Achieving Robust Self-Management for Large-Scale Distributed Applications

Ahmad Al-Shishtawy[1,2], Muhammad Asif Fayyaz[1], Konstantin Popov[2], and Vladimir Vlassov[1]

1 - Royal Institute of Technology (**KTH**), Stockholm, Sweden
`{ahmadas, mafayyaz, vladv}@kth.se`
2 - Swedish Institute of Computer Science (**SICS**), Stockholm, Sweden
`{ahmad, kost}@sics.se`

SASO 2010
Budapest, September 29, 2010

Outline

1. Introduction
2. Automatic Reconfiguration
3. Evaluation
4. Conclusions and Future Work

Outline

1 Introduction

2 Automatic Reconfiguration

3 Evaluation

4 Conclusions and Future Work

Outline

1. Introduction
2. Automatic Reconfiguration
3. Evaluation
4. Conclusions and Future Work

Outline

1. Introduction
2. Automatic Reconfiguration
3. Evaluation
4. Conclusions and Future Work

Outline

1. Introduction
2. Automatic Reconfiguration
3. Evaluation
4. Conclusions and Future Work

Motivation

- Achieving self-management can be challenging
- Becomes more challenging in dynamic environments with resource churn (Join/Leave/Fail)
- Dealing with the effect of churn on management increases the complexity of the management logic

We propose

Robust Management Element (RME) abstraction that are able to heal themselves under continuous churn

Motivation

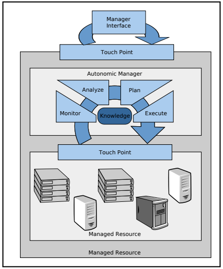
- Achieving self-management can be challenging
- Becomes more challenging in dynamic environments with resource churn (Join/Leave/Fail)
- Dealing with the effect of churn on management increases the complexity of the management logic

We propose

Robust Management Element (RME) abstraction that are able to heal themselves under continuous churn

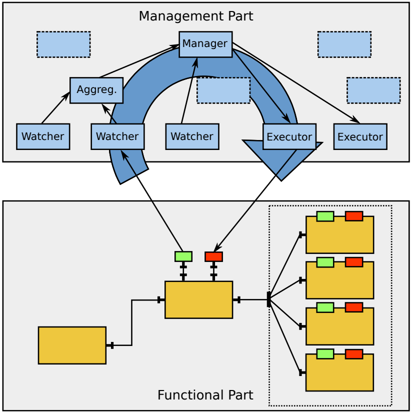
Use Case: The Niche Platform

- Niche is a Distributed Component Management System
- Niche implements the Autonomic Computing Architecture
- Niche targets large-scale and dynamic distributed environment and applications
  - Resources and components are distributed
  - Autonomic managers are distributed
  - Sensors and Actuators are distributed



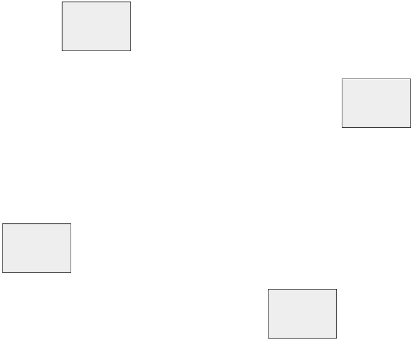
Niche Management Part

Autonomic Managers (control loops) built as network of management elements (MEs)



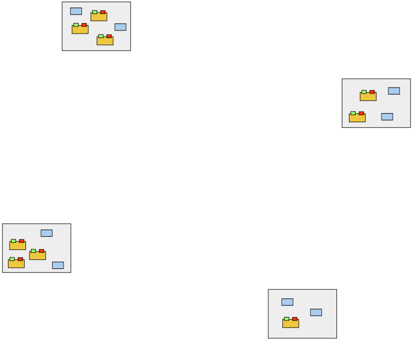
Niche Runtime Environment

- Containers that host components and MEs
- Use a Structured Overlay Network (SON) for communication



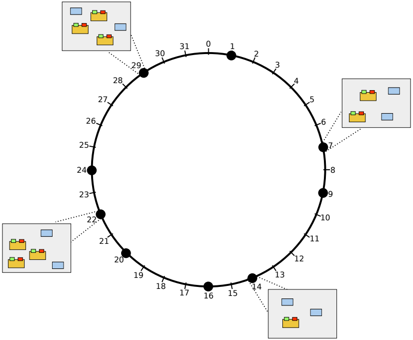
Niche Runtime Environment

- Containers that host components and MEs
- Use a Structured Overlay Network (SON) for communication



Niche Runtime Environment

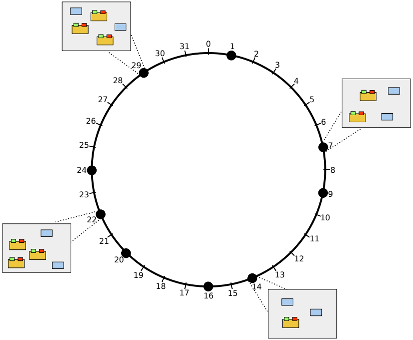
- Containers that host components and MEs
- Use a Structured Overlay Network (SON) for communication



Dealing With Resource Churn

How to deal with failures?

