GIA: Making Gnutella-like P2P Systems Scalable

Yatin Chawathe
Sylvia Ratnasamy, Scott Shenker, Nick Lanham,
Lee Breslau
2003

(Several slides have been taken from the authors' original presentation)

The Problem

- Large scale P2P system: millions of users
  - Wide range of heterogeneity
  - Large transient user population (in a system with 100,000 nodes, 1600 nodes join and leave per minute)
- Existing search solutions cannot scale
  - Flooding-based solutions limit capacity
  - Distributed Hash Tables (DHTs) not necessarily appropriate (for keyword-based searches)

A Solution: GIA

- Scalable Gnutella-like P2P system
- Design principles:
  - Explicitly account for node heterogeneity
  - Query load proportional to node capacity
- Results:
  - GIA outperforms Gnutella by 3–5 orders of magnitude

Outline

- Existing approaches
- GIA: Scalable Gnutella
- Results: Simulations & Experiments
- Conclusion

Distributed Hash Tables (DHTs)

- Structured solution
  - Given the exact filename, find its location
- Can DHTs do file sharing?
  - Yes, but with lots of extra work needed for keyword searching
- Do we need DHTs?
  - Not necessarily: Great at finding rare files, but most queries are for popular files
- Poor handling of churn – why?

Other Solutions

- Supernodes [KaZaA]
  - Classify nodes as low- or high-capacity
  - Only pushes the problem to a bigger scale
- Random Walks [Lv *et al*]
  - Forwarding is blind
  - Queries can get stuck in overloaded nodes
- Biased Random Walks [Adamic *et al*]
  - Right idea, but exacerbates overloaded-node problem

Outline

- Existing approaches
- GIA: Scalable Gnutella
- Results: Simulations & Experiments
- Conclusion

GIA: High-level view

- 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 converts them into high-degree nodes
- 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), and keep low capacity nodes within short reach from them.
- Per-node *level of satisfaction*, S:
  - 0 = no neighbors, 1 = enough neighbors

  Satisfaction S is a function of:

  - Node's capacity
  - Neighbors' capacities
  - Neighbors' degrees

  When S << 1, look for neighbors aggressively

Dynamic Topology Adaptation

Each GIA node maintains a host cache containing a list of other GIA nodes. The host cache is populated using a variety of methods (like contacting well-known web-based hosts, and exchanging host information using PING-PONG messages.

A node X with $S < 1$ randomly picks a node Y from its host cache, and examines if it can be added as a neighbor.

