A New Perspective On Group Communication in a Fully Asynchronous Distributed Setting

Andrey Ermolinskiy
Mohit Chawla
CS 294 Project Poster
November 28
Motivations

- Building distributed systems that satisfy formal correctness criteria (such as safety and liveness) is very hard in a fully asynchronous environment.
  - Fundamentally impossible to tell the difference between failure and non-responsiveness. ➔ Unreliable failure detectors.
  - System designers are forced to introduce synchrony assumptions (e.g., bounds on communication latency) that may not always hold.

- Traditional approaches sacrifice safety or performance.
  - Taking an action based upon a premature failure observation or a delayed message may move the system into an unsafe state.
  - The Virtual Synchrony model for group communication builds safety on top of an unreliable failure detector, but faces scalability challenges.

- This project proposes a new primitive for safe and lightweight group communication in an asynchronous distributed environment:
  - The Failure-Causal Communication model (FC-Comm) protects against unreasonably delayed messages and incorrect failure observations.
  - Main idea: track causal dependencies between failure notifications and updates to application state.
Related Work: Virtual Synchrony

- The VS model provides atomic multicast delivery to a process group.
- Each process is provided with a group membership view that changes as other processes join and leave.
- VS enforces several constraints on the delivery order of data messages and group membership events:
  
  (1) All processes observe a single, globally-agreed upon sequence of view changes.

  (2) Message delivery never crosses view boundaries. If $P$ observes delivery of message $M$ in view $V$ before advancing to $(V+1)$ then every other member of $V$ will also observe $M$ before advancing.

Virtual Synchrony in the Real World

AEGIS (U.S. Navy)
Our Assumptions

- Fully asynchronous message-passing environment.
  - Unbounded message propagation delays and clock drift rates.
- Reliable FIFO channels for pairwise communication between processes (e.g., TCP).
- Any process may fail by crashing at any time.

Our Claim

- **FC-Comm** is a useful general-purpose building block that encapsulates the complexities of asynchrony and reduces implementation effort for application developers without the performance penalty of VS.

- We use the classic hard problem of distributed mutual exclusion as a motivating example and demonstrate a very simple and lightweight locking protocol that relies on FC-Comm to achieve both safety and progress. **Eventual goal**: a provably safe Chubby-like lock service.
FC-Comm (Overview)

- FC-Comm provides a combined message delivery and failure notification service. Tracks the set of failure observations that may have causally affected the state of the local process.

### FC-Comm API

- `FCSendMsg(destProcID, msg)`;
- `FCEnableFailureNotif(procID)`;
- `FCDisableFailureNotif(procID)`;

### FC-Comm Events

- `MsgArrivalEvent<srcProcID, msg>`: Signals arrival of an application-level message from another group member.
- `FailureSuspicionEvent<procID>`: Posted by the failure detector upon expiration of the keep-alive timeout for a remote process.
- `SendMsgStatusEvent<SeqNum, Status>`: Reports message delivery status to the sender. 
  - **Status** is one of:
    - **OK** - Message delivered successfully.
    - `UpdateFailureSet<F>` - Sender is missing some failure notifications.
- `GroupEvictEvent`: Reports group eviction to the local process.

### FC-Comm State Variables

- **Int groupSize**
- **Int localProcID**
- **Queue eventQueue**: Queue of events waiting to be delivered to the application process.
- **Bitmap FIS[groupSize]**: Failure Interest Set (indexed by procID).
- **Bitmap FCS[groupSize]**: Failure Causality Set (indexed by procID).
FC-Comm (Event Ordering Constraints)

- Let $G$ denote the set of all process group members. If $M$ is a message sent by $p_S \in G$ to $p_R \in G$, let $F_c(M)$ denote the subset of group members such that:
  $$p_f \in F_c(M) \iff \exists \text{ process } p \in G \text{ such that } p \text{ has observed }\text{FailureSuspicionEvent}^{<p_f>} \text{ and this observation may have causally affected } M.$$

- Then for any two processes $p_S, p_R \in G$ and any message $M$ sent by $p_S$ to $p_R$, FC-Comm guarantees the following invariants on their delivery order:
  1. $p_R$ will observe $\text{MsgArrivalEvent}^{<p_S,M>}$ only after observing $\text{FailureSuspicionEvent}^{<k>}$ for all $k \in F_c(M)$.
  2. $p_R$ will not observe $\text{MsgArrivalEvent}^{<p_S,M>}$ if the state of $p_R$ at the time of $M$'s arrival is causally dependent on some $\text{FailureSuspicionEvent}^{<p_f>}$ observed by some process $p \in G$, for some $p_f \notin F_c(M)$.

