Making Gnutella-like P2P Systems Scalable

Presented by: Karthik Lakshminarayanan

Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker

Central philosophy of the work

- File-sharing is a dominant P2P application
- DHTs might not be suitable for file-sharing
- Gnutella's design
  - Simplicity
  - Unscalable (number of queries and system size)
- Improve Gnutella
  - Adapt overlay topology and search algorithms to accommodate heterogeneity

Why not DHTs?

- P2P clients are extremely transient
  - Can DHTs handle churn as well as unstructured?
  - How would Bamboo compare with Gnutella?
- Keyword searches are more prevalent
  - Inverted indices might not scale
  - No unambiguous naming convention
- Most queries are for hay
  - Well-replicated content is queried for most

Gnutella's scaling problems

- Gnutella performs flooding-based search
  - Find files if they are replicated at small number of nodes
  - Obvious scaling issues
- Random walks
  - Forwarding is oblivious to node contents
  - Forwarding is oblivious to node load
- Bias towards high degree
  - Node capacity still not taken into account

GIA design

1. Dynamic topology adaptation
   - Nodes are close to high-capacity nodes
2. Active flow control scheme
   - Avoid overloaded hot-spots
   - Explicitly handles heterogeneity
3. One-hop replication of pointers to content
   - Allows high-capacity nodes to answer more queries
4. Search protocol
   - Based on random walks towards high-capacity nodes

Exploit heterogeneity

1. Topology adaptation

- Goal: Make high-capacity nodes have high degree (i.e., more neighbors)
- Each node has a level of satisfaction, S
  - S = 0 if no neighbors (dissatisfied)
  - S = 1 if enough good neighbors (fully satisfied)
  - S is a function of capacity, degree, age of neighbors and capacity of node
  - Improve the neighbor set as long as S < 1

1. Topology adaptation

- Improving neighbor set
  - Pick a new neighbor
  - Decide whether to preempt an existing neighbor
    - Depends on degree, capacity of neighbors
    - Asymmetric links?
- Issues
  - Avoid oscillations – use hysteresis
  - Converge to a stable state

2. Proactive flow control

- Allocate tokens to neighbors based on processing capability
  - Cannot perform arbitrary dropping due to random walk mechanism of GIA
- Allocation is proportional to neighbors' capacities
  - Incentive to announce true capacities
- Uses token assignment based on SFQ

3. One-hop replication

- Each GIA node maintains index of contents of all neighbors
- Exchanged during neighbor setup
- Periodically incrementally updated
- Flushed on node failures

4. Search protocol

- Biased random walk
  - Pick highest capacity node to which it has tokens
  - If no tokens, queues till tokens arrive
- TTLs to bound duration of random walks
- Book-keeping
  - Maintain list of neighbors to which a query (unique GUID) has been forwarded

Simulation results

- Compare four systems
  - FLOOD: TTL-scoped, random topologies
  - RWRT: Random walks, random topologies
  - SUPER: Supernode-based search
  - GIA: search using GIA protocol suite
- Metric:
  - Success-rate, Delay, Hop-count
    - Knee/collapse point at a particular query rate
  - *Collapse point*:
    - Per-node query rate at the knee
    - Aggregate throughput that the system can sustain

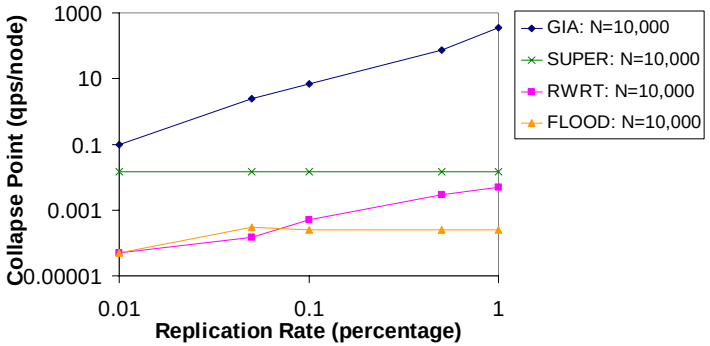
System model

- Capacities of nodes based on UW study
  - Separated by 4 orders of magnitude
- Query generation rate for each node
  - Limited by node capacity
- Keyword queries are performed
  - Files are randomly replicated
- Control traffic consumes resources
- Use uniformly random graphs
  - Prevent bias against FLOOD and RWRT

Questions addressed by simulations

- What is the relative performance of the four algorithms?
- Which of the GIA components matters the most?
- What is the impact of heterogeneity?
- How does the system behave in the face of transient nodes?

