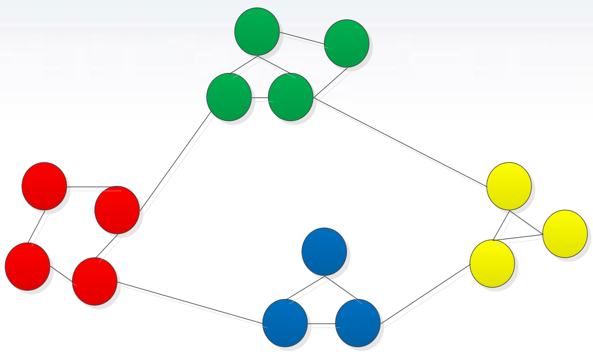
Motivation

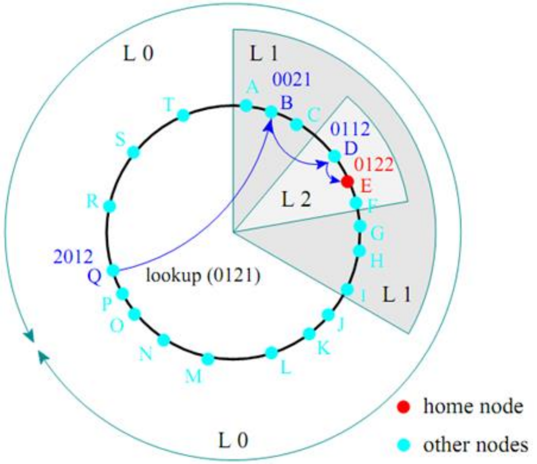
- Many small communities are emerging within P2P systems
- Community – subset of peers that share some similarity
  - Semantic
    - Many BitTorrent communities – music, movies, games, Linux distributions, private communities
  - Geography
    - For 60% of files shared by eDonkey peers, more than 80% of their replicas were located in a single country [Handurukande, 2006]
  - Organizational
    - Peers within an AS, members of a professional organization, group of universities
    - To share resources & limit unrelated external traffic

S. B. Handurukande et al., “Peer sharing behaviour in the eDonkey network, and implications for the design of server-less file sharing systems,” EuroSys ‘06, Apr. 2006.

Motivation (cont.)

- Content popularity in P2P follows Zipf's-like distribution
- Improve lookup
  - Restructure overlay based on similarity
  - Cache most globally popular content
- However
  1. Communities are not isolated
  2. Individual communities don't rank high in popularity
  3. Not every node can or interested in caching



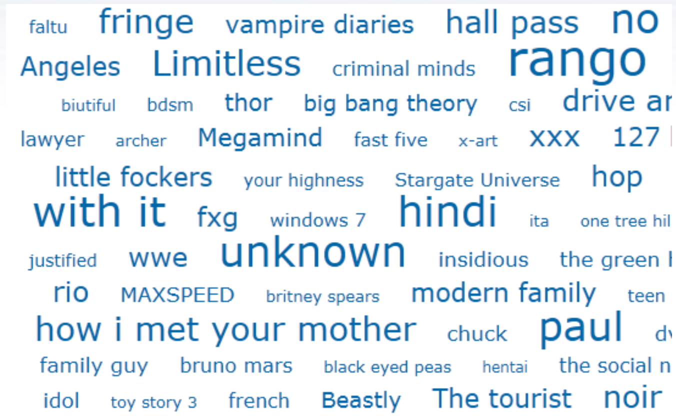


[Ramasubramanian, 2004]

Ramasubramanian and Sirer, "Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays," USENIX NSDI '04, 2004.

Content Popularity in Communities

1. Communities are not isolated



BitTorrent Communities

Community*	EX	FE	SP	TB	TS	TE	TR
fenopy.com (FE)	0.38						
seedpeer.com (SP)	0.00	0.00					
torrentbit.net (TB)	0.40	0.29	0.00				
torrentscan.com (TS)	0.48	0.33	0.00	0.48			
torrentsection.com (TE)	0.53	0.23	0.00	0.31	0.25		
torrentreactor.net (TR)	0.10	0.08	0.00	0.06	0.09	0.06	
youbittorrent.com (YB)	0.36	0.35	0.00	0.29	0.42	0.20	0.04

