Atomicity

Bailu Ding

Oct 18, 2012

Outline

Introduction

- Implementing Fault-Tolerance Services Using State Machine Approach
- Sinfonia: A New Paradim for Building Scalable Distributed Systems
- The Dangers of Replication and a Solution

Outline

State Machine

- Server
  - State variables
  - Commands
  - Example: memory, reads, writes.
  - Outputs of a state machine are completely determined by the sequence of requests it processes
- Client
- Output

Causality

- Requests issued by a single client to a given state machine are processed by the order they were issued
- If request r was made to a state machine sm caused a request r' to sm, then sm processes r before r'

Fault Tolerance

- Byzantine failures
- Fail-stop failures
- t fault tolerant

Fault-Tolerant State Machine

- Replicate state machine
- t fault tolerant
    - Byzantine: 2t+1
    - Fail-stop: t+1

Replica Coordination

Requriements

- Agreement: receive the same sequence of requests
- Order: process the requests in the same relative order

Agreement

- Transmitter: disseminate a value to other processors
- All nonfaulty processors agree on the same value
- If the transmitter is nonfaulty, then all nonfaulty processors use its value as the one on which they agree

Order

- Each request has a unique identifier
- State machine processes requests ordered by unique identifiers
- Stable: no request with a lower unique identifier can arrive

Challenge

- Unique identifier assignment that satisfies causality
- Stability test

Order Implementation

Logical Clocks

- Each event e has a timestamp $T(e)$
- Each processor p has a counter $T(p)$
- Each message sent by p is associated with a timestamp $T(p)$
- $T(p)$ is updated when sending or receiving a message
- Satisfy causality
- Stability test for fail-stop failures
  - Send a request r to processor p ensures $T(p) > T(r)$
  - A request r is stable if $T(p) > T(r)$ for all processors

Order Implmentation

Synchronized Real-Time Clocks

- Approximately synchronized clocks
- Use real time as timestamps
- Satisfy causality
  - No client makes two or more requests between successive clock ticks
  - Degree of clock synchronization is better than the minimum message delivery time
- Stability test I: wait after delta time
- Stability test II: receive larger identifier from all clients

Order Implementation

Replica-Generated Identifiers

- Two phase
  - State machine replicas propose candidate unique identifiers
  - One of the candidates is selected
- Communication between all processors are not necessary
- Stability test:
  - Selected candidate is the maximum of all the candidates
  - Candidate proposed by a replica is larger than the unique identifier of any accepted request
- Causality: a client waits until all replicas accept its previous request

Faulty Clients

- Replicate the client
- Challenges
  - Requests with different unique identifiers
  - Requests with different content

Reconfiguration

- Remove faulty state machine
- Add new state machine

Outline

Sinfonia

- Two Phase Commit
- Sinfonia

Two Phase Commit

Problem

All participate in a distributed atomic transaction commit or abort a transaction

Two Phase Commit

Problem

All participate in a distributed atomic transaction commit or abort a transaction

Challenge

A transaction can commit its updates on one participate, but a second participate can fail before the transaction commits there. When the failed participant recovers, it must be able to commit the transaction.

Two Phase Commit

Idea

Each participant must durably store its portion of updates before the transaction commits anywhere.

- Prepare (Voting) Phase: a coordinator sends updates to all participants
- Commit Phase: a coordinator sends commit requests to all participants

Motivation

Problem

- Data centers are growing quickly
- Need distributed applications scale well
- Current protocols are often too complex

Idea

New building block

Scope

- System within a data center
  - Network latency is low
  - Nodes can fail
  - Stable storage can fail
- Infrastructure applications
  - Fault-tolerant and consistent
  - Cluster file systems, distributed lock managers, group communication services, distributed name services

Approach

Idea

What can we sqeeuze out of 2PC?

Approach

Idea

What can we sqeeuze out of 2PC?

Observation

For pre-defined read set, an entire transaction can be piggybacked in 2PC.

Approach

Idea

What can we sqeeuze out of 2PC?

Observation

For pre-defined read set, an entire transaction can be piggybacked in 2PC.

Solution

Minitransaction: compare-read-write

Minitransaction

Minitransaction

- Compare items, read items, write items
- Prepare phase: compare items
- Commit phase: if all comparison succeed, return read items and update write items; otherwise, abort.

Minitransaction

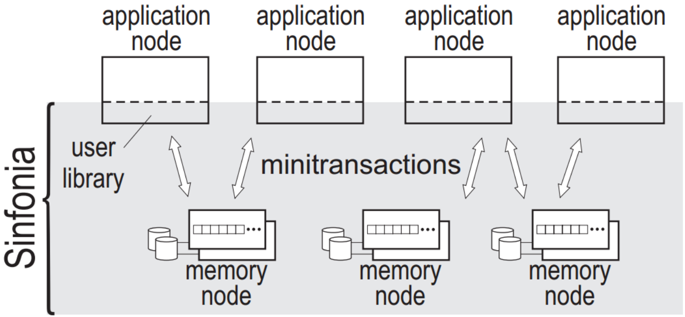
Minitransaction

- Compare items, read items, write items
- Prepare phase: compare items
- Commit phase: if all comparison succeed, return read items and update write items; otherwise, abort.

Applications

- Compare and swap
- Atomic read of multiple data
- Acquire multiple leases
- Sinfonia File System

Architecture



Fault Tolerance

- App crash, memory crash, storage crash
- Disk images, logging, replication, backup

Outline

Contribution

Dangers of Replication

A ten-fold increase in nodes and traffic gives a thousand fold increase in deadlocks or reconciliations.