Single search response



- GIA outperforms SUPER, RWRT & FLOOD by many orders of magnitude in terms of aggregate query load
- Also scales to very large size network as replication factor determines scalability

Factor Analysis

Algorithm	Collapse point
RWRT	0.0005
RWRT+OHR	0.005
RWRT+BIAS	0.0015
RWRT+TADAPT	0.001
RWRT+FLWCTL	0.0006

Algorithm	Collapse point
GIA	7
GIA – OHR	0.004
GIA – BIAS	6
GIA – TADAPT	0.2
GIA – FLWCTL	2

10000 nodes, 0.1% replication

- No single component is useful by itself; the *combination* of all of them is what makes GIA scalable

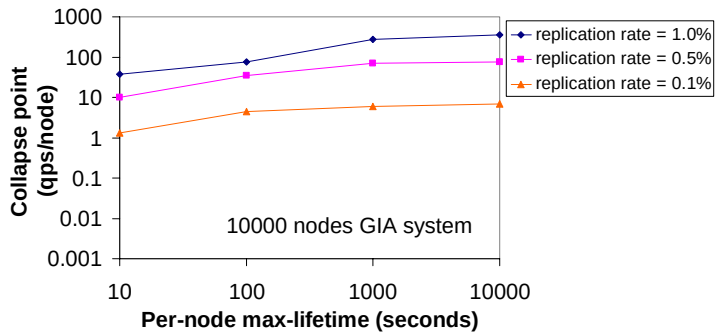
Impact of Heterogeneity

Algorithm	Collapse Point	Hop-count
GIA w/ Gnutella capacity distribution	7	15.0
GIA w/ uniform capacity distribution	2	46.0
RWRT w/ Gnutella capacity distribution	0.0005	978
RWRT w/ uniform capacity distribution	0.0525	987

10000 nodes, 0.1% replication

- GIA improves under heterogeneity
- Large CP-HC for GIA under uniform capacities as queries are directed towards high capacity nodes

Node failures

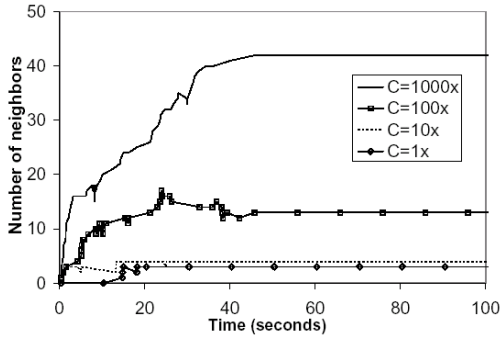


- Even under heavy churn GIA outperforms the other algorithms (under no churn) by many orders of magnitude

Implementation

- Capacity settings
  - Bandwidth, CPU, disk access
  - Configured by user
- Satisfaction level
  - Based on capacity, degree, age of neighbors and capacity of node
  - Adaptation interval $I = T.\ K^{-(1-s)}$, K = degree of aggressiveness
- Query resilience
  - Keep-alive message periodically sent
  - Optimizations on adaptation to avoid query dropping

Deployment



- Ran GIA on 83 nodes of PlanetLab for 15 min
- Artificially imposed capacities on nodes
- Progress of topology adaptation shown

Conclusions

- GIA: scalable Gnutella
  - 3–5 orders of magnitude improvement in system capacity
- Unstructured approach is good enough!
  - DHTs may be overkill
  - Incremental changes to deployed systems
- Can DHTs be used for file-sharing at all?