Topology adaptation steps

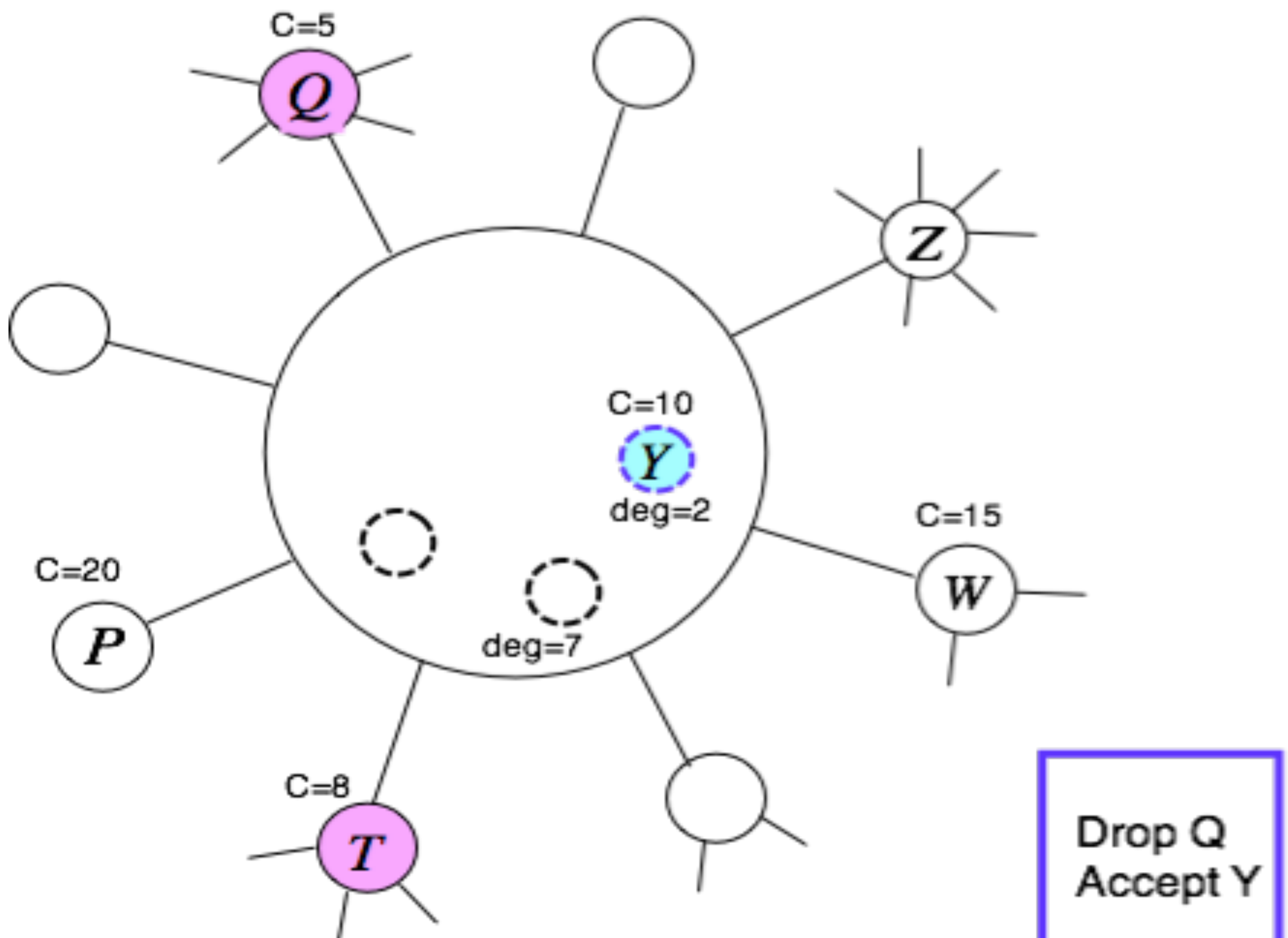
Life of Node X : it picks node Y from its host cache

Case 1 {Y can be added as a new neighbor}
(Let C_i represent capacity of node i)
if num nbrsX + 1 < max nbrs that it can handle **then** there is room
ACCEPT Y ; return

Case 2 {Node X explores if to replace an existing neighbor in favor of Y}
subset := every neighbor i from nbrsX such that $C_i \leq C_Y$
if subset is empty, i.e. no such neighbors exist **then** REJECT Y ; return
else candidate Z := highest-degree neighbor from subset

If Y has higher capacity than Z
or (num nbrsZ > num nbrsY + H) {Y has fewer nbrs}
then DROP Z; ACCEPT Y
else REJECT Y
{Do not drop poorly connected nodes in favor of well-connected ones}

Topology adaptation steps



Active Flow Control

- Accept queries based on capacity
  - Actively allocate “tokens” to neighbors
  - Send query to neighbor only if we have received token from it
- Incentives for advertising true capacity
  - High capacity neighbors get more tokens to send outgoing queries
  - Nodes not using their tokens are marked inactive and this capacity is redistributed among its neighbors.

