Self-repairing Replicated Systems and Dependability Evaluation

Toronto, August 27, 2010
CANOE Workshop
So what's so great about being on-line?
I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions.
Context – Multiple Data Centers

Site X
- Node X1
  - ServiceA
- Node X2
  - ServiceB

Site Y
- Node Y1
  - ServiceC
- Node Y2

Wide Area Network

Client

Hein Meling, CANOE Workshop, Toronto, August 2010
Context – Failures will occur
Common Solution is Redundancy

Site X

Node X1
- ServiceA
- ServiceB

Node X2
- ServiceC
- ServiceB

Site Y

Node Y1
- ServiceA
- ServiceC

Node Y2
- ServiceC
- ServiceA

Wide Area Network

Clients
Middleware for Fault Tolerance

- It is difficult to support fault tolerance
  - Tolerate object, node and *network failures*

- Techniques
  - Redundancy
  - Masking failures (failover)

- Reuse fault tolerance mechanisms
  - Use a group communication system (e.g. Jgroup or Spread)

- Focus on development issues
Group Communication

Clients

Logical unit

Group of servers

Server

S1
S2
S3

Hein Meling, CANOE Workshop, Toronto, August 2010
The Group Membership Service

S1
singleton views

S2
first full view

S3
partitioning

S3 crashes; a new view is installed

Hein Meling, CANOE Workshop, Toronto, August 2010
Middleware for Fault Treatment

- Further improve the system's dependability characteristics
  - Consider: Deployment and operational aspects

- Autonomous Fault Treatment
  - Recovery from node, object and *network failures*
  - Not just tolerate faults, repair them as well
  - Without human intervention
  - *Let groups be self-healing* (deal with its own internal failures)

- Goal: Minimize the time spent in a state of reduced failure resilience
Evaluation Techniques

- Trivial performance evaluation of repair mechanism
  - For a single failure injection

- But more interesting
  - Can we find a way to quantify/predict the improvement in availability by running experiments?
  - (Without running them for many years to get the exact numbers.)
Moving to large-scale (Cloud)

- Assume now the number of services to deploy becomes very large
  - We need to find placements for the services to avoid bottlenecks
  - Multiple conflicting requirements/goals for these services
  - Placement is a multi-criteria optimization problem

- Placement becomes NP-hard
  - Centralized optimization techniques fall short quickly

- Also, if it were possible to compute the optimal placement
  - Would it still be valid when we are ready to deploy/reconfigure?

- Distributed heuristic to compute near optimal placements
  - Based on a technique called Cross-Entropy Ant System
Outline

- Introduction and motivation
- Related work
- Distributed Autonomous Replication Management (DARM)
- Simple Network Partition Evaluation of DARM
- Dependability Evaluation Technique
- Concluding remarks
Related work: Virtualization

Hein Meling, CANOE Workshop, Toronto, August 2010
Related work: Virtualization
Related work: Virtualization

Failover = Reboot/start

SPOF

Storage

Hein Meling, CANOE Workshop, Toronto, August 2010
Assumptions

- Pool of processors to host applications
- Replicated stateful applications
- (Wide area network)
- Shared-nothing architecture
  - Neither disk or main memory is shared by processes
  - Avoid distributed file systems
  - State of application must be transmitted across network
Related work: Centralized Recovery Decisions

- AQuA
  - Leader of group affected by a failure joins the centralized dependability manager to report failure
- FT CORBA
- Jgroup/ARM
  - Report failures to centralized replication manager
ARM Overview
ARM Architecture
Failure Monitoring

Replication Manager

notify(NodePresence)
notify(ReplicaFailure)
notify(ViewChange)

notify(IamAlive)

JVM

Supervision Module

Node

Factory

JVM

Supervision Module

Node

Factory

notify(NodePresence)
notify(ReplicaFailure)

Periodic

Event driven

Hein Meling, CANOE Workshop, Toronto, August 2010
Crash Failure and Recovery

- N1 crashed
- notify(ViewChange)
- N2
- notify(ViewChange)
- N3
- notify(ViewChange)
- N4
- createReplica()
- RM
- join
- notify(ViewChange)
- Leader

Hein Meling, CANOE Workshop, Toronto, August 2010
Outline

- Introduction and motivation
- Related work
- Distributed Autonomous Replication Management (DARM)
- Simple Network Partition Evaluation of DARM
- Dependability Evaluation Technique
- Concluding remarks
Why go distributed?