EX – extratorrent.com

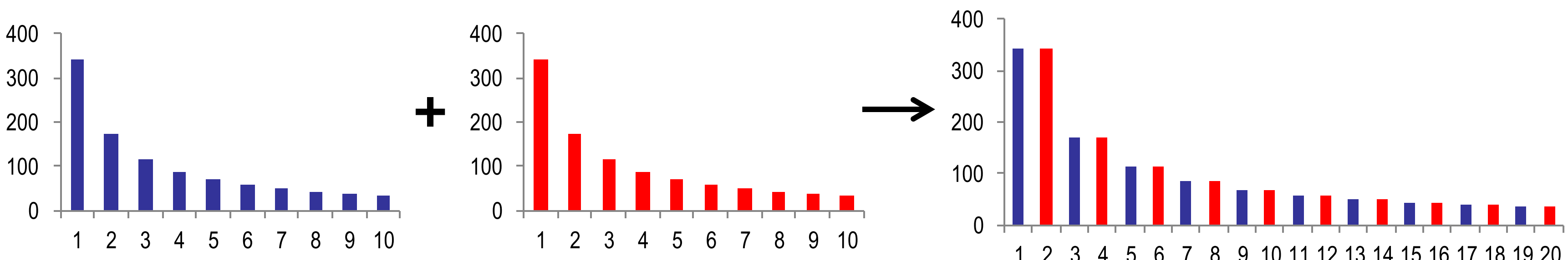
Content Popularity in Communities (cont.)

2. Communities have different Zipf's parameters

$$f_r = \frac{1/r^{\alpha}}{\sum_{n=1}^{N} 1/n^{\alpha}}$$

   - α = 0.53, 0.66, 0.79, 0.98
   - Aggregation of multiple Zipf's distributions is not necessarily Zipf
   - Caching on a structured P2P system with alternative paths [Rao, 2007]

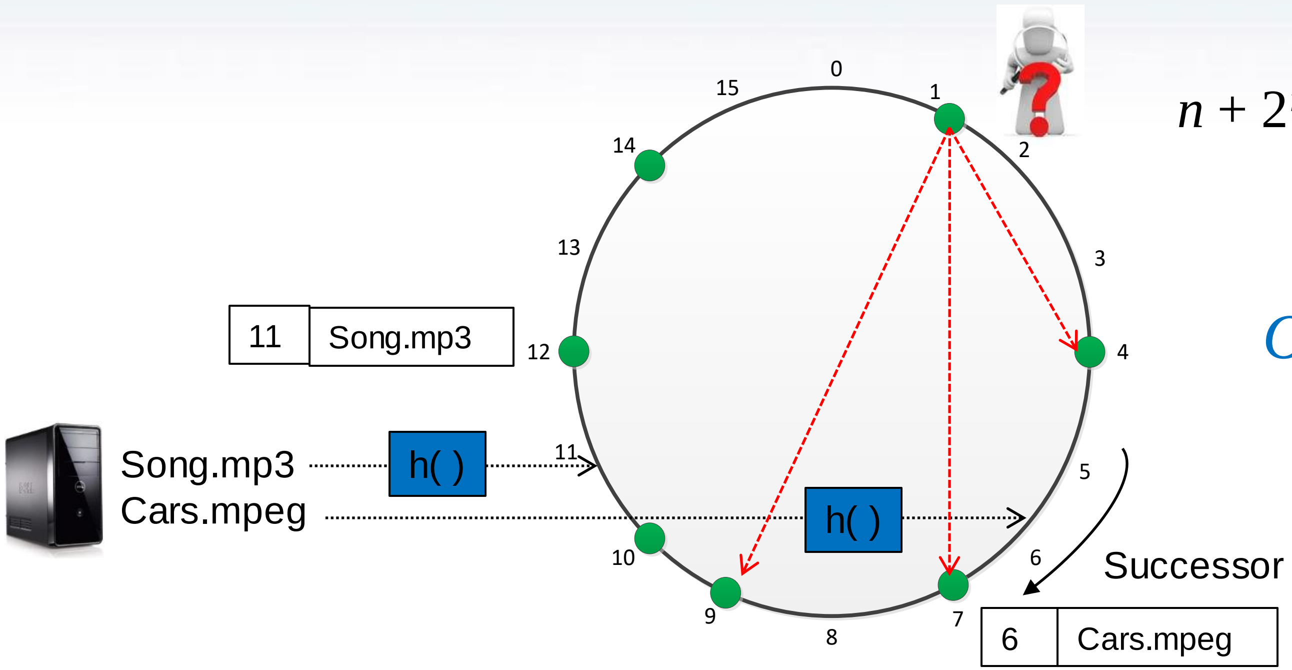
$$H = \log N - \sum_{r=1}^{C} f_r \log f_r - \log_k L$$



α = 1.0 α = 1.0

W. Rao et al., "Optimal proactive caching in peer-to-peer network: analysis and application," 6th ACM Con. on Information and Knowledge Management, Nov. 2007.

