Optimizing File Availability in P2P Content Distribution

Jussi Kangasharju
University of Helsinki
TU Darmstadt

Keith W. Ross
Brooklyn Polytechnic

David A. Turner
CSU San Bernardino

P2P Content Management Problem

- A community of peers access a set of files
  - Peers members of a DHT-based file sharing community
  - Large, popular files, e.g., media or software
- Goals and challenges:

1. Adaptively manage content to minimize download delay
   - Assume downloads in community are fast
   - Hence, roughly equivalent to maximizing hit rate in community
2. Design a simple, yet efficient algorithm to address:
   - Replication
   - File replacement
   - Load balancing

Why Replication?

- Peer-to-peer systems based on unreliable peers
- Need for building reliable services on top of peers
- Simple answer: Replication

Replication benefits:

- Improves availability and level of service
- "Easy" to implement

Replication problems:

- Creating and managing additional copies is costly
- Consistency problems with modifiable content

Replication Issues

Main questions with replication:

1. What do we want to achieve?
   - For example, availability of X nines?
2. How many copies are needed?
3. How many copies we can afford?
4. Where to put copies?
5. Did we achieve our goal?
6. Is 100% guaranteed availability possible?

- Yes, at least in some cases… ;-)
  - But probably never in practice

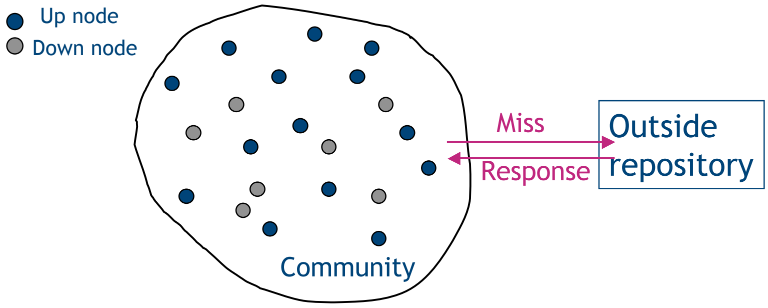
Contributions

1. Main contribution:
   - Set of adaptive algorithms for dynamically replicating and replacing files in a P2P community
   - Optimal replication theory for P2P communities
   - No assumptions about nodes or node behavior, or file request probabilities
   - Algorithms are simple, adaptive, and fully distributed
   - Top-K MFR algorithm can be shown to be near-optimal
2. Second contribution:
   - Investigation of load balancing techniques for P2P communities
   - Without any load balancing, load concentrates on a few nodes
   - Fragmentation approach achieves a general load balance
   - Overflow approach allows for individual variation
   - Both shown to be very effective

Outline

- Community model
- Optimization theory
- Simple algorithms and evaluation
- Most Frequently Requested Algorithm and evaluation
- Load balancing
  - Fragmentation approach
  - Overflow approach
- Summary

Abstract Community Model



- Examples of communities: Campus, distribution engine
- Assume good bandwidth within community
- Goal: Satisfy requests from within community

Replication Issues

- How many copies of each object in community?
- Which peers in community have copies?
- Is there an algorithm that is:
  - simple
  - decentralized
  - adaptively replicates objects
  - provides near-optimal replica profile?

Assumptions

- Community based on a distributed hash table (DHT)
  - Any existing DHT can be used or modified
- Assume that when given an object, DHT gives us an ordering of nodes (i.e., which nodes are responsible)
  - First node is 1st place winner, second 2nd place winner, etc.
- Peers are up with a certain probability (up probability)
- Peers offer some amount of space for community
- File popularities follow Zipf-like distribution

Replication Theory

- J objects, I peers
- object j
  - requested with probability q_j
  - size b_j
- peer i
  - up with probability p_i
  - storage capacity S_i
- decision variable
  - $x_{ij} = 1$ if a replica of j is put in i; 0 otherwise

- Goal: maximize hit probability in community (availability)
- Extension to byte hit probability is possible

Optimization Problem

Minimize $$\sum_{j=1}^{J} q_j \prod_{i=1}^{I} (1-p_i)^{x_{ij}}$$

subject to $$\sum_{j=1}^{J} b_j x_{ij} \le S_i, \quad i = 1,\ldots,I$$

$$x_{ij} \in \{0,1\}, \quad i = 1,\ldots,I, \quad j = 1,\ldots,J$$

Can be reduced to Integer programming problem: NP

Homogeneous Up Probabilities

- Suppose $p_i = p$
- Let $n_j = \sum_{i=1}^{I} x_{ij}$ = number of replicas of object j
- Let S = total group storage capacity
- Minimize $\sum_{j=1}^{J} q_j (1-p)^{n_j}$
- subject to: $\sum_{j=1}^{J} b_j n_j \le S$

Can be solved by dynamic programming

Extension: Erasure Codes

- Above theory considers only full replicas
  - Number of copies must be an integer
- Removing this restriction gives us an upper bound
- Upper bound for hit-rate with erasure coding is derived in paper
- Upper bound can also be used for case without erasures
  - Details in paper
- Optimal number of copies (non-integer!) turns out to be as follows…

Optimal Replication

(1) Order objects according to q_j / b_j

(2) There is an L such that $n^*_j = 0$ for all $j > L$.

