Principles of Dependable Distributed Systems
Spring term 2010

Lecture 12: Service Replication
(State-Machine Approach)
June 2, 2010

Zinaida Benenson
Dependable Distributed Systems Lab
http://pi1.informatik.uni-mannheim.de
University of Mannheim
Course Plan

• Motivation, examples ✔
• System models ✔
• Algorithms for systems with crash failures ✔
  – Reliable broadcast ✔
  – Logical time, causal broadcast ✔
  – Consensus ✔
  – Consensus variants ✔
    • Total order broadcast, atomic commit, terminating reliable broadcast ✔
  – Group communication ✔
• Protocols for arbitrary (Byzantine) failures ✔
• Replication using state-machine approach
Dependable Distributed Systems

- Two most important paradigms for building dependable distributed systems:
  - (1) Group Communication (Lecture 10)
    - Reliable Broadcast + Group Membership
      - We considered crash failures
    - Protocols for Byzantine failures also exist:
      - Byzantine Generals (= reliable broadcast)
      - Byzantine Agreement (= consensus ⇒ implement group membership)
  - (2) State Machine Replication
    - Can be used to make any service fault-tolerant (dependable)
      - Crash and Byzantine failures
    - Prerequisite: service is implemented as a “state machine”
General Scenario

- A set of clients, a single server
  - Server is single point of failure
- Server should be made highly available through replication
Server is a State Machine

• Given: Arbitrary **state machine**
  – Consists of **state variables** (encode state)
  – Set of **commands** (transform state)
• **Clients** of the state machine can issue request events
  – \(<\text{state\_machine}, \text{command}, \text{arguments}>\rangle\)
• State machine satisfies some **safety** and some **liveness** properties
• Example: **memory**
  – State: Array of memory words
  – Commands:
    • read(location)
    • write(location, value)
Memory

Client p1

write(2,42)

read_req(2)

Client p2

read_ind(2,42)

Memory
Goal and Idea

• Implement transparent fault-tolerance
  – Simulate a single server
  – Simulated server is highly available

• Conditions:
  – Clients must be able to issue requests as before
  – Results satisfy the safety and liveness properties of the state machine (server)

• Idea: Replicate state machine \( n \) times

• Assumptions:
  – Maximum \( t \) faults of replicas
  – Crash failures (Byzantine considered later)
  – Clients correct (faulty clients considered later)
State Machine Assumptions

• Requests are processed \textit{atomically}
• Requests are processed consistent with \textit{causality}
  – Common sense requirement
• Results of requests are \textit{deterministic}
State Machine Approach

- Degree of replication for t crash-stop faults: \( n = t + 1 \)
- Frontend uses (crash-stop tolerant) total order broadcast
- Backend waits for first response
Abstract Architecture

(1) request

(2) total order broadcast

(3) replicas process request

(4) responses sent back

(5) backend sorts out responses

(6) response

client p1

frontend

backend

replica 1

replica 2

replica n

client p2

frontend

backend
Architecture Overview

• State machine is replicated \( n \) times
  – Degree of replication determined by \( t \)

• Clients equipped with \textit{frontend}
  – Frontend distributes request using crash-tolerant (\textit{uniform}) terminating reliable broadcast

• Client equipped with \textit{backend}
  – Backend collects and processes command responses
  – Type of processing determined by \( t \)
Correctness

• Given safety property $S$ and liveness property $L$ of state machine

• Does the state machine approach guarantee $S$ and $L$ in the presence of $t$ crash faults?
  – $S$ is an arbitrary safety property on the interface of the state machine
  – $L$ is an arbitrary liveness property

• How can we argue?
Liveness

• Compare replicated system with non-replicated system
  – $L$ is a property satisfied in finite time
  – Any execution of the system has “a good ending”
Liveness Proof Sketch

• Client issues a request
• If it expects no response, finished
• If it expects a response, in the non-replicated setting eventually some response is received
• We need to argue that at least one response is received in replicated setting
  – Guaranteed through $n = t + 1$ and processing of the back end
  – In the worst case, $t$ replicas fail
  – The final replica will send the response
Safety

- S is an arbitrary safety property
  - If there is a response, it satisfies a certain predicate (possibly depending on request)
- Compare with non-replicated setting
  - Note that (due to asynchrony) service may be non-deterministic although server is deterministic
Some Lemmas

- **Lemma 1**: Every replica goes through the same sequence of states until it crashes
  - **Proof**:
    - Use properties of total order broadcast (uniform total order delivery)
    - Also necessary: state machines are deterministic

- **Lemma 2**: Sequence of states of a replica correspond to a possible sequence of states of a non-replicated server
  - **Proof**:
    - Schedule messages at non-replicated server appropriately
Safety Proof Sketch

• Assume: Client receives a response
• Prove: Response corresponds to one in a non-replicated setting

• Proof:
  – Response was issued by some replica
    • Follows from algorithm
  – Identity of replica does not matter
    • Follows from Lemma 1
  – Response corresponds to a “real” response
    • Follows from Lemma 2
Byzantine Failures

• Results sent by Byzantine processes can be arbitrary
  – Not correct to wait for first result
• What degree of replication do we need for maximum of $t$ Byzantine replicas?
Byzantine Failures

- Results sent by Byzantine processes can be arbitrary
  - Not correct to wait for first result
- What degree of replication do we need for maximum of $t$ Byzantine replicas?
  - $n=2t+1$
Optimizations for Crash Failures

• For read-only requests, uniform total order broadcast can be weakened in crash-stop case
  – Sufficient that one correct replica process receives the request

• Uniform total order of broadcast can be relaxed
  – If the order of processing request $r$ and request $r'$ does not matter (if $r$ and $r'$ commute)
Faulty Clients

- What if clients can fail?
- Replicate clients:
  - Need a voter at the server replicas that combines output of replicated clients

![Diagram showing client replicas communicating with server replicas via voters.](image-url)
Tolerating Faulty Backends

• What if the backend can fail?
  – Replicate backends
  – Somewhere a "single value" has to be computed
Maintenance

• **Reconfiguration:**
  – System can be reconfigured as long as redundancy
does not run out
  – Sometimes it is non-trivial to identify faulty components!

• Replace failed components as long as
  – At least one component remains operational (for crash
faults)
  – A majority of components remains correct (for Byzantine
faults)

• **Integration:** new replicas need to be brought into
the same state as other replicas before they can
process requests
Replication Summary

• Material based on Fred B. Schneider's tutorial in ACM Computing Surveys, 22 (4), Dec. 1990
  – Original ideas usually attributed to Lamport (Time, clocks and the ordering of events in a distributed system, Comm. ACM, 1978)

• State machine approach is an established engineering paradigm for transparent fault-tolerance
  – Used in almost all high-reliability settings (e.g., Space Shuttle)
Lecture Wrap-up

• Motivation, examples ✔
• System models ✔
• Algorithms for systems with crash failures ✔
  – Reliable broadcast ✔
  – Logical time, causal broadcast ✔
  – Consensus ✔
    • FLP impossibility, weakest failure detector ✔
  – Consensus variants ✔
    • Total order broadcast, atomic commit, terminating reliable broadcast ✔
  – Group communication ✔
• Protocols for arbitrary (Byzantine) failures ✔
• Replication using state-machine approach ✔