- Less infrastructure - less complex
- No need to maintain consistent replicated (centralized) database of deployed groups
- Less communication overhead
DARM Overview

Group A

Group B

Group C

Clients

Group leader

Factory leader

Network

Factories

Friday, August 27, 2010
Spread communication

Node A

- Spread Client
- libspread
- Spread Daemon

Node B

- Spread Client
- libspread
- Spread Daemon

Network
DARM Components

- DARM Client
- libdarm
- libspread
- Spread Daemon
- DARM Factory
The Factory Group

- Used to install replicas of a given service
- Keeps track of
  - Node availability
  - Local load of nodes
- Interacts with the DARM library
  - To install replacement replicas
- Does not maintain any state about deployed replicas
  - In case of failure: just restart factory to host new replicas
Factory group install replacement replicas

- Factory leader
- createReplica()
- createReplicaOnNode()
Replica Placement Policy

- **Purpose of replica placement policy:** Describe how replicas should be allocated onto the set of available sites and nodes

1. Find the site with the least # of replicas of the given type
2. Find the node in the candidate site with the least load; ignoring nodes already running the service

- **Objective of this policy:** Ensure available replicas in each *likely* partition that may arise
  - Avoid collocating two replicas of the same service on the same node
  - Disperse replicas evenly on the available sites
  - Least loaded nodes in each site are selected
  - (Same node may host multiple distinct service types)
Fault Treatment Policy

- **KeepMinimalInPartition:**
  - Maintain a minimal redundancy level in each partition

- **RemovePolicy:**
  - Remove excessive replicas
  - Replicas no longer needed to satisfy the fault treatment policy

- **KeepMinimalInPrimaryPartition:**
  - Maintain a minimal redundancy level in the primary partition only

- **RedundancyFollowsLoad:**
  - Increase redundancy in loaded part of the network
Crash failure–recovery behavior

Legend: ● Leader  View no. i: $v_i$

$N1$  $N2$  $N3$  $N4$

$v_0$  $v_1$  $v_2$  $v_3$

Fault treatment pending

createReplica()

Join

Hein Meling, CANOE Workshop, Toronto, August 2010
Failure-recovery with network partitioning and merging

Legend:
- Leader
- View no. i: $v_i$
The DARM Library

- libdarm wraps around libspread and intercepts
  - Connection requests to the daemon
    - To verify and finalize runtime configuration of DARM
    - Join DARM private group of the associated application
  - Message receives - SP_receive()
    - If message belongs to DARM private group pass message to DARM
    - Otherwise pass message to application
    - Call SP_receive() again: to avoid having to return control to the application without passing a message

- libdarm also provides functions to set
  - Minimum and maximum number of replicas for the group
  - The recovery and remove delays for the group
The DARM Library

- Membership messages for the DARM private group
  - Used to decide whether fault treatment is needed

- Bootstrapping applications:
  - Only a single instance of an application needs to be started
  - Assuming the application is configured with some minimum number of replicas
  - DARM will install the required number of replicas
Outline

- Introduction and motivation
- Related work
- Distributed Autonomous Replication Management (DARM)
- Simple Network Partition Evaluation of DARM
- Dependability Evaluation Technique
- Concluding remarks
Target system

Site $x$

- $E_1^{x_1}$ Factory
- $E_2^{y_1}$ Factory

Site $y$

- $E_2^{y_1}$ Factory
- $E_3^{z_1}$ Factory
- $E_4^{z_2}$ Factory
- $E_5^{z_3}$ Factory

Site $z$

- $E_3^{z_1}$ Factory
- $E_4^{z_2}$ Factory
- $E_5^{z_3}$ Factory

Network

Inject($xy \mid z$)

