GIA: Making Gnutella-like P2P Systems Scalable

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

SIGCOMM 2003

1

Acknowledgements

Most of the followings slides are borrowed from the talk by Yatin Chawathe (Intel)

2

The Peer-to-peer Phenomenon

- Internet-scale distributed system
  - Distributed file-sharing applications
  - E.g., Napster, Gnutella, KaZaA
- File sharing is the dominant P2P app
- *Mass-market*
  - Mostly music, some video, software

3

The Problem

- Potentially millions of users
  - Wide range of heterogeneity
  - Large transient user population
- Existing search solutions cannot scale
  - Flooding-based solutions limit capacity
  - Distributed Hash Tables (DHTs) not necessarily appropriate

4

Why Not DHTs

- Structured solution
  - Given a filename, find its location
- Can DHTs do file sharing?
  - Probably, but with lots of extra work: Caching, keyword searching
- Do we need DHTs?
  - Not necessarily: Great at finding rare files, but most queries are for popular files

Note: Not questioning the utility of DHTs in general, merely for mass-market file sharing

Why Not DHTs

- Structured solution
  - Given a filename, find its location
  - Tightly controlled topology & file placement
- Unsuitable for file-sharing
  - Transient clients cause overhead
  - Poorly suited for keyword searches
  - Can find rare files, but that may not matter

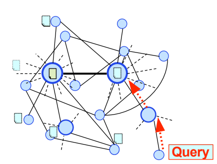
Note: Not questioning the utility of DHTs in general, merely for mass-market file sharing

Proposed Solution: GIA

- Unstructured, but take node capacity into account
  - High-capacity nodes have room for more queries: so, send most queries to them
- Will work only if high-capacity nodes:
  - Have correspondingly more answers, and
  - Are easily reachable from other nodes

GIA Design

- Make high-capacity nodes easily reachable
  - Dynamic topology adaptation
- Make high-capacity nodes have more answers
  - One-hop replication
- Search efficiently
  - Biased random walks
- Prevent overloaded nodes
  - Active flow control



Dynamic Topology Adaptation

- Make high-capacity nodes have high degree (i.e., more neighbors)
- Per-node *level of satisfaction*, S:
  - 0 ⇒ no neighbors, 1 ⇒ enough neighbors
  - Function of:
    - Node's capacity, Neighbors' capacities, Neighbors' degrees
    - Sum of neighbors capacities (normalized by their degrees) divided by the node's own capacity
    - Intuition: a node with capacity *C* will forward *C* queries per unit time at full load and needs enough capacity from all its neighbors to be able to handle that load
  - When S << 1, look for neighbors aggressively

9

Dynamic Topology Adaptation (cont'd)

- Each node keeps a host cache populated with nodes it knows about or discovers
- If S < 1, then it tries to add nodes from its host cache to its neighbor list
  - If number of neighbors reaches a maximum level, then some current neighbor has to be dropped to make room for the new neighbor
  - If the new neighbor has higher capacity than an existing neighbor then it is added
  - O/w, the new node is added if it has a lower degree than the current neighbor with the highest degree
    - Neighbor with highest degree has least to lose if it is dropped

10

Flow Control

- Active flow control
  - Senders are allowed to direct queries to a neighbor only if that neighbor has notified the sender that it is willing to accept queries from the sender
  - Each GIA client periodically assigns flow-control tokens to its neighbors
    - Each token represents a single query
    - Tokens assigned using Start-time Fair Queuing (a proportional-share scheduling algorithm)
    - Neighbors assigned tokens in proportion to their advertised capacity

11

Other Design Features

- One-hop replication
  - Each node actively maintains an index of the content of all its neighbors
- Search algorithm
  - Biased random walk
  - A node forwards a query to the highest capacity neighbor for which it has flow control tokens
    - If no tokens, query is queued until tokens arrive
  - TTLs used to bound the duration of the random walk and book-keeping techniques to avoid redundant paths (unique GUID per query + query history)
  - Query duration also bounded by MAX_RESPONSES parameter

12

Other Design Features (cont'd)

- Query resilience
  - Drawbacks of random walk: if a node dies before it has forwarded a query, the query will be lost
  - GIA relies on query keep-alive messages to address this issue
  - Query responses serve as implicit keep-alive messages
  - If a query is forwarded several times without any responses, an explicit keep-alive message is sent to the originator, who can reissue the query

13

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:
  - *Collapse point*: aggregate throughput that the system can sustain

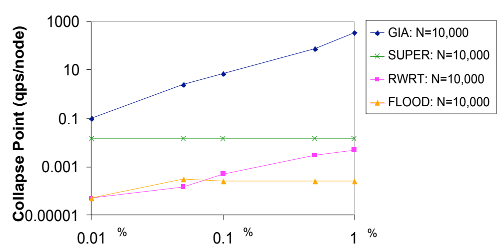
14

Questions

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

15

System Performance



GIA outperforms SUPER, RWRT & FLOOD by many orders of magnitude in terms of aggregate query load

16

Factor Analysis

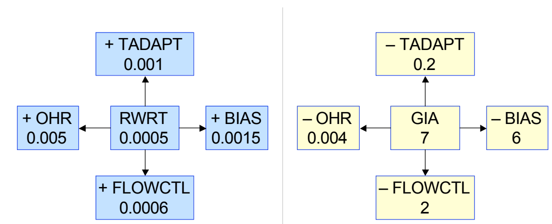
Algorithm	Collapse point
RWRT	0.0005
RWRT+OHR	0.005
RWRT+BIAS	0.0015
RWRT+TADAPT	0.001
RWRT+FLWCTL	0.0006

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

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

17

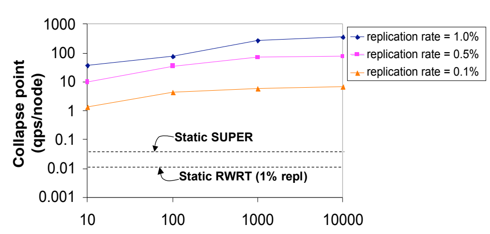
Factor Analysis



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

18

Transient Behavior



Even under heavy churn GIA outperforms the other algorithms by many orders of magnitude

19

Summary

- 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
- Status: Prototype implementation deployed on PlanetLab

20