Outline

- Existing approaches
- GIA: Scalable Gnutella
- Results: Simulations & Experiments
- Conclusion

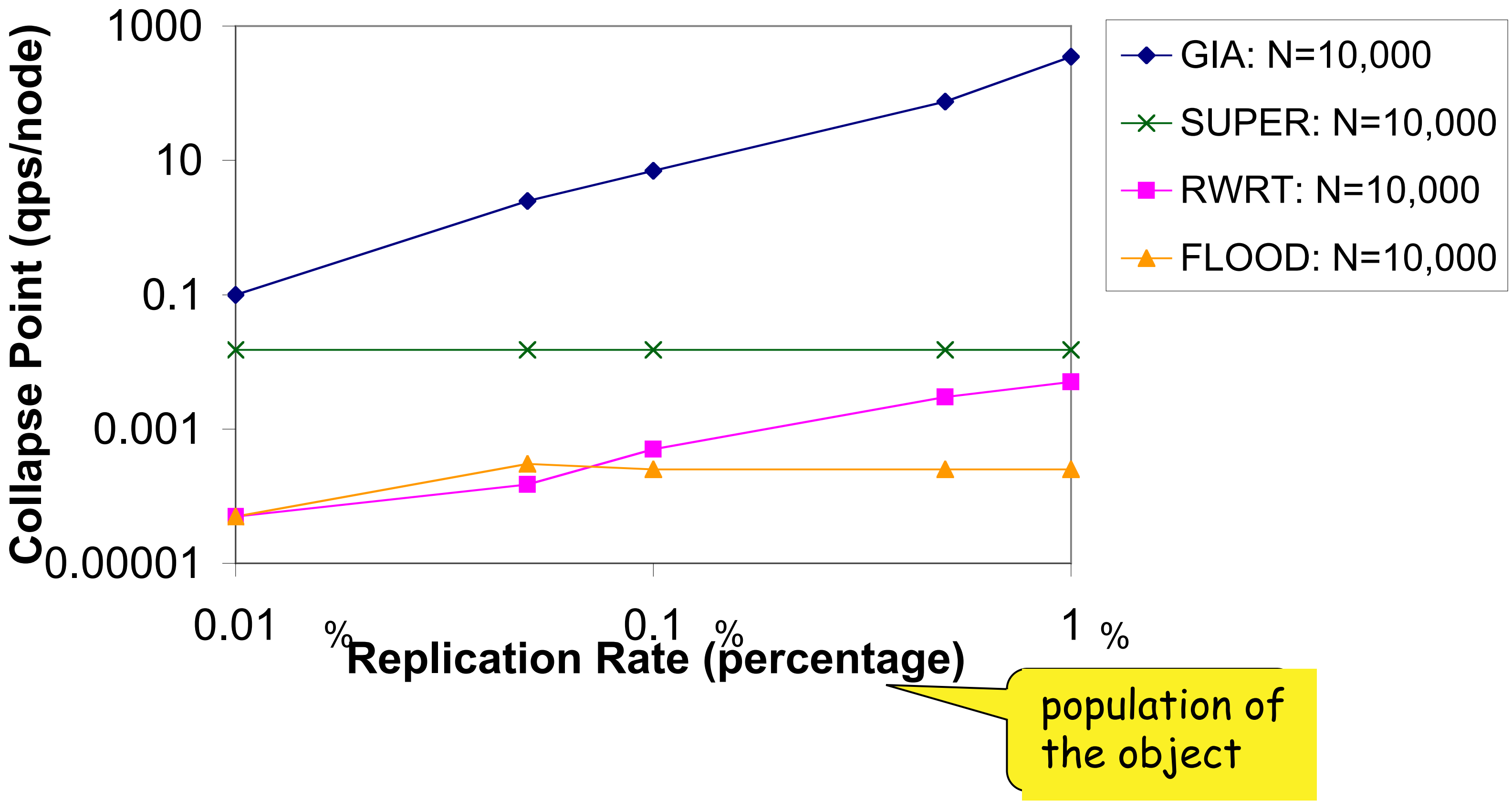
Simulation Results

- 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 (per node query rate beyond which the success rate drops below 90%)

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?

