YACA: Yet Another Chat Application

Group 23: Simon Egger and Nicolai Krebs

1 Architecture Design

1.1 Primitives

Notation We use V to denote the (finite) set of server processes. To coincide with the notation of the lecture and [1], we sometimes use g do denote the server group.

Messages In general, all messages inherit the base class Message, which includes fields for the message Header, Content, and Metadata. Furthermore, base functionalities include encoding and decoding the messages, using JSON. The message type is uniquely identified by the header, which tells the process how the message's content should be processed and which metadata is included. Examples for included metadata are digital signatures, sequence numbers, piggybacked acknowledgements, or nonces.

Delivery Queues Another useful primitive is the FIFO delivery queue. Our general approach for processing incoming messages is to have one listening thread per listening port, which handles configuration messages (e.g. heartbeat messages, negative acknowledgements, or proposal messages for TO-multicast) and forwards only the relevant messages via a delivery queue to one or multiple worker threads. Therefore, we use locks to realize this in a thread-safe manner. Furthermore, we use semaphores to provide a blocking consume and a non-blocking produce operation.

Reliable Multicast (R-Multicast) In general, we implement R-multicast using a combination of reliable multicast over IP multicast, as described in [1, p. 649]. Each process $p \in V$ manages a local sequence number S_p, which is initialized as zero and is atomically incremented each time p sends a message, and a dictionary $R_g : V \to \mathbb{N}$, where $R_g(q)$ keeps track of q's last delivered message sequence number (for each $q \in V$). When sending a message (via IP multicast), p includes both S_p and $R_g^p := \{(q, R_g(q)) | q \in V\}$ as metadata. Therefore, when a message m is received with the attached sequence number S by some process q, there are three possible cases:

i) $S \leq R_g(q)$: p already delivered m and simply discards the duplicate
ii) $S = R_g(q) + 1$: p delivers m and additionally stores it indefinitely
iii) $S > R_g(q) + 1$: p detects that there are missing messages and places m in a holdback-queue

Note that piggybacking R_g^p to a message implicitly acts as an acknowledgement. As this only works if p continuously sends messages, we periodically send heartbeat messages as well.

If p detects missing messages, by inspecting piggybacked acknowledgements of normal messages or heartbeat messages, it sends negative acknowledgements to the sender (which is not necessarily the original sender of the message). Since each message is digitally signed, we assume that messages forwarded in this manner cannot be manipulated by the forwarding process (which is an additional guarantee on top of the checksums used internally for IP multicast). After missing messages are delivered, p checks its holdback-queue for messages that are now able to be delivered.

Similar to the original protocol described in [1, p. 649], it is easy to see that above protocol satisfies the following properties:

- *Integrity:* Duplicates are detected by inspecting the attached sequence number. IP multicast uses checksums, and we use additional digital signatures to ensure that the received message is identical to the message signed by the sending process.
- *Validity:* Trivially satisfied by the usage of IP multicast.
- *Agreement:* Each process is able to detect that some process q delivered m by inspecting R_g^q attached to the following messages. Since q stores m indefinitely, p can request the missing message by sending negative acknowledgements.

1.2 Causal-Ordered Reliable Multicast (CO-R-Multicast)

The protocol above can now be extended to support causal ordering by using the strategy described in [1, p. 657-658]: Each process manages a further holdback-queue and an additional vector stamp $R_g^{CO} : V \to \mathbb{N}$, where $R_g^{CO}(q)$ manages the last CO-R-delivered message sequence number of process q. When process p R-delivers a message m of sender q with attached vector stamp R and sequence number S, it places m in the second holdback-queue until the contraints

i) $R_g^{CO}(q) + 1 = S$, and
ii) $R_g^{CO}(q') \geq R(q')$ for each $q' \in V \backslash \{q\}$

are satisfied. The proof of correctness is identical to the one given in [1, p. 658].

1.3 Suspicion Protocol

When having malicious servers in our system it is advantageous to have a protocol that is able to suspend them, if inconsistent or incorrect behavior is detected. In this context, suspending a server means that no more incoming messages are accepted anymore, or to be more exact, no new message is accepted from the suspended server that was not already acknowledged by another server. This can easily be achieved by using digital signatures and only accepting messages that are gathered via negative acknowledgements.

One straight-forward strategy to achieve an agreed suspension is to send *suspect messages* to all other servers and suspend a server as soon as at least one suspect message has arrived. To avoid having a malicious server sending suspect messages for all servers and suspending them, we modify this protocol to allow for $f < n/3$ malicious servers (since we utilize the Phase-King algorithm this limit is of course capped by $f < \lceil n/4 \rceil$):

– If detecting inconsistent or incorrect behavior (e.g. timeout on response), send a suspect message. The suspect message contains the suspected server identification and the reason why the server is suspected (e.g. "proposal timeout on *msg_id*" for the ISIS algorithm).
– If a server gathers more than f suspect messages for the same server and the same reason, he sends a suspect message himself if he did not already send one.
– If a server gathers at least $n - f$ suspect messages, the server is suspended.

