Detecting heavy-hitters in a P2P network

Zbigniew Gołębiewski[1], Jarosław Kutyłowski[2], Mirosław Kutyłowski[1], Filip Zagórski[1]

Wrocław University of Technology[1], Paderborn University[2]

N2S 2009

FRONTS, 7th Framework Programme, contract 215270

P2P networks
ideas and advantages

P2P architecture

no central control: self-organization

- **dynamic:** a peer can join and leave the network, nevertheless the network works properly

distributed memory: information spread among the peers, allocation usually with distributed hash tables

P2P networks
ideas and advantages

P2P architecture

no central control: self-organization

- **dynamic:** a peer can join and leave the network, nevertheless the network works properly

distributed memory: information spread among the peers, allocation usually with distributed hash tables

Advantages

1. global scale network
2. small administration overhead, no manual work
3. efficient communication framework
4. cheap
5. resilient to faults

Heavy hitter

unfair use of P2P networks

Normal user

- a few querries, a few downloads
- contribution proportional to usage

Heavy hitter
unfair use of P2P networks

Normal user

- a few querries, a few downloads
- contribution proportional to usage

Heavy hitter

- many querries, many downloads,
  unfair use of databases
- crawlers
- parasite networks stealling data from P2P and offering them elsewhere

Heavy hitter
goal

Detection

- detect P2P nodes that are using the network unfairly
- detect the nodes that **contact a fraction of all nodes**

Heavy hitter
goal

Detection

- detect P2P nodes that are using the network unfairly
- detect the nodes that **contact a fraction of all nodes**

Limitations

- must be a fully distributed solution, no central control
- low communication overhead

Algorithm overview
idea

Math background idea

exponential growth within $\frac{1}{2}\log n$ steps:

1. from $\sqrt{n}$ to n
2. from 1 to only $\sqrt{n}$

Algorithmic idea

- give enough time ($\frac{1}{2}\log n$) for key information to disseminate to all nodes
- but not enough time to disseminate noise

Algorithm overview

Input

Each node A holds a list A_L of all nodes that have requested some service from A.

Algorithm overview

Input

Each node A holds a list A_L of all nodes that have requested some service from A.

Phase 1

Each node A fetches a small random sublist of B_L of a random B and computes its intersection with A_L (*short list*)

Algorithm overview

Gołębiewski, Kutyłowski[2], Zagórski

Input

Each node A holds a list A_L of all nodes that have requested some service from A.

Phase 1

Each node A fetches a small random sublist of B_L of a random B and computes its intersection with A_L (*short list*)

Phase 2

1. the short lists are disseminated.
2. a node merges its own short list and the lists received.

Algorithm overview

Input

Each node A holds a list A_L of all nodes that have requested some service from A.

Phase 1

Each node A fetches a small random sublist of B_L of a random B and computes its intersection with A_L (*short list*)

Phase 2

1. the short lists are disseminated.
2. a node merges its own short list and the lists received.

Phase 3

1. each node inspects some number of short lists
2. a node considered heavy hitter if on most of these list

Algorithm overview
idea

What happens with a heavy hitter than appears on a fraction α of all lists?

phase 1
: if the list have size m and sublists have size k, then it appears on a random sublist with pbb

$$\alpha^2 \cdot \frac{k}{m}$$

i.e. some fraction has the heavy hitter on the short lists

phase 2
: heavy hitter disseminated back to almost all lists

phase 3
: just checking a few lists to exclude noise entries (i.e. honest peers)

Algorithm overview
idea

What happens with an honest peer S that used K servers

phase 1 if the list have size m and sublists have size k, then it appears on a random sublist with pbb

$$\left(\frac{K}{m}\right)^2 \cdot \frac{k}{m}$$

i.e. only incidentally a short list may contain S

phase 2 the number of list containing S grows but still not to a constant fraction of all lists

phase 3 during checking very unlikely that majority of lists contain P

Hash intersection

idea

Problem

Given:

- peer A holds a big list A_L
- peer B holds a small list B_R

Find intersection of A_L and B_R,
but minimize communication.

Simple solution

- B sends B_R to A.
- A computes the intersection.

if the intersection is small, this might be a waste of communication

Hash intersection
mechanism

Round 1

1. each entry in B_R hashed (keyed hash)
2. each hash truncated to l_1 bits
3. the list of truncated hashes sent to A
4. A responds with a bitvector stating which elements from B_R are not in A for sure:
   i.e. which truncated entries correspond to no truncated hash computed for A_L

Hash intersection
mechanism

Gołębiewski, Kutyłowski[2], Zagórski

Round 2

repeat with the candiates left, with a new hash function and truncation to l_2 bits

Round 3,...

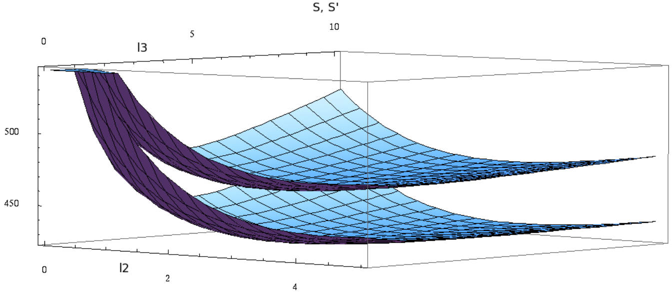
...