System Performance



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

Factor Analysis

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

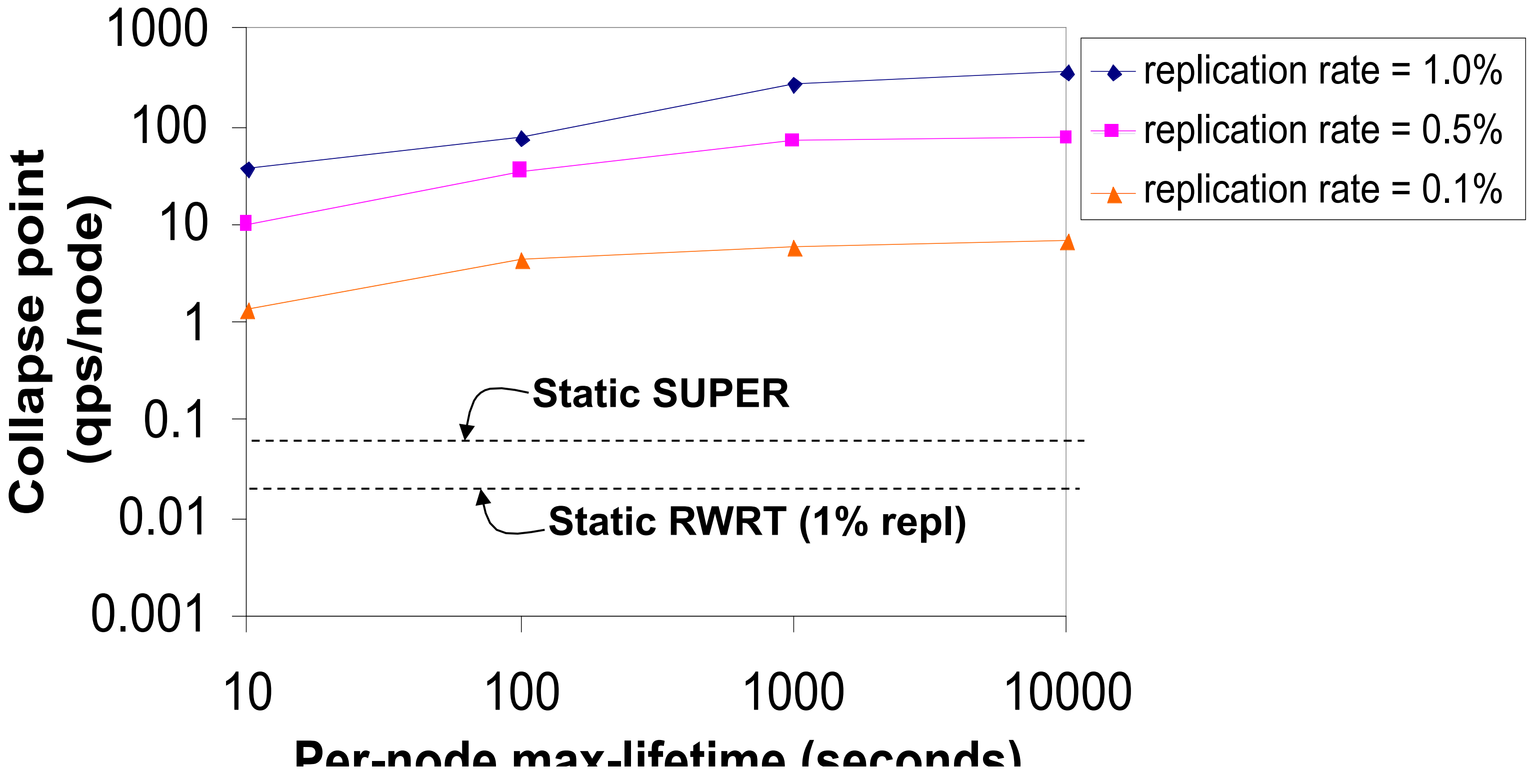
Algorithm	Collapse point
GIA	7
GIA – OHR	0.004
GIA – BIAS	6
GIA – TADAPT	0.2
GIA – FLWCTL	2