(3) For $j <= L$, "logarithmic replication rule":

$$n_j^* = \frac{S}{B_L} + \frac{\sum_{l=1}^{L} b_l \ln(q_l / b_l)}{B_L \ln(1-p)} + \frac{\ln(q_j / b_j)}{\ln(1/(1-p))}$$

$$= K_1 + K_2 \ln(q_j / b_j)$$

Logarithmic replication rule

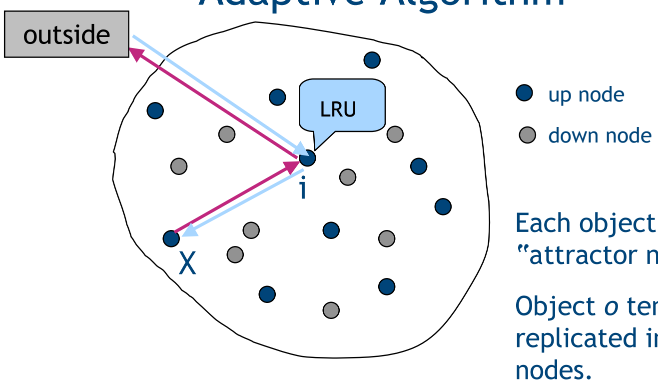
Adaptive Algorithm: Simple Version

Suppose *X* is a node that wants object *o*.

1) *X* uses DHT to find 1st-place up node *i* for *o*
2) *X* asks *i* for *o*
3) If *i* doesn't have *o*, *i* retrieves *o* from the "outside" and stores a copy in its shared storage.
4) *i* sends *o* to *X*

Each node uses LRU replacement policy in shared storage

Adaptive Algorithm



Each object *o* has "attractor nodes"

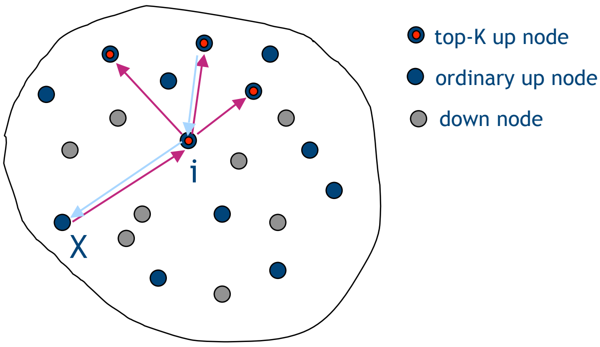
Object *o* tends to get replicated in its attractor nodes.

Queries for *o* tend to be sent to attractor nodes.
➡ tend to get hits

Problem: Can miss even though object is in an up node in the community

Ubiquitous Peer-to-Peer Infrastructures Group
Department of Computer Science

Top-K Algorithm



- If *i* doesn’t have *o*, *i* pings top-K winners.
- *i* retrieves *o* from one of the top-K if present.
- If none of the top-K has *o*, *i* retrieves *o* from outside.

03.06.2007 17

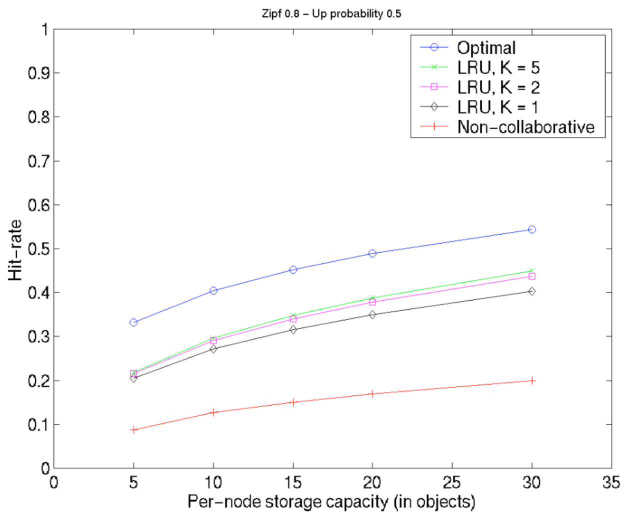
Ubiquitous Peer-to-Peer Infrastructures Group
Department of Computer Science