For the proof of correctness, it is easy to see that if a server gathers $n - f$ suspect messages, he can be sure that all other correct processes gather at least $n - 2f > f$ (for $f < n/3$) suspect messages of the same kind. Thus, eventually, every correct process eventually sends a suspect message and suspends the server. On the other hand, if a server receives at least f suspect messages, he can be sure that at least one correct (or honest) server is suspecting the accused server.

1.4 Phase-King Algorithm

Generally, we implement the Phase-King algorithm, as presented in the lecture, on top of CO-R-multicast, which ensures that the messages of different rounds are separated more elegantly. Furthermore, we use the suspicion protocol described above if a server detects a timeout while waiting for a message. Then, the accused server is either suspended by all correct servers, or the missing message is eventually delivered by using negative acknowledgements.

Furthermore, we ensure that the Phase-King algorithm is only executed when each correct server has the same group-view. During the execution, we therefore do not allow new servers to join (TODO reference).

1.5 Total-Ordered Reliable Multicast (TO-R-Multicast)

We build TO-R-multicast on top of CO-R-multicast using the ISIS algorithm described in [1, p. 655-656]. We use CO-R-multicast for the following three reasons:

i) Proposal messages are delivered after the original message
ii) To support our JOIN mechanisms of new servers (TODO reference)
iii) To perform the Max-Phase-King algorithm (cf. Appendix A.1) on the agreed sequence numbers

This, however, does not guarantee that the total-ordered messages satisfy causal-ordering as well.

Each server p maintains variables P_g^p and A_g^p. Upon sending a message via TO-R-multicast, the sender adds a randomly generated message ID and CO-R-multicasts it. When a server CO-R-delivers such a message, he responds with a proposal $P_g^p = max(A_g^p, P_g^p) + 1$ for the message's sequence number. This is also done via CO-R-multicast, as opposed to unicast [1, p. 655-656]. This allows us to use the Max-Phase-King algorithm directly as described in Appendix A.1.

References

1. George Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. *Distributed Systems: Concepts and Design.* Addison-Wesley Publishing Company, USA, 5th edition, 2011.

A Appendix

A.1 Max-Phase-King Algorithm

```
function MaxPhaseKing($v_{init}$: int, $f$: int) for $p_i \in V$ begin
    // Preparation
    broadcast $v_{init}$ to all processes
    await value $v_j$ from each process $p_j \in V$
    $\tilde{v}$ ← $max\{v_j \mid p_j \in V\}$
    $v$ ← $\tilde{v}$

    for $phase$ = $1, \dots, f+1$ do
        // Round 1
        broadcast $v$ to all processes
        await value $v_j$ from each process $p_j \in V$
        $mode$ ← mode of the $v_j$'s
        $mult$ ← number of times that $mode$ occurs
        $median$ ← median of the $v_j$'s

        // Round 2
        if $i = phase$ then
            broadcast $\lceil median \rceil$ to all processes
        receive $tiebreaker$ from $p_{phase}$
        if $mult > |V|/2 + f$ then
            $v$ ← $mode$
        else if $|\{v_j \mid v_j \leq tiebreaker\}| > f$
            $v$ ← $tiebreaker$
    done

    output $v$
end
```

Notation As before, we use V to denote the set of all processes. Let $n := |V|$ and $f < \lceil n/4 \rceil$ be the number of malicious processes. We denote $V_g \subseteq V$ as the set of correct processes and use the notation $v^{(p,k)}$ to denote the value of v for process p in phase k ($k = 0$ is used for the preparation phase). If the variable does not change with different phases, we omit k.

Proof of Correctness Our main goal is to show that all correct processes output the same value v and that $v \geq v_{init}^{(p)}$ for all $p \in V_g$.

Lemma 1. *Let p_i be the phase-king. If p_i is correct then $|\{v_j^{(p_k,i)} \mid v_j^{(p_k,i)} \leq tiebreaker^{(p_k,i)}\}| > f$ for all correct processes $p_k \in V_g$.*

Proof. If p_i is correct, he gathers $\{v_j^{(p_i,i)} \mid p_j \in V\}$. After rewriting these values as

$$v_{(1)} \leq v_{(2)} \leq \ldots \leq v_{(n)},$$

the median is computed as $median = v_{(\frac{n+1}{2})}$ if n is odd, or $median = (v_{(\frac{n}{2})} + v_{(\frac{n}{2}+1)})/2$ if n is even. In both cases we have that

$$|\{v_j^{(p_i,i)} \mid v_j^{(p_i,i)} \leq \lceil median \rceil\}| \geq \left\lceil \frac{n}{2} \right\rceil,$$

which in turn yields:

$$\begin{aligned} |\{v_j^{(p_k,i)} \mid v_j^{(p_k,i)} \leq tiebreaker^{(p_k,i)}\}| &\geq |\{v_j^{(p_i,i)} \mid v_j^{(p_i,i)} \leq \lceil median \rceil\}| - f \\ &\geq \left\lceil \frac{n}{2} \right\rceil - f \\ &> \left\lceil \frac{n}{2} \right\rceil - \left\lceil \frac{n}{4} \right\rceil \\ &\geq \left\lceil \frac{n}{4} \right\rceil - 1 \qquad \geq f \end{aligned}$$