Structured Overlay – Chord DHT

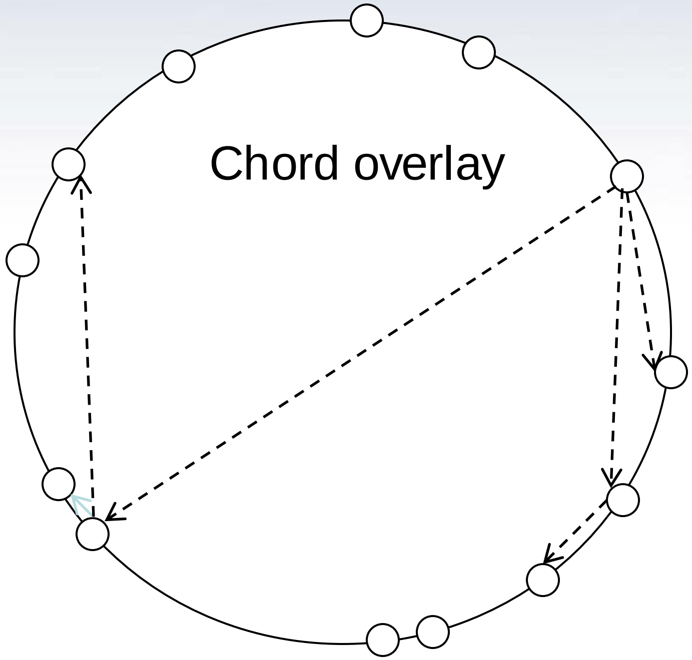


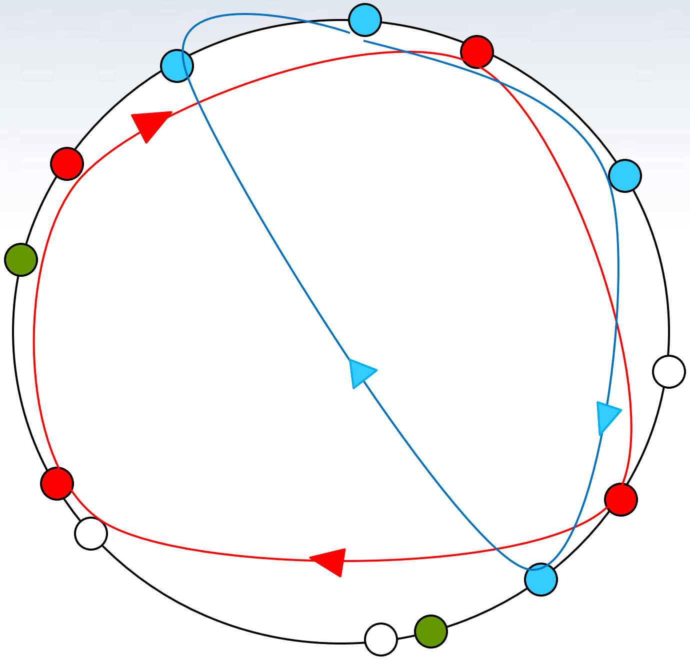
$n + 2^{i-1}, 1 \leq i \leq m$

$O(log\ N)$ hops

I. Stoica et al., "Chord: a scalable peer-to-peer lookup service for internet applications," ACM SIGCOMM '01, Aug. 2001.

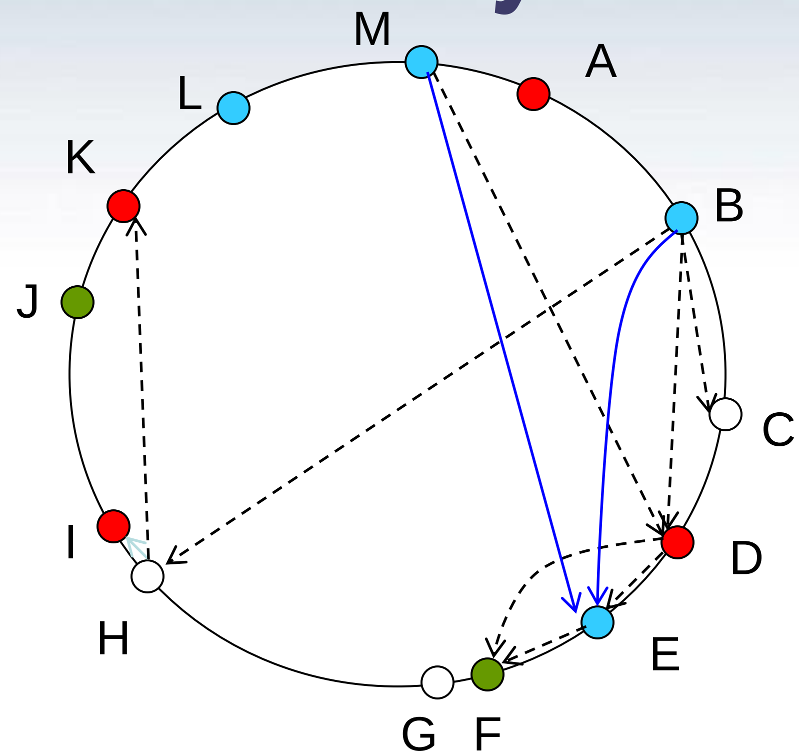
Sub-Overlay Formation





- Goal – not to isolate communities or mix contents
- Each community forms a sub-overlay
  - Form links/fingers to community members
- Enable nodes to identify what's popular in their community & cache accordingly
  - Forward queries to community members hoping that they may have already cached required contents

Sub-Overlay Formation (cont.)



B → D → F = 2 hops
B → E → F = 2 hops

If E cache F's content
B → E = 1 hop

No of distinct node found by probing i-th finger & it's successor

$$2(i + 2\log N - m) - 1$$

N – No of nodes
m – Key length
$1 \le i \le m$

- Nodes have 1 or more community IDs
  - Communities based on different similarity measures – semantic, geography
  - Support exceptions – user in USA can be a member of a community in India
- Identify community members that are at an exponentially increasing distances in key space
  - Sample nodes pointed by links & their successors
  - Long distant links (large i) are more important & easy to find

Caching Algorithm

- Cache based on community interest
  - Queries go through community members → Nodes get to know what's popular in their community
- Local statistics are sufficient to estimate relative popularity
  - Focus on community interest
  - No assumption on popularity distribution
- Weighted least frequently used caching
  - Evaluate demand at arrival of each query q → Adaptive
  - Weight α determine bias towards short or long term trends

$$\begin{cases} demand_i^k = (1+\alpha)\times demand_{i-1}^k & \text{If } q \text{ is for } k \\ demand_i^k = (1-\alpha)\times demand_{i-1}^k & \text{else} \end{cases} \qquad 0 \le \alpha \le 1$$