Simulation

- Adaptive and optimal algorithms
- 100 nodes, 10,000 objects
- Zipf = 0.8, 1.2
- Storage capacity 5-30 objects/node
  - Focus on large files, hence small storage capacity
- All objects the same size
  - Heterogeneous sizes yield similar results
- Up probabilities 0.2, 0.5, and 0.9
- Top K with K = {1, 2, 5}

03.06.2007 18

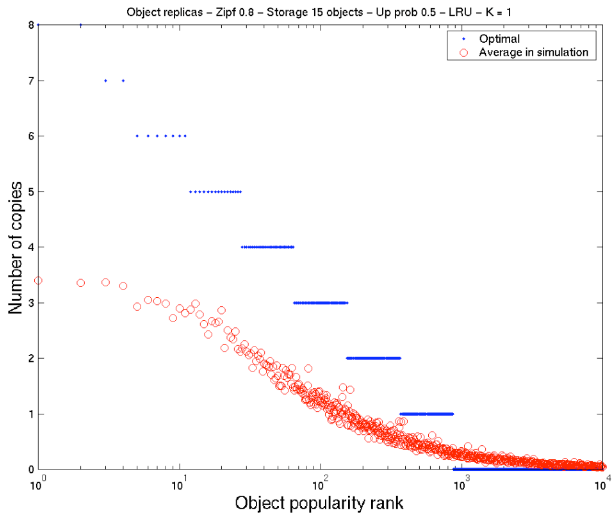
Hit-Probability vs. Node Storage



p = P(up)
= .5

Zipf = .8

Number of Replicas



p = P(up)
= .5

15 objects
per node

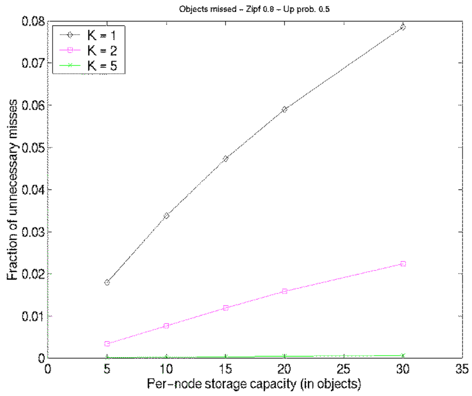
K = 1

Zipf = .8

Ubiquitous Peer-to-Peer Infrastructures Group
Department of Computer Science

General observations

- Community improves performance significantly
- LRU is lets unpopular objects linger in peers
- Top-K algorithm is needed to find object in aggregate storage (see right)



How can we do better?

03.06.2007 21

Ubiquitous Peer-to-Peer Infrastructures Group
Department of Computer Science

Most Frequently Requested (MFR)

- Each peer estimates local request rate for each object
  - Denote $\lambda_o(i)$ for rate at peer *i* for object *o*
- Peer only stores the most requested objects
  - Packs as many objects as possible

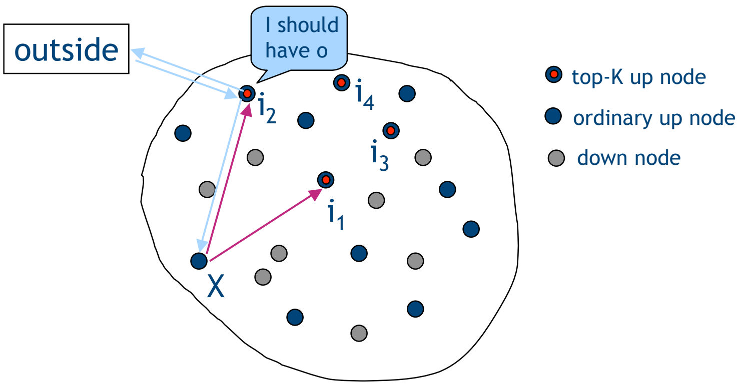
Suppose *i* receives a request for *o*:

- *i* updates $\lambda_o(i)$
- If *i* doesn't have *o* & MFR says it should:

  i retrieves *o* from the outside

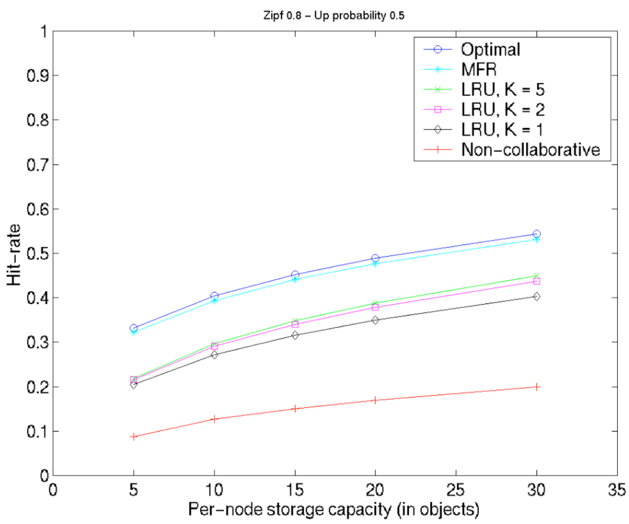
03.06.2007 22

Most-Frequently-Requested Top-K Algorithm



MFR combines replacement and admission policies

Hit-Probability vs. Node Storage



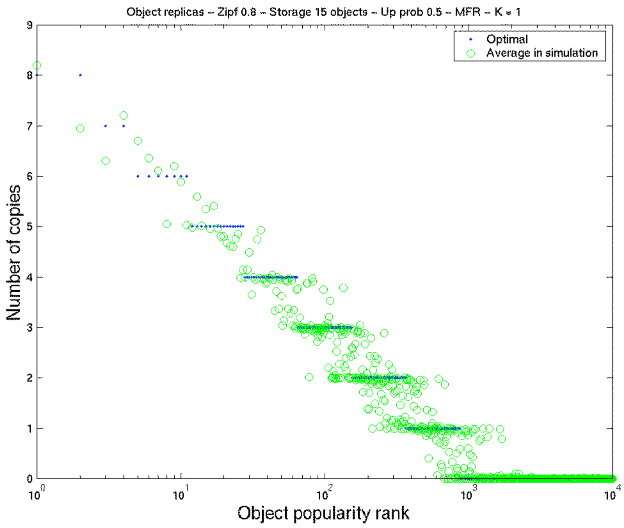
p = P(up) = .5

MFR: K=1

Zipf = .8

Ubiquitous Peer-to-Peer Infrastructures Group
Department of Computer Science

Replica Profile



p = P(up) = .5

15 objects per node

K = 1

Zipf = .8

Replica profile almost optimal

03.06.2007 25

Ubiquitous Peer-to-Peer Infrastructures Group
Department of Computer Science

Optimality of MFR

- Recall basic idea of MFR:
  - Each peer estimates local request rate for each object
- Analytical (offline) procedure for MFR Top-I: (all nodes)
  - Init: $\gamma_j = q_j / b_j$, $j = 1, \ldots, J$, and $T_i = S_i$, $i = 1, \ldots, I$
  1. Find file j with largest γ_j
  2. Sequentially examine winners for j until $T_i \geq b_j$ and $x_{ij} = 0$
     - Set $x_{ij} = 1$
     - Set $\gamma_j = \gamma_j(1-p_i)$
     - Set $T_i = T_i - b_j$
     - If no such node, remove file j from consideration
  3. If still files to be considered go to step 1, otherwise stop.
- Above procedure near-optimal
  - Difference at most 1 or 2 copies, usually no difference

03.06.2007 26

Summary: MFR Top-K Algorithm

Implementation

- Layers on top of DHT substrate
- Decentralized
- Simple: each peer keeps track of a local MFR table

Performance

- Provides near-optimal replica profile

Load Balancing

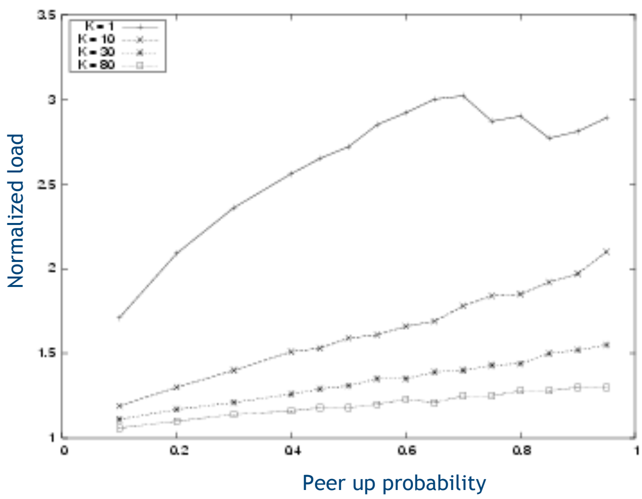
- What if the first place winner for a popular object is (almost) always up?
- Problem: How to balance the load between the peers in the community?

