How to keep track of the latest gossip

Vijay D’silva

Chair of Software Engineering,
ETH Zürich
Papers


Problem overview

- Distributed system with \( n \) processes.
- Communication via message passing on point-to-point channels.
- Unbounded delays and message re-ordering.
- *Gossip* is information about other processes.
- \( p \) tells \( q \) about \( r \). \( q \) knows something about \( r \). Can \( q \) decide whose gossip is hot?
- Can the overhead required to do this be bounded?
Purported applications

- Obtaining *distributed snapshots* of the system’s global state.
- Ordering messages based on causality.
- Sensor networks.
- Network-on-Chip architectures.
- Other upcoming applications of gossip-based protocols.
A message passing system

- $e$ is an event. Two possibilities.
- $send(p, q, m)$ - process $p$ sends message $m$ (gossip) to $q$
- $recv(q, p, m)$ - process $q$ receives gossip $m$ from $p$
- Latest gossip? Temporal ordering required.
- Local events - $e < e'$ if $e$ occurs before $e'$ in a process.
- External events - $send(p, q, m) < recv(q, p, m)$
- Say everything you know when you gossip.
- Use this information to compute ordering of remaining events.
Simple solution

- Each process has a local counter.
- *Events* are time-stamped.
- Compare time-stamps to obtain the latest gossip.
- Caveat: counters are unbounded.
- Problem: Message overhead is unbounded
Desired solution

We want a bounded message overhead. This entails:

- Reusing time-stamps/labels.
- Bounding the number of unacknowledged messages sent.
- However, acknowledgements are not sufficient.

**Intuition:** A label can be reused only if no other process is using them.

**Issues to be addressed:**

- A way to use labels to compute temporal ordering of events.
- Identify when labels can be reused.
What information is available?

- **latest**: latest events from the current snapshot.
- **unack**: local *unacknowledged* messages.
- **ack\textsubscript{pending}**: external *unacknowledged* messages.

**Question**: When is it enough?

- Discard *stale* information.
Primary graph

- \(\text{primary}\_g(C_p)\): process \(p\)'s view of the system.
- Used to communicate information.
- Edge between every two temporally ordered events.
- Vertices: \((e, l)\), \(e\) - event, \(l \in \{\text{latest}, \text{unack}\}\)
- Primary graph and \(\text{ack}_{\text{pending}}\) sent with every message.
Observations

- $\text{recv}(q, p, m)$ does not have *hot* gossip if $\text{send}(p, q, m) \in C_q$.

- $\text{primary}(C_p)$ and $\text{primary}(C_q)$ sufficient to know what is new.

- Check $\text{latest}(C_p) \cap \text{unack}(C_q)$ and vice versa.

- $\text{ack}_{\text{pending}}$ can be inherited from new events.

- Use $\text{ack}_{\text{pending}}$ to update $\text{unack}$.
Observations stated are sufficient to compute latest gossip.

Remains to bound timestamps.

Only interested in current events.

At most $b \times n^2 + b$ distinct events in $\text{primary}_g$

At most $b \times n$ events in $\text{ack}_{\text{pending}}$

Bound on events in system: $n \times ((2 \times b + 1) \times n + b \times n^2)$
Protocol

- Communicate $primaryg$ and $ack_{pending}$ information.
- Maintain a list of current events in different $primaryg$
- Update on each communication.
Criticism and appreciation

- Overhead is quite large. Protocol might not be practical for domain of applicability.
- Paper was notationally heavy. Quite difficult to read especially with time constraints.
- The result applies to a very general setting which may explain the complexity.
- Labels need not be timestamps. Causal ordering is computed independent of the kind of labels used. Quite remarkable.