Topology adaptation

Flow control

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

Transient Behavior

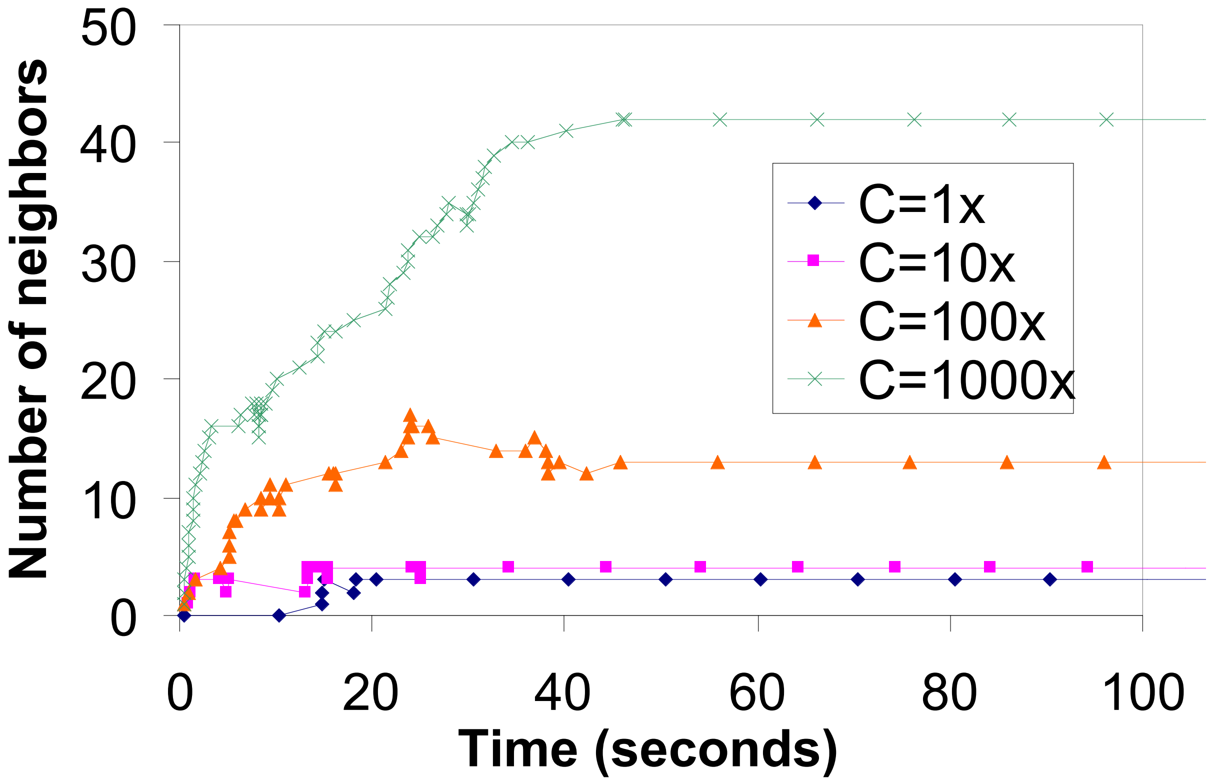


Even under heavy churn, GIA outperforms the other algorithms by many orders of magnitude

Deployment

- Prototype client implementation using C++
- Deployed on PlanetLab:
  - 100 machines spread across 4 continents
- Measured the progress of topology adaptation…

Progress of Topology Adaptation



Nodes quickly discover each other and soon reach their target "satisfaction level"

Outline

- Existing approaches
- GIA: Scalable Gnutella
- Results: Simulations & Experiments
- Conclusion

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