Fault injector

Replica

Replacement replica

Hein Meling, CANOE Workshop, Toronto, August 2010
Network Partition/Merge Experiments

- Want to determine
  - the single partition recovery durations
  - corresponding merge of partitions
    (and removal of excessive replicas)
Fast Spread; partition with 2 live replicas

Partition (2 live replicas, 1 added) – Density estimates for detection and recovery times (N=194)

- Partition detection, ($\mu=0.9$, $\sigma=0.261$)
- Replica create, ($\mu=2.9$, $\sigma=0.209$)
- Final view, ($\mu=3$, $\sigma=0.304$)
Fast Spread; partition with 1 live replica

Partition (1 live replica, 2 added) – Density estimates for detection and recovery times (N=136)

- Partition detection, ($\mu=0.9$, $\sigma=0.284$)
- Replica create, ($\mu=2.9$, $\sigma=0.288$)
- Final view, ($\mu=5$, $\sigma=0.273$)
Fast Spread; Merge, removing 2 replicas

Network merge – Density estimates for detection and remove times (N=600)

- Merge detection, ($\mu=2$, $\sigma=0.226$)
- Replica remove, ($\mu=4.1$, $\sigma=0.23$)
- Merged view, ($\mu=6.1$, $\sigma=0.22$)
Outline

- Introduction and motivation
- Related work
- Distributed Autonomous Replication Management (DARM)
- Simple Network Partition Evaluation of DARM
- Dependability Evaluation Technique
- Concluding remarks
“It’s the latest innovation in office safety. When your computer crashes, an air bag is activated so you won’t bang your head in frustration.”
Objective of Evaluation

- Provide estimates for dependability attributes:
  - Unavailability
  - System failure intensity
  - Down time
Predicting Dependability Attributes

- Use stratified sampling
- Series of lab experiments are performed
  - One or more fault injections in each experiment
    - (all faults manifest themselves as crash failures)
  - According to a homogeneous Poisson process
- Strata := the number of near-coincident failure events
  - A posteriori stratification: Experiments are allocated to different strata after experiment completion
  - Three strata: single, double, and triple failures
Predicting Dependability Attributes

- **Offline a posteriori analysis**
  - Events are recorded during experiments
  - Used to construct single global timeline of events
  - Compute trajectories on a predefined state machine

- **Analysis provide strata classification and various statistics**
  - The statistical measures are used as input to estimators for dependability attributes:
    - Unavailability
    - System failure intensity
    - Down time
Target System Illustrated

P4, 2.4GHz; Linux 2.6.3; Java JDK 5.0

createGroup(MS)  notifyFailure(MS3)

100 Mbps Ethernet

Experiment Engine

MS1 Group MS2

#5 #6

RM1 -> RM2

#1 #2

RM3

#3

MS4

#4

AS1

#8

faultInjection()
Target System – State Machine

- Failure-recovery behavior of a service
  - Modeled as a state machine (next slide)
  - Events are as seen by the service replicas

- The state machine is only used a posteriori
  - To compute statistics of the experiment
  - (not used to control fault injections)
- Fault Injection can occur in all states
  - Causes different trajectories in the state machine
- Circular states: UP
- Squared states: DOWN

![State Machine Diagram]
Measurement Approach: Timeline of events

- Place multiple processor failures close together
  - Examine system behavior of such rare events
  - (determine the rate at which they cause system failure)
  - Use these results to compute system unavailability

- (Given MTBF for a single processor)
The Failure Trajectory

Injected crash faults in the target system

$X_i(t)$

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

$X_0$ $X_1$ $X_4$ $X_5$ $X_8$ $X_6$ $X_7$ $X_3$

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Seconds

MS replica failed, RM replica failed
MS View-2, RM View-2
MS replica failed, RM replica created
MS View-1
MS View-1, RM View-3
MS replica created
MS replica created
MS View-1
MS View-2
MS View-3

