Bimodal Multicast

Harald Gjermundrød
CptS/EE 562
February 10, 2004

Bimodal Attack

- General commands soldiers
- If *almost all* attack victory is certain
- If *very few* attack, they will perish but Empire survives
- If *many* attack, Empire is lost

Gossip Architecture

- “Bad news travel fast”
  - Question: How?
  - Answer: By ???
- Can this be used to communicate fast between computers
- The idea:
  - You exchange news with your neighbors about the latest gossip you have heard
  - If the neighbors have not heard it, they will ask you to tell them
  - You send the gossip to the neighbors
  - The neighbors will then again gossip it to its neighbors
- Since everybody is interested in hearing all the latest gossip the messages will be delivered to all the participants in a group

Reliable Multicast

- Family of protocols for 1-n communication
- Increasingly important in modern networks
- “Reliability” tends to be at one of two extremes on the spectrum:
  - Very strong guarantees, handles worst case scenarios at all time
    - Does not scale well, due to large and growing overheads
  - Best effort guarantees, for applications like Internet conferencing or radio
    - Scales better, but large uncertainty of delivery
Current Models

- The two “schools” of reliable multicast
  - Virtual synchrony model (Isis, Horus, Ensemble)
    - All or nothing message delivery (atomicity) with ordering
    - Membership managed on behalf of group
    - State transfer to joining member
  - Scalable Reliable Multicast (MUSE, SRM, RMTP)
    - Local repair of problems but no end-to-end guarantees

- Bimodal places itself between these two classes
  - Predictable performance
  - Easy to tune the different tradeoffs
  - Scales well

Bimodal multicast a new design

- Bimodal multicast is reliable in a sense that can be formalized, at least for some networks
  - Generalization for larger class of networks should be possible but maybe not easy
- Protocol is also very stable under steady load even if 25% of processes are perturbed (slow)
- Scalable in much the same way as SRM

- Two Implementations:
  - Using Ensemble: stack microprotocols on top of each other. Like the Horus project
  - Spinglass: Free-standing solution
    (Differences: Spinglass uses gossip to track membership, Ensemble version uses a microprotocol for membership)

Properties Of The Protocol

- Atomicity: Almost all or almost none (see fig first page)
- Throughput Stability: Want to minimize the variation of throughput
- Ordering: Messages are delivered in FIFO order. (Total Ordering possible by inserting an extra layer above)
- Multicast Stability: delivers messages stable to the application layer and garbage collect
- Detection of lost messages: Application layer informed by an upcall
- Scalability: Overheads are constant as a function of group size, throughput variance grows slowly (log of group size)

Limitations/Assumption

- Assume that most links have known throughput and loss properties
- Assume that most processes are responsive to messages in bounded time, (synchronies)
- But can tolerate some flaky links and some crashed or slow processes
- Does not consider Byzantine failures
How the protocol works

Start by using unreliable multicast to rapidly distribute the message. But some messages may not get through, and some processes may be faulty. So initial state involves partial distribution of multicast(s).

How the protocol works cont.

Periodically (e.g. every 100ms) each process sends a digest describing its state to some randomly selected group member. The digest identifies messages. It doesn’t include them.

How the protocol works cont.

Recipient checks the gossip digest against its own history and solicits a copy of any missing message from the process that sent the gossip.

How the protocol works cont.

Processes respond to solicitations received during a round of gossip by retransmitting the requested message. The round lasts much longer than a typical RPC time.
**Delivery? Garbage Collection?**

- Deliver a message when it is in FIFO order
- Garbage collect a message when you believe that no “healthy” process could still need a copy
- Match parameters to intended environment

**Worries:**
- Someone could fall behind and never catch up, endlessly loading everyone else
- What if some process has lots of stuff others want and they bombard him with requests?
- What about scalability in buffering and in list of members of the system, or costs of updating that list?