  - If $demand^k > D_{cache}$ – Indicate node's interest to cache by append to query q
- Query response is send to query originator & all nodes that want a copy to cache

Caching Algorithm (cont.)

- Reevaluates what keys to cache at arrival of a query
  - Naturally adapts to varying trends of community interests
  - Computationally efficient
- Track contents even if not cached
  - Threshold to remove least popular ones
- D_{cache} – Caching threshold
  - Prevents cache thrashing
  - $D_{cache} > \alpha$

```
void forward(key, msg, nextHop*)
    If msg.type = PUT                           //put message
        return
    If msg.type = GET                           //get message
      addLookup(key)                            //Track demand
      If key ∈ C                                //In cache
        sendDirect(msg.source, key, C[key])
        For each i in msg.cList[ ]              //Send to each cache requester
          sendDirect(msg.cList[i], key, C[key])
        nextHop ← NULL                          //Drop original get message
      Else                                      //Not in cache
        If C.size( ) = C_max                    //Cache already full
          key_lowest ← getCachedKeyWithLowestDemand(L[ ] )
          If L[key] > L[key_lowest]             //Higher demand
            msg.cList[ ] ← myNodeID             //Request a copy
            C[key_lowest]. remove               //Remove lowest key
        Else
          If L[key] > D_cache                   // Higher demand
            msg.cList[ ] ← myNodeID             //Request a copy

void addLookup(key)
    For each i in L[ ]
      If i = key                                //Increase demand for key
        L[i] = (1 + α) × L[i]
      Else                                      //Decrease demand for others
        L[i] = (1 – α) × L[i]
      If L[i] < D_remove                        //Very low demand
        L[i].remove                             //Remove key
```