2010
The Failure Trajectory

- Characteristics obtainable from the failure trajectory
  - Unavailability:
    - Down time for trajectory $i$
      \[ Y_i^d = g(X_i) = \sum_{j=1}^{m_i} I(X_{i,j} \in \mathcal{F})(t_{i,j+1} - t_{i,j}) \]
    - Unavailability
      \[ \hat{U} = \frac{E(Y^d)}{E(Y^d) + (n\lambda)^{-1}} \approx E(Y^d)n\lambda. \]
  - Probability of failure (reliability)
    - (formulas in the paper)
Experimental Strategy

- Consider multiple near-coincident failures
- Classify experiments into strata $S_k$
  - If $k$ failure events occurred in the trajectory
- Each strata sampled separately
- Collected samples for each stratum
  - Can obtain statistics for the system in that stratum
  - E.g., the expected duration of a stratum $S_k$ trajectory:

  \[ \Theta_k = E(T|S_k) \text{ and } \sigma_k = Var(T|S_k) \]
Sampling Scheme

\[ k^* \]

\[ (T|k = 1) \quad (T|k = 2) \quad (T|k = 3) \]

Processor failures

\[ t_{i_1} \quad f_{i_1} \quad f_{i_2} \quad f_{i_3} \quad t_{i_{m_i}} \quad T_{\text{max}} \]

Hein Meling, CANOE Workshop, Toronto, August 2010
Estimators

- In real systems, failure intensity $\lambda$ very low;
  - i.e, $\lambda^{-1} >> T_{max}$
  - $\pi_k = \text{probability of a trajectory reaching stratum } S_k$

$$\pi_k = \sum_{i \in S_k} P_i$$

- Unconditional probability of a sample in
  - Stratum $S_2$
    $$\pi_2 = (n - 1) \lambda \Theta_1 \pi_1$$
  - Stratum $S_3$
    - (in the paper)
Experimental Results

- Perform fault injections on target system according to sampling scheme
- 3000 (lab) experiments performed
  - Aiming for 1000 in each stratum
  - Classified as stratum $S_k$ if exactly $k$ failures occur before completion of experiment
Table 1. Results obtained from the experiments (in milliseconds).

| Classification | Count | \( \Theta_k = E(T|S_k) \) | sd = \( \sqrt{\sigma_k} \) | \( \Theta_k \), 95% conf.int. |
|----------------|-------|----------------|-----------------|-------------------------------|
| Stratum \( S_1 \) | 1781  | 8461.77        | 185.64          | (8328.98, 8594.56)           |
| Stratum \( S_2 \) | 793   | 12783.91       | 1002.22         | (12067.01, 13500.80)         |
| Stratum \( S_3 \) | 407   | 17396.55       | 924.90          | (16734.96, 18058.13)         |

| Classification | Count | \( \theta_k = E(T|S_k) \) | sd = \( \sqrt{\sigma_k} \) | \( \theta_k \), 95% conf.int. | Highest | Lowest |
|----------------|-------|----------------|----------------|-------------------------------|---------|--------|
| \( Strata_1 \) | 2265  | 2569.22       | 478.23         | (1631.89, 3506.55)            | 16659   | 1742   |
| \( Strata_2 \) | 591   | 4158.83       | 1039.10        | (2122.18, 6195.47)            | 12869   | 2496   |
| \( Strata_3 \) | 110   | 5966.58       | 1550.90        | (2926.82, 9006.35)            | 16086   | 3046   |
Experimental Results

- 19 experiments (0.63%) were classified as inadequate
  - 16 experiments failed to recover
  - 3 experiments experienced additional not-intended failures
  - Of the 16, two were for S1, 6 for S2 and 11 for S3
  - These 16 are due to deficiencies in Jgroup/ARM

- These inadequate runs are accounted for as trajectories visiting a down state for 5 minutes (typically a reboot)

- For DARM there were 2 inadequate experiments
Prob. Density Function

Density estimate of Jgroup/ARM crash recovery times

- Single node failure (N=1781, BW=126.9)
- Two nearly coincident node failures (N=793, BW=652.9)
- Three nearly coincident node failures (N=407, BW=637.2)

Time since injection of first crash failure (ms)