Solution

- Two-tier replication algorithm
- Commutative transactions

Existing Replication Algorithms

Replication Propagation

- Eager replication: replication as part of a transaction
- Lazy replication: replication as multiple transactions

Replication Regulation

- Group: update anywhere
- Master: update the primary copy

Analytic Model

Parameters

- Number of nodes (*Nodes*)
- Number of transactions per second (*TPS*)
- Number of items updated per transaction (*Actions*)
- Duration of a transaction (*Action_Time*)
- Database size (*DB_Size*)
- Serial replication

Analysis of Eager Replication

Single Node

- Concurrent Transactions:
  $Transactions = TPS \times Actions \times Action\_Time$
- Resource: $Transactions \times Actions/2$
- Locked Resource: $Transactions \times Actions/2/DB\_Size$
- Probability of Waits Per Transaction:
  $PW = (1 - Transactions \times Actions/2/DB\_Size)^{Actions} \approx Transactions \times Actions^2/2/DB\_Size$
- Probability of Deadlocks Per Transaction: $PD \approx PW^2/Transactions = TPS \times Action\_Time \times Actions^5/4/DB\_Size^2$
- Deadlock Rate Per Trasction:
  $DR = PD/(Actions \times Action\_Time) \approx TPS \times Actions^4/4/DB\_Size^2$
- Deadlock Rate Per Node:
  $DT = TPS^2 \times Actions^5 \times Action\_Time/4/DB\_Size^2$

Analysis of Eager Replication

Multiple Nodes

- Transaction Duration: $Actions \times Nodes \times Action\_Time$
- Concurrent Transactions:
  $Transactions = TPS \times Actions \times Action\_Time \times Nodes^2$
- Probability of Waits Per Transaction: $PW_m \approx PW \times Nodes^2$
- Probability of Deadlocks Per Transaction:
  $PD_m \approx PW^2 / Transactions = PD \times Nodes^2$
- Deadlock Rate Per Transaction: $DR_m \approx DR \times Nodes$
- Deadlock Rate Total: $DT_m \approx DT \times Nodes^3$
- DB Grows Linearly (unlikely): $DT \times Nodes$

Analysis of Eager Replication

Master

- Serialized at the master
- No deadlocks if each transaction updates a single replica
- Deadlocks for mutiple masters

Lazy Replication

Lazy Group Replication

- No waits or deadlocks, but reconciliation.
- Reconciliation rate:
  $TPS^2 \times Action\_Time \times (Actions \times Nodes)^3/2/DB\_Size$

Lazy Master Replication

Reconciliation rate is quadratic to *Nodes*.

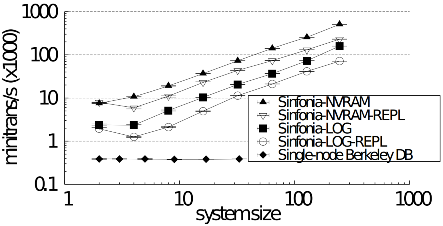
Sinfonia Revisit

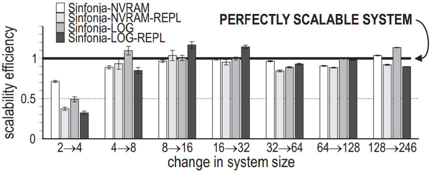
Analysis of Scalability

- The number of application nodes: App_Nodes
- The number of memory nodes: Mem_Nodes
- Total TPS: $TPS' = TPS \times App\_Nodes$
- Total DB size: $DB\_Size' = DB\_Size \times Mem\_Nodes$
- Single App/Mem node: $Rate = TPS^2 xAction\_TimexActions^5/4/DB\_Size^2$
- Multiple App/Mem nodes: $Rate' = TPS'^2 xAction\_TimexActions^5/4/DB\_Size'^2 = (App\_Nodes/Mem\_Nodes)^2 xRate$

Sinfonia Revisit

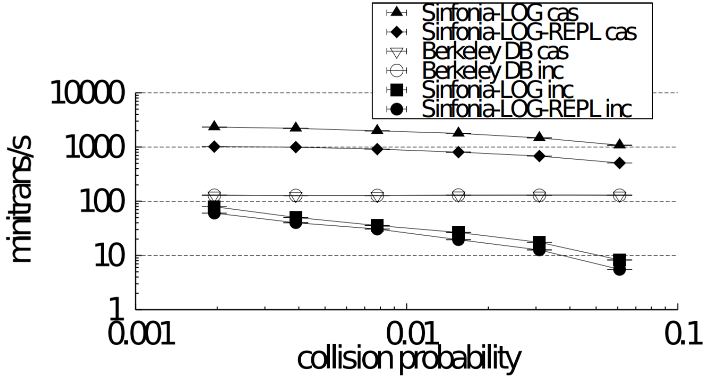
Analysis





Sinfonia Revisit

Analysis



Discussion

Parallel Eager Replication

- Transaction Duration: $Actions \times Action\_Time$
- Concurrent Transactions: $Transactions = TPS \times Actions \times Action\_Time \times Nodes$
- Probability of Waits Per Transaction: $PW_p \approx PW \times Nodes$
- Probability of Deadlocks Per Transaction: $PD_p \approx PW^2 / Transactions = PD \times Nodes$
- Deadlock Rate Per Transaction: $DR_p \approx DR$
- Deadlock Rate Total: $DT_p \approx DT \times Nodes$
- DB Grows Linearly: $DT / Nodes$
- Any problem?

Discussion

Fault Tolerance

Logging v.s. Replication?

Ordering

Timestamping in recent system, i.e. Percolator?