□

Lemma 2. *Let p_i be the phase-king. If p_i is correct, then all correct processes share the same value for v after the current phase.*

Proof. Let p_j and p_k be arbitrary correct processes. Due to Lemma 1, v is updated with the *mode* or the *tiebreaker* value. Therefore, there are three cases to consider:

i) p_j and p_k use their *mode* value: Then, p_j received $mode^{(p_j,i)}$ more than $n/2 + f$ times, which implies that p_k also received $mode^{(p_j,i)}$ more than $n/2$ times. Therefore, there is no other possibility than $mode^{(p_k,i)} = mode^{(p_j,i)}$.
ii) p_j uses its *mode* value and p_k uses the *tiebreaker* value: As before, p_j received $mode^{(p_j,i)}$ more than $n/2+f$ times, which implies that the phase-king p_i also received $mode^{(p_j,i)}$ more than $n/2$ times. Therefore, we have $median^{(p_i,i)} = mode^{(p_j,i)} \in \mathbb{N}$, which is then received by p_k as the *tiebreaker*.
iii) p_j and p_k use the *tiebreaker* value: As p_i is assumed to be correct, both receive the same *tiebreaker* value.

□

Lemma 3. *Let $1 \leq i \leq f + 1$ be a phase with a correct phase-king. Then, the following holds true:*

$$\forall p_j, p_k \in V_g \forall phase \in \{i, \ldots, f+1\} : v^{(p_j,phase)} = v^{(p_k,phase)}$$

That is, once there was a correct phase-king, all correct processes continue to share the same value for v afterwards.

Proof. Lemma 2 handles the case of $phase = i$. Afterwards, all correct processes broadcast the same value of v, which implies that $mult^{(p_j,phase)} \geq n-f > n/2+f$ is satisfied for all $p_j \in V_g$ and $i < phase \leq f+1$. Using the same argument as *i*) in the proof of Lemma 2, it is easy to see that the statement holds true. □

Lemma 4. *The following statement holds true:*

$$\forall p_j, p_k \in V_g \forall phase \in \{0, \dots, f+1\} : v^{(p_j,phase)} \geq v_{init}^{(p_k)}$$

That is, the value of v for a correct process p_j is always larger than the initial values of all correct processes.

Proof. We use an inductive argument to prove this statement. For $phase = 0$, $v^{(p_j,0)}$ is initialized in line 6. As all correct processes p_k broadcast their initial value $v_{init}^{(p_k)}$ the statement is trivially satisfied.

For $phase > 0$, $v^{(p_j,phase)}$ there are three cases to consider:

i) $v^{(p_j,phase)}$ is not updated, i.e., $v^{(p_j,phase)} = v^{(p_j,phase-1)}$: By the induction hypothesis, the statement holds true.

ii) $v^{(p_j,phase)}$ is updated in line 21. Then $mult^{(p_j,phase)} > n/2+f$ which implies that at least one correct process p_l sent $mode^{(p_j,phase)}$ at the beginning of the phase. Using the induction hypothesis, we have:

$$v^{(p_j,phase)} = mode^{(p_j,phase)} = v^{(p_l,phase-1)} \geq v_{init}^{(p_k)}, \quad \forall p_k \in V_g$$

iii) $v^{(p_j,phase)}$ is updated in line 23. Let

$$C = \{v_k^{(p_j,phase)} \mid v_k^{(p_j,phase)} \leq tiebreaker^{(p_j,phase)}\},$$

then the condition in line 22 states $|C| > f$. Therefore, there exists at least one correct process p_l with $v_l^{(p_j,phase)} \in C$. This yields:

$$v^{(p_j,phase)} = tiebreaker^{(p_j,phase)} \geq v_l^{(p_j,phase)} = v^{(p_l,phase-1)} \geq v_{init}^{(p_k)},$$

for all $p_k \in V_g$.

□

Theorem 1. *Each correct process p_j that executes the Max-Phase-King algorithm outputs $v^{(p_j,f+1)}$ that satisfies*

i) $\forall p_k \in V_g : v^{(p_j,f+1)} = v^{(p_k,f+1)}$, *and*

ii) $\forall p_k \in V_g : v^{(p_j,f+1)} \geq v_{init}^{(p_k)}$.

Proof. Lemma 3 proves statement *i*) and Lemma 4 proves statement *ii*). □