- Two approaches:
  - Fragmentation
  - Overflow

Load Balancing: Solutions

- Fragmentation
  - Idea: Divide each object into chunks, store chunks individually
  - One chunk is much smaller than a file, hence load is balanced better, since chunks are stored on different peers
  - Achieves overall load balancing
- Overflow
  - Idea: Allow peers to refuse requests
  - Request passed on to the next winner (eventually to outside)
    - Load on others will increase and hit-rate may decrease!
  - Allows a peer to decide how much traffic to handle
  - Achieves individual load balancing
- Fragmentation + Overflow
  - Use both approaches

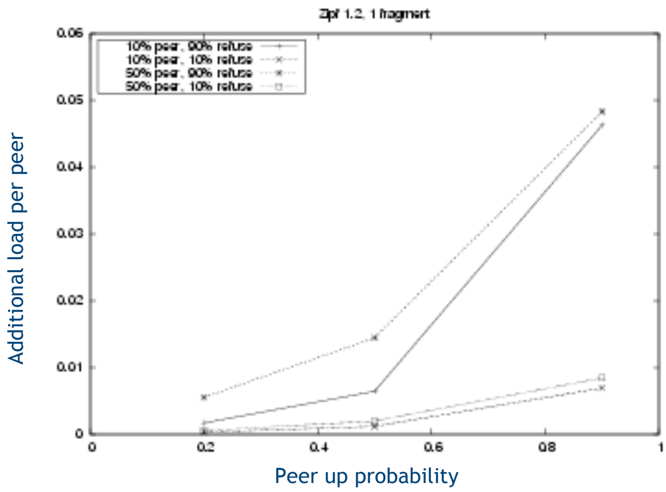
Load Balancing: Fragmentation



- 90-percentile load for Zipf parameter 1.2
- K = number of chunks
- Load normalized to "fair share"
- Works well for large number of chunks

Ubiquitous Peer-to-Peer Infrastructures Group
Department of Computer Science

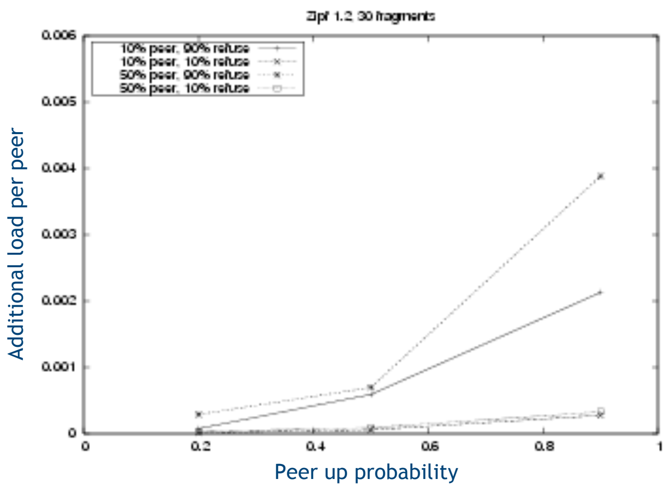
Load Balancing: Overflow



- Overflow with 1 chunk
- Different amounts of refused traffic
- Calculate new load on other peers
- Worst case: 5% additional load for each peer

Ubiquitous Peer-to-Peer Infrastructures Group
Department of Computer Science

Fragmentation + Overflow



- Same as above, but with 30 chunks per file
- Additional load less than 0.5% in all cases

Overflow: Refused Traffic

- When large number of traffic is refused, it goes to the outside, thus reducing hit-rate
- How much is hit-rate affected?
- Rough rule of thumb: Proportion of reduced traffic reduces overall storage capacity by the same proportion
- Example: If 50% of peers are refusing 50% of the traffic, then overall storage capacity is reduced by 25%

Ubiquitous Peer-to-Peer Infrastructures Group
Department of Computer Science

Load Balancing: Summary

- Without any load balancing mechanism, load is severely unbalanced
- Fragmentation approach works well for achieving a uniform load on all peers
- Pure overflow approach allows individual peers to reduce their load at a cost of increased load to others
- Overflow with fragmentation works best
- Refused traffic ends up effectively reducing the overall amount of storage offered by the community

Summary

1. Main contribution:
   - Set of adaptive algorithms for dynamically replicating and replacing files in a P2P community
   - No assumptions about nodes or node behavior, or file request probabilities
   - Algorithms are simple, adaptive, and fully distributed
   - Top-K MFR algorithm can be shown to be near-optimal
2. Second contribution:
   - Investigation of load balancing techniques for P2P communities
   - Without any load balancing, load concentrates on a few nodes
   - Fragmentation approach achieves a general load balance
   - Overflow approach allows for individual variation
   - Both shown to be very effective

Thank You!