Max. 0.00116
Compared to JGroupwARM, DARM generally achieves a full recovery almost 5 seconds faster than its predecessor. Figure 5.2 presents the probability density graph for Strata 2. We see that the expectancy is more spread out compared to Figure 5.1 for Strata 1. This is expected as the different trajectories for a Strata 2 is more variable considering that the two fault injections occur at different time intervals. The highest values of Strata 2 recoveries are 439 and 286 seconds. They have both been manually checked against the logs in suspicion of the same cause for increased recovery times as observed for Strata 1. However, the explanations for these values seem to be that the interval of failures have been close to the maximum of what DARM "allows" without recovery taking place in between. The mean value of a Strata 2 recovery is 458 seconds. The highest value observed for a Strata 2 recovery in DARM is only 26 seconds above the mean value of Strata 2 recovery in JGroupwARM. Again, the performance of DARM is proven better than that of the JGroupwARM framework, the variance however is somewhat the same for both frameworks.

Figure 5.3 presents the probability density graph for Strata 3. The graph is somewhat misleading since its lower bound covers areas below its lowest observed value. Notice...
# Applying the Equations

Table 2. Computed probabilities, unavailability metric and the system MTBF.

<table>
<thead>
<tr>
<th>Experiment Recovery Period</th>
<th>Processor Recovery (5 min.)</th>
<th>Manual Processor Recovery (2 hrs.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Processor Mean Time Between Failure (MTBF=λ⁻¹) (in days)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>200</td>
</tr>
<tr>
<td>π₁</td>
<td>0.999999314</td>
<td>0.99999657</td>
</tr>
<tr>
<td>π₂</td>
<td>6.855602 · 10⁻⁶</td>
<td>3.427801 · 10⁻⁶</td>
</tr>
<tr>
<td>π₃</td>
<td>4.072921 · 10⁻¹¹</td>
<td>1.018230 · 10⁻¹¹</td>
</tr>
<tr>
<td>Λ⁻¹</td>
<td>20.3367 yrs</td>
<td>40.6741 yrs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Experiment Recovery Period</th>
<th>Processor Recovery (5 min.)</th>
<th>Manual Processor Recovery (2 hrs.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Processor Mean Time Between Failure (pmtbf=λ⁻¹) (in days)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>200</td>
</tr>
<tr>
<td>π₁</td>
<td>0.9999979184</td>
<td>0.9999989592</td>
</tr>
<tr>
<td>π₂</td>
<td>2.0815438 · 10⁻⁶</td>
<td>1.0407719 · 10⁻⁶</td>
</tr>
<tr>
<td>π₃</td>
<td>4.0903937 · 10⁻¹²</td>
<td>1.0225984 · 10⁻¹²</td>
</tr>
<tr>
<td>Λ⁻¹</td>
<td>212 yrs</td>
<td>851 yrs</td>
</tr>
</tbody>
</table>
Concluding Remarks

- DARM supports autonomous fault treatment
  - Recovery decisions are distributed to the individual groups
  - In previous systems recovery decisions were centralized
    - Complex and error-prone
- DARM has been released as open source at:
  - darm.ux.uis.no
- We are performing more advanced measurements
  - Client perceived availability
  - Longer executions and with other parameters to get statistically significant results
- Experimental results indicate that self-repairing systems can obtain very high availability and MTBF
- Automated fault injection tool
  - Proved very useful for uncovering a number of subtle bugs
  - Allows for systematic stress and regression testing
Open Issues

- Handling full group failures
  - ARM have a centralized component to monitor all groups
  - DARM only monitors the group from within itself
  - Could let the factory handle this in some way
    - Lease/Renew or simple pinging

- Management tasks to simplify deployment of applications
  - Self-configuration
  - Reconfiguration of nodes that can host replicas

- Express policies in terms of equations

- Implement more policies
Group Failure Handling

N1 crashed

N2 crashed

N1:
notify(IamAlive)

N2:
notify(IamAlive)

N3:
join

N4:
createReplica()

RM:
notify(ViewChange)

join

notify(ViewChange)

notify(IamAlive)

timeout
Thanks!
References