Simulation Setup

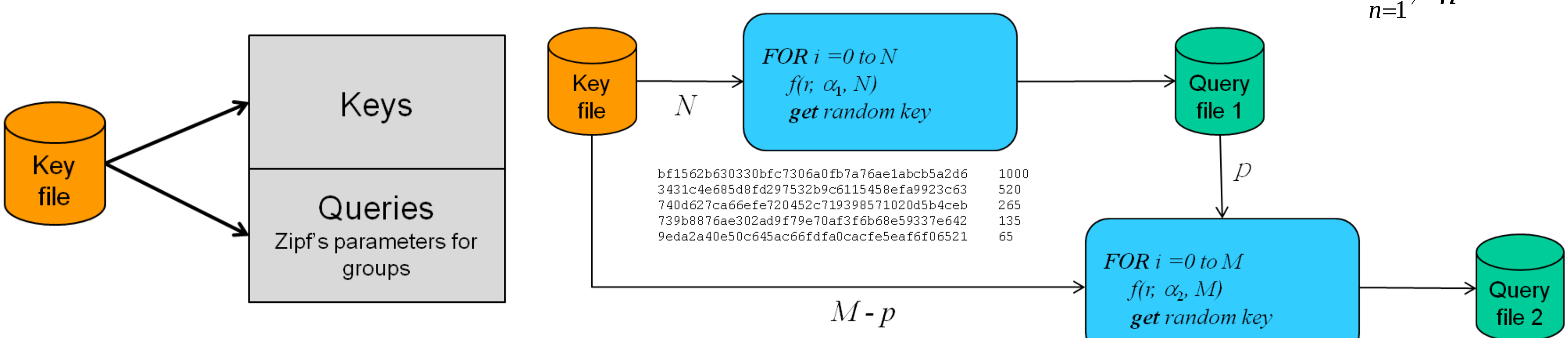
Community	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}
No of nodes (apx.)	600	600	600	1,200	1,200	1,200	1,200	1,200	2,400	4,800
Zipf's parameter	0.85	0.95	1.10	0.5	0.80	0.80	1.0	0.90	0.90	0.75
No of distinct keys	40,000	30,000	30,000	40,000	40,000	40,000	50,000	50,000	50,000	50,000
Similarity with community (*x*)	0.2 (C_8)	0	0.1 (C_7)	0.2 (C_9)	0.3 (C_8) 0.5 (C_7)	0	0.1 (C_3) 0.5 (C_5)	0.3 (C_5) 0.2 (C_1)	0.4 (C_1) 0.2 (C_4) 0.3 (C_{10})	0.3 (C_9)
Queries for rank 1 key	4,516	8,535	17,100	603	6,454	6,454	21,059	11,956	23,911	17,030

- OverSim P2P simulation environment
- Sub-overlay formation & caching implemented on top of Chord overlay
- 15,000 nodes
- 10 communities of different sizes
- Different Zipf's parameters
- Queries after system got stabilized – around 2000 sec
- 10 samples