Hash intersection
parameter choice

Optimization

- find the optimal number of rounds, and the lengths $l_1, l_2, \ldots$
- formulas derived, numerical estimation of minima possible in practical situtations

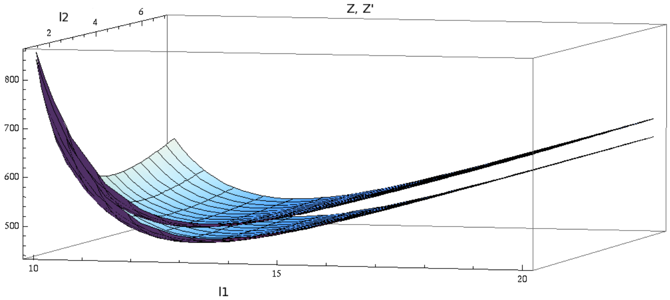
Hash intersection
choice of parameters



The expected communication complexity in case of 3 rounds algorithm and honest users (S - smaller values) and heavy hitter (S' - bigger values) for $k = 30$, $m = 1024$, $l_1 = 12$, adress space $N = 2^{30}$.

Hash intersection

choice of parameters



The expected communication complexity in case of 2 rounds algorithm and honest users (Z - smaller values) and heavy hitter (Z' - bigger values) for $k = 30, m = 1024, N = 2^{30}$.

Hash intersection
advantages

the optimal choice of parameters l_1, l_2, l_3, "c.c" denotes expected communication complexity (respectively, S, S', Z and Z'), "rel. c.c." denotes, respectively, $T3$, $T3'$, $T2$, $T2'$:

case	l_1	l_2	l_3	c.c.	rel. c.c
$r = 3$ no h.h.	12	2	4	425	0.472
$r = 3$ with h.h.	12	2	4	462	0.513
$r = 2$ no h.h.	12	4	-	441	0.490
$r = 2$ with h.h.	12	4	-	474	0.527

Phase 1

1. hash intersection used,
2. some numbers:
   - heavy hitter on 50% lists, i.e. $\alpha = 0.5$
   - $m = 1024$, i.e. each server holds 1024 names
   - $k = 32$, the size of random sublists

   then

   - a heavy hitter on $\approx$ 0.0078 intersection lists
   - an honest user on $\approx$ 0.00000003 approximation list

Phase 2

Epidemic process

PUSH, r_1 rounds: during a round each node that holds a non-empty intersection list chooses another node uniformly at random and sends there its intersection list.

PULL, r_2 rounds: during a round a node having an empty intersection list asks a node chosen uniformly at random for its intersection list. If the answer is a non-empty list, the asking node takes it.

r_1 and r_2 must be carefully chosen

Phase 3
voting

Gołębiewski, Kutyłowski[2], Zagórski

algorithm

1. each node asks c random nodes for their short lists
2. only a node on majority of these lists considered as heavy hitter

- trade-off between false (positives, negatives) and communication
- c must be carefully chosen

Evaluation

HH	r_1	r_2	il_1	il_2	il_3	CC	len_{il}
0	4	3	1.5%	84.8%	0.08%	$2.8 \cdot M$	1.05
0	5	2	1.5%	83.1%	0.12%	$2.3 \cdot M$	1.06
0	6	1	1.5%	80.0%	0.23%	$2.2 \cdot M$	1.07
1	4	3	2.3%	93.6%	37.6%	$2.7 \cdot M$	1
1	5	2	2.3%	92.0%	43.5%	$2.4 \cdot M$	1
1	6	1	2.3%	89.0%	53.2%	$2.7 \cdot M$	1
5	4	3	5.3%	99.7%	70.7%	$2.6 \cdot M$	1.4
5	5	2	5.3%	99.3%	84.7%	$3.3 \cdot M$	1.9
5	6	1	5.3%	98.0%	92.2%	$5.4 \cdot M$	2.8
100	4	3	52.1%	100%	97.7%	$24.9 \cdot M$	15
100	5	2	52.1%	100%	99.8%	$56.9 \cdot M$	40
100	6	1	52.1%	100%	99.9%	$114.1 \cdot M$	67

$M = 2^{19}$, $m = 512$, $k = 16$, $\alpha = 0.5$, $c = 5$, HH=the number of heavy hitters, r_1 and r_2 = numbers of rounds in Phase 2, il_j=fraction of servers with a nonempty intersection list after phase j, CC=communication complexity of 2nd phase and len_{il} =the average size of intersection lists

Attacks

Security

1. changing the lists on a limited number of peers does not change the result of the algorithm
2. no single point of failure

Final remark

Hash intersection

after some tuning it improves the algorithm presented at ACNS'2009 just two weeks before in Paris

Thank you for your attention!