Adaptive Gossip-based Broadcast

Antonio Bucchiarone

Fondazione Bruno Kessler, Trento – Italy
bucchiarone@fbk.eu

8 October 2018

Gossip in Distributed Systems

- Every node that *receives* a message, *buffers it*, and then *forwards it* a certain number of times, each time to a *randomly selected subset of processes*.

  - **Strong Assumption**: enough buffering resources exist on all nodes.

  - **Real Context**: the nodes must be equipped with *enough resources* to ensure that messages are gossiped a sufficient number of times.

    - If *a node does* not have enough resources, it *may drop a large number of messages* thar are being forwarded.

    - If *several nodes* do not have enough resources, *reliability might end up being drastically impacted*.

Problem Definition - I

- A **large scale publish-subscribe application**
  - *Publishers* which brodcast information
  - *Subscribers which register interest in receiving certain types of information.*
- Gossip-based broadcast is typically used to *disseminate the information* from the *publishers* to the *interested subscribers*.
- Since different nodes are interested in different type of information we can have *different broadcasts groups*.
- *Any node may belong to more than one broadcast group*, and this number varies as nodes dynamically subscribe to new types or cancel previous subscriptions.
- The resources at each node *are limited*, every node has to *dynamically divide the available resources among the groups it belongs to*.

Problem Definition - II

- We need a dynamic feedback mechanism where each node has *different and varying amount of resources*.

- Classical gossip-based algorithms *discard messages in overload conditions*, without providing to the source (the sender) any feedback regarding the reliability of the operation.

- The *rate* of new messages in the system is *unpredictable* and depends on the *sum of the individual emission rates*.

- To *estimate the global congestion* and control the message emission at each sender.

The Idea

To disseminate and gather information about the *resources available in a broadcast group* such that every sender can *adjust its emission accordingly*.

To ensure that senders are able to *perceive the quality of the algorithm operation*, in terms of reliability with the current system configuration, *without interacting with other nodes*.

Algorithm Mechanisms

FONDAZIONE BRUNO KESSLER

1. *Distributed mechanism to determine resource availability* (i.e., the size of buffers), which changes only upon reconfiguration of nodes.

2. *A local mechanism to determine resource usage* (i.e., buffer occupancy), whose variation is far more being frequent and unpredictable as it is affected by the timing of senders and network delays.

3. The resulting informations from the combined mechanisms can be then used *to adjust the rate at which each node is allowed to send messages*.

Distributed Discovery of Resource Availability

Initially:

$s = \Delta$ *Interval for each estimate – period s*

$\text{minbuff}_p^r = |\text{events}|_m$, for all $1 \leq r \leq \Delta$ *Size of the smallest buffer in the group.*

every T ms:

...

{Add information to gossip message}

gossip.s $\leftarrow$ s

gossip.minBuff $\leftarrow \text{minBuff}_p^s$

...

In every gossip round, values *s* an *minBuff* are included in the message header.

upon RECEIVE(gossip):

...

{*Compute new known minimum*}

if gossip.s=s $\wedge$ gossip.minBuff $< \text{minBuff}_p^s$ **then**

$\text{minBuff}_p^s \leftarrow$ gossip.minBuff

Every time a node q receives a message from another process p, it updates its own estimate of minBuff for period s

(min between p and q in s).

every S ms:

emph{Enter new period}

$s \leftarrow s + 1$

$\text{minBuff}_p^s \leftarrow |\text{events}|_m$

$\text{minBuff} = min(\text{minBuff}_p^s, \ldots, \text{minBuff}_p^{s-\Delta+1})$

Estimating Buffer Congestion

Initially:
  $\text{avgAge} = (H + L)/2$
  $\text{lost} = \emptyset$
upon RECEIVE(gossip):
  ...
  {*Update congestion estimate*}
  while $|\text{events} \setminus \text{lost}| > \text{minBuff}$ **do**
    select oldest element e *from* events \ lost
    $\text{avgAge} = \alpha\ \text{avgAge} + (1 - \alpha)\ \text{e.age}$
    $\text{lost} \leftarrow \text{lost} \cup \{\text{e}\}$
{*Garbage collect events*}
  while $|\text{events}| > |\text{events}|_m$ **do**
    remove oldest element e *from* events

Upon *receiving each gossip message*, after *storing events* and *updating their ages*, *minBuff* is used as a threshold *to select which events would have to be discarded*.

Rate Adaptation

FONDAZIONE BRUNO KESSLER

To avoid that each sender changes its rate with every minor oscillation of *avgAge,* causing *a continuous oscillation of the system.*

every T ms:

...

{*Throttle sender*}

if avgAge $> H \wedge$ avgTokens$<$max$/2 \wedge$

$\wedge$ rand $> W$ **then**

rate $\leftarrow$ rate$\times(1 + r_H)$

if avgAge $< L \vee$ avgTokens$>$max$/2$ **then**

rate $\leftarrow$ rate$\times(1 - r_L)$

Two Threshold L and H.

L -> low-age mark

H -> high-age mark

- When the *system is congested* (i.e., the average age is below L), the sender reduces its rate by some amount denoted *rL.*
- When *new resources are released in the system*, and the rate can be increased (i.e,. The average age is is above H), the.
- *sender increases by an amount denoted rH*

System Configuration

- $S = 2 \times T$
- $\Delta = 2$
- $\alpha = 0.8$
- $H = 7$, $L = 5$
- $rH = 5\%$, $rL = 5\%$
- $W = 0.5$