Implementing the FC-Comm model

- Sender ($p_s$) augments each outgoing message $M$ with $F_c(M)$.
- Receiver ($p_R$) delivers all failure notifications in $F_c(M)$ before delivering $M$.
- $p_R$ rejects $M$ if its local state depends on some $\text{FailureSuspicionEvent}^{<k>}$ for some $k \notin F_c(M)$. 
Solving Distributed Mutual Exclusion with FC-Comm

- **Distributed Mutual Exclusion** - problem of managing access to a single indivisible resource that can be accessed by at most one process at a time.

**Problem Definition**

- We have a set of client processes \( C = \{C_0, C_1, C_2, ..., C_n\} \) that require interaction with some shared resource \( R \) (also a process).
- Each client \( C_i \) holds an atomic request sequence \( S_i = <r_{i1}, r_{i2}, ..., r_{ik}> \) that represents a sequence of operations to be executed by \( R \) on \( C_i \)'s behalf.
- We say that the state of \( R \) is **consistent** at a given instant if the sequence of requests observed by \( R \) is some serial ordering of \( \{\rho(S_i), \rho(S_j), \rho(S_k), ...\} \) for distinct \( \{i, j, k, ....\} \), where \( \rho(S_i) \) denotes some prefix of \( S_i \).
- **Goal**: execute all request sequences, ensuring that \( R \) remains consistent at all times.

- **FC-MUTEX** is a simple and lightweight locking protocol that solves distributed mutual exclusion and relies on FC-Comm to achieve both safety and liveness.
Demonstrating Correctness of FC-MUTEX

Proof outline

(Details in http://www.cs.berkeley.edu/~andreye/cs294/fc-comm-spec.pdf)

We must demonstrate that every valid execution history satisfies:

- **Safety**: The state of \( R \) remains consistent at all times.
  
  (1) Show that for any request sequence \( S_i \), the subset of requests from \( S_i \) that \( R \) may observe in a valid history always constitutes some prefix of \( S_i \).
  
  (2) Show that if \( R \) observes some Request\(<r_i^a>\) from client \( C_i \) before observing Request\(<r_j^b>\) from \( C_j \) then all requests \(<r_i^*>\) from \( C_i \) are observed by \( R \) before \(<r_j^b>\).

  Together, (1) and (2) imply safety.

- **Liveness**: Every non-faulty client makes progress and eventually terminates by observing COMPLETION or GROUP_EVICT.

  It suffices to demonstrate that every non-faulty client \( C_i \) will eventually observe a \(<\text{LockGranted}>\) message from \( L \) in every valid execution history.

- **Non-triviality**: No correct client observes GROUP_EVICT unless a violation of synchrony bounds occurs.

  Group eviction can occur only after an erroneous failure observation.
Comparing FC-Comm to Virtual Synchrony

- System-wide overhead of a single failure observation
  - Suppose $P_1$ observes $\text{FAILURE}(P_2)$ in a group with $N$ processes.

  **Advancing the membership view in the Virtual Synchrony model**
  - $P_1$ multicasts $\text{ViewChange}(\text{FAILURE}(P_2))$ to the group. **(N-1) messages.**
  - Upon receiving $\text{ViewChange}$, each group member:
    - Marks $P_2$ as faulty and quiesces the application process.
    - Multicasts copies of all *unstable* messages to the group, followed by a $\text{FLUSH}$ message. **$(N-1)^2 \cdot (M + 1)$**, where $M$ is # of messages sent by $P_2$.
    - Upon receiving $\text{FLUSH}$ from every other process, installs the new view and resumes the application.

  Quadratic communication overhead and unbounded application delay

---

**Failure notification in FC-Comm**

FC-Comm delivers $\text{FailureSuspicionEvent}<P_2>$ to $P_1$ and updates its FCS.

Local action

**FC-Comm overhead**

- **Communication**: every data message $M$ carries its $F_c(M)$.
- **Storage**: FIS and FCS for each member process.
Summary and Future Work

- **FC-Comm** is a new group communication model for distributed applications.
  - **Primary goal**: Minimize the impact of delayed messages and premature failure observations that make it hard to **achieve correctness** in a **fully asynchronous** setting.
  - **Main idea**: **Track causal dependencies between failure observations and application state**.
  - Reduces implementation complexity for application developers and obviates the need for ad-hoc application-specific mechanisms.

- We have implemented a proof-of-concept prototype on Linux.
  - **FC-Comm**: 1150 lines of C code (implemented as a shared library).
  - State machine implementation of **FC-MUTEX**: 400 lines of C code.

- **Future work**
  - Apply the **FC-Comm** model to other hard problems (e.g., **general state machine replication**).
  - Explore recovery from incorrect failure observation through snapshots and coordinated rollback. Examine performance trade-offs between group eviction and rollback.