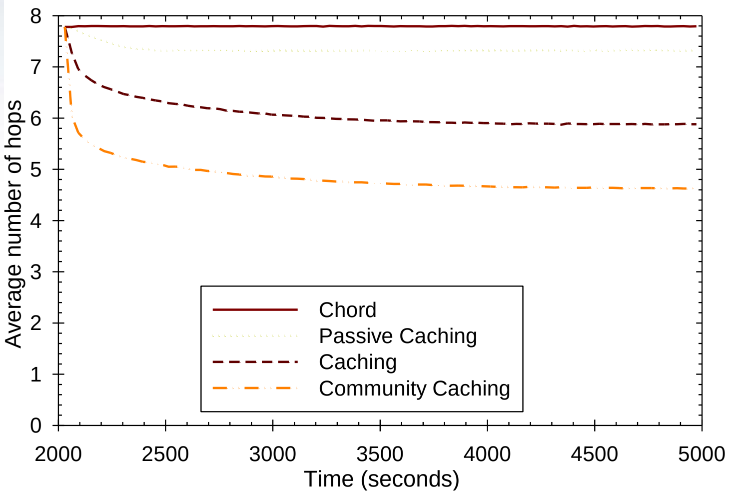
Community, Keys & Query Generation

- Peers know their group ID at initialization
- Each peer
  - Maintain a key index – no capacity limit
  - Maintain a cache – fixed capacity
- Generate fixed set of keys a-priory
  - Peers read keys from a file & store in appropriate nodes
- Queries
  - Use set of Zipf's parameters observed form BitTorrent

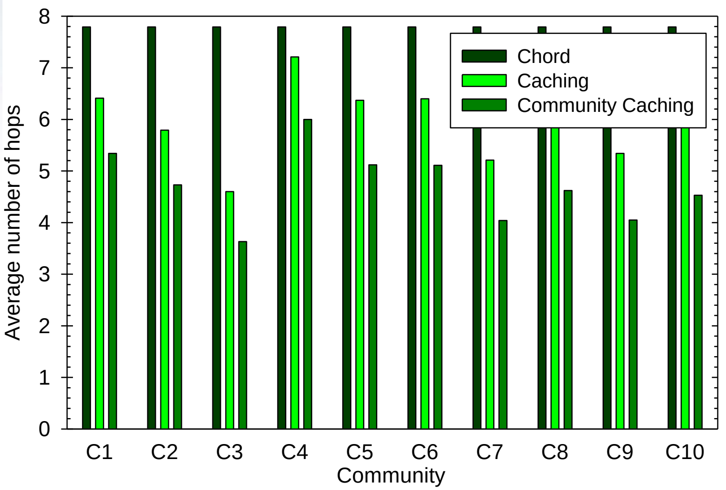
$$f(r,\alpha,N) = \frac{1/r^{\alpha}}{\sum_{n=1}^{N} 1/n^{\alpha}}$$



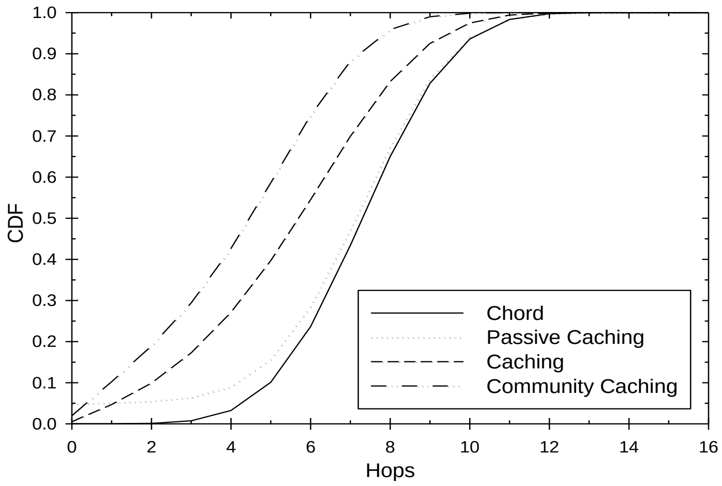
Performance Analysis



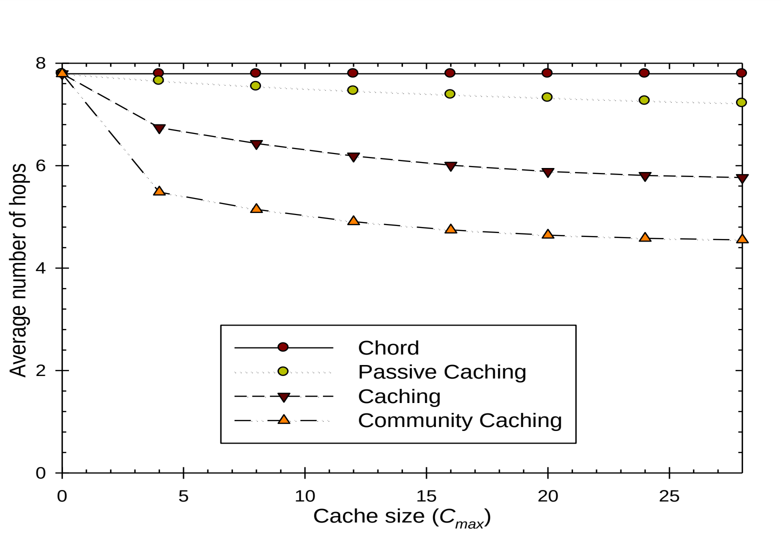
$D_{cache} = 0.12$
$\alpha = 0.1$
$C_{max} = 20$