**Optimizations**

- Request retransmissions most recent multicast first
  - Idea is to “catch up quickly” leaving at most one gap in the retrieved sequence

- Participants bound the amount of data they will retransmit during any given round of gossip. If too much is solicited they ignore the excess requests

- Label each gossip message with senders gossip round number
  - Ignore solicitations that have expired round number, reasoning that they arrived very late hence are probably no longer correct

**Optimizations cont.**

- Don’t retransmit same message twice in a row to any given destination (the copy may still be in transit hence request may be redundant)

- Use IP multicast when retransmitting a message if several processes lack a copy
  - For example, if solicited twice
  - Also, if a retransmission is received from “far away”
  - Tradeoff: excess messages versus low latency

- Use regional TTL to restrict multicast scope

**How fast does the gossip travel**

- Number of rounds with gossip before all the processes have received the message
  - Both have 1000 process, notice scale on the right one
  - Left: initial multicast fails
  - Right: initial multicast reaches 90% of the processes
**How well does it scale**

- How does this round number increase with increasing number of processes
  - Y axes: probability of a correct process to receive a message
  - X axes: the number of rounds
  - Conclusion: Number of rounds increases log (#processes)

**Throughput**

- Objective to have stable throughput at the correct processes and don’t worry about the faulty processes
  - Traditional = Virtual Synchronous Ensemble
  - Make 1 of the 8 processes faulty be sending it to sleep
  - Y axes: Number of messages sent (left 100 pr sec, right 150 pr sec)
  - X axes: how much of the time the faulty process sleeps

**Variance of Throughput**

- Variance grows as a function of group size
  - Group members: 16, 96, 128
  - 25% of the members are slowed down
  - Y axes: throughput and variance
  - X axes: how much the faulty members are slowed down

**Delivery Latency**

- Left: Inter arrival spacing at a receiver
- Bottom: In fifo few outliers effect the deliverance latency
- Leftmost is an unordered protocol
### Overhead

- The overhead grows for bimodal multicast, but at a constant function of the group size

![Graphs showing overhead growth](image)

### Discussion

- Saw that stability of protocol is exceptional even under heavy perturbation
- Overhead is low and stays low with system size, bounded even for heavy perturbation
- Throughput is extremely steady
- In contrast, virtual synchrony and SRM both are fragile under this sort of attack

### One protocol side-by-side with the other

- Use virtual synchrony for replicated data and control actions, where strong guarantees are needed for safety
- Use Bimodal Multicast for high data rates, steady flows of information, where longer term properties are critical but individual multicast is of less critical importance

**Examples**

<table>
<thead>
<tr>
<th>Bimodal</th>
<th>VScast</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stock quotes</td>
<td>Money transfer</td>
</tr>
<tr>
<td>Air Traffic Control track updates</td>
<td>Air Traffic Control controller handoffs</td>
</tr>
<tr>
<td>Telemetry for a hospital patient</td>
<td>Change medication orders</td>
</tr>
</tbody>
</table>

*Bimodal for high data rates, virtually synchronous for low or bursty traffic but very critical events*

### Epilogue

- Baron: “Although successful, your attack was reckless! It is impossible to achieve consensus on attack under such conditions!”
- General: “I merely took a calculated risk, for which the bimodal guarantee suffices….”
Tools build using the Bimodal Multicast protocol

- Astrolabe
- Gravitational Gossip
- Anonymous Gossip

**Astrolabe**

- Used to support distributed shared memory in the form of a hierarchical table.
- The leaves are regional tables
  - One row per participating computer or application program
  - Columns contain application-defined data (ints, objects, web pages, etc).
- The higher levels of the hierarchy are formed using summary functions.
  - A summary function is a computation on a column of a regional table that reduces the contents to a single value.
- Supports OLE-DB, so from a Windows PC you have drag-and-drop access to Astrolabe tables.

**Gravitational Gossip**

- Designed to support large number of subgroups within a single network.
- Useful for supporting publish-subscribe styles of communication.
- The group members can specify the amount of the messages it wants to receive. Like 100%, 70%, or %10 of the total number of messages send within the group.
- This reduces the cost of being a member of a group.
- Useful in settings where it is meaningful to take actions based on a randomly selected subset of sensor values.

**Anonymous Gossip**

- Used for wireless multicast in ad-hoc networks.
- Idea is to use a multicast protocol and then superimpose a gossip repair mechanism, like in Bimodal Multicast.
- Difference is that in ad-hoc networks little information is available about the identity of peers.
- Anonymous Gossip solves the problem by sending gossip messages that travel some distance over a randomly chosen path in the ad-hoc network.