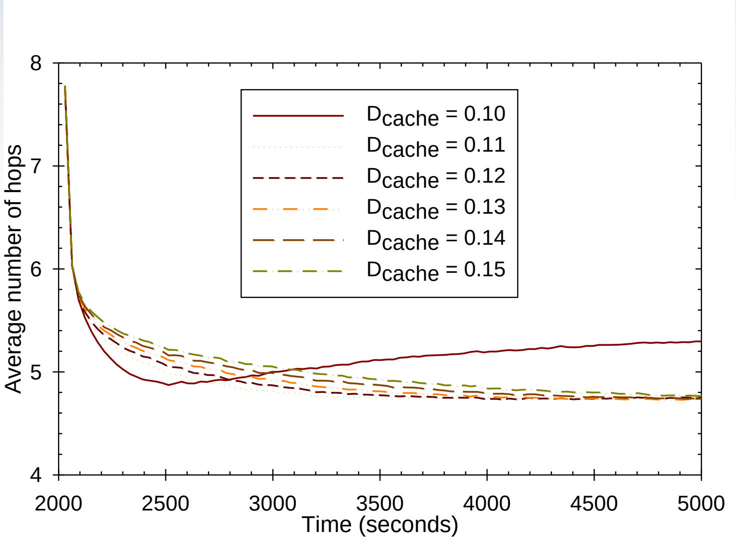


- Reduced path length
  - Overall system – 40.5%
  - More popular communities – 48-53%
  - Least popular community – 23% reduction (7% with caching)
- Performance depends on skewness
  - C_1, C_5, & C_6
- Most queries are responded within few hops

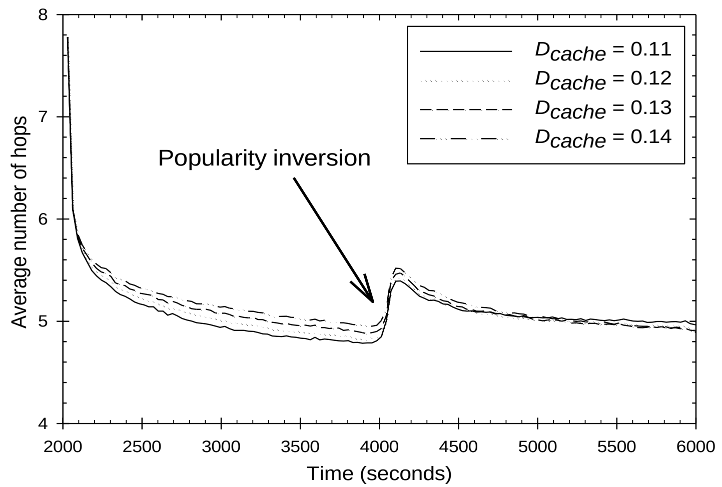


Performance Analysis (cont.)





- Small cache size per node
- D_{cache} reduce cache thrashing, overhead ,& long-term path length
- Rapidly respond to popularity changes
- Better load distribution
  - Max with Chord – 27,574
  - Max with Community Caching – 1,677



Summary

- Community-aware caching solution for structured P2P
  - Allows queries to be forwarded through community members
  - Enable nodes to cache resources that of interest to their community
- Properties
  - Improve both communitywide & system-wide performance
  - Works with any structured P2P system that provides alternative paths to a given destination
    - Preserve overlay bound $O(\log N)$
  - Independent of popularity distribution & how communities are formed
  - Based on local statistics
  - Adaptive
  - Introduces minimum cache storage, network, & computational overhead
- Current/future work
  - Analyze performance under peer churn, heterogeneous caches, & geography based communities
  - In-network community identification & formation

Engineering Research Center for
Collaborative Adaptive Sensing of the Atmosphere

Questions ?

Anura.Jayasumana@ColoState.edu
www.cnrl.colostate.edu

CASA is primarily supported by the Engineering Research Centers Program of the National Science Foundation under NSF award number 0313747.