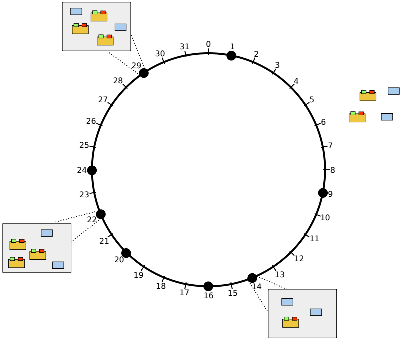
- MEs heal the functional part
- How to heal failed MEs?
    - Programmatically in the management logic
    - Transparently by the platform



Dealing With Resource Churn

How to deal with failures?

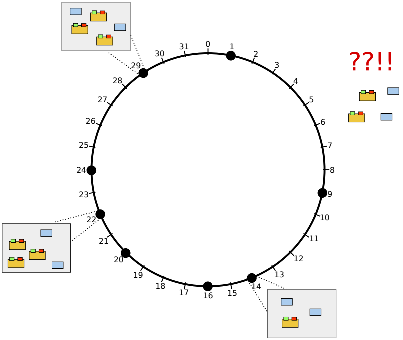
- MEs heal the functional part
- How to heal failed MEs?
  - Programmatically in the management logic
  - Transparently by the platform



Dealing With Resource Churn

How to deal with failures?

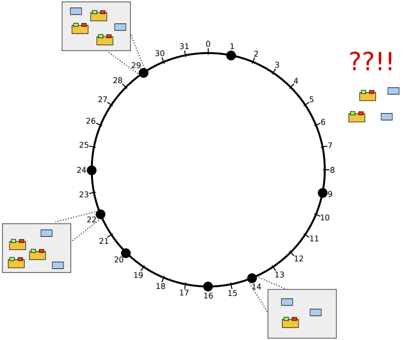
- MEs heal the functional part
- How to heal failed MEs?
  - Programmatically in the management logic
  - Transparently by the platform



Dealing With Resource Churn

How to deal with failures?

- MEs heal the functional part
- How to heal failed MEs?
    - Programmatically in the management logic
    - Transparently by the platform



Robust Management Elements

A Robust Management Element (RME):

- is replicated to ensure fault-tolerance
- tolerates continuous churn by automatically restoring failed replicas on other nodes
- maintains its state consistent among replicas
- provides its service with minimal disruption in spite of resource churn (high availability)
- is location transparent, i.e., RME clients communicate with it regardless of current location of its replicas

Robust Management Elements

A Robust Management Element (RME):

- is replicated to ensure fault-tolerance
- tolerates continuous churn by automatically restoring failed replicas on other nodes
- maintains its state consistent among replicas
- provides its service with minimal disruption in spite of resource churn (high availability)
- is location transparent, i.e., RME clients communicate with it regardless of current location of its replicas

Robust Management Elements

A Robust Management Element (RME):

- is replicated to ensure fault-tolerance
- tolerates continuous churn by automatically restoring failed replicas on other nodes
- maintains its state consistent among replicas
- provides its service with minimal disruption in spite of resource churn (high availability)
- is location transparent, i.e., RME clients communicate with it regardless of current location of its replicas

Robust Management Elements

A Robust Management Element (RME):

- is replicated to ensure fault-tolerance
- tolerates continuous churn by automatically restoring failed replicas on other nodes
- maintains its state consistent among replicas
- provides its service with minimal disruption in spite of resource churn (high availability)
- is location transparent, i.e., RME clients communicate with it regardless of current location of its replicas

Robust Management Elements

A Robust Management Element (RME):

- is replicated to ensure fault-tolerance
- tolerates continuous churn by automatically restoring failed replicas on other nodes
- maintains its state consistent among replicas
- provides its service with minimal disruption in spite of resource churn (high availability)
- is location transparent, i.e., RME clients communicate with it regardless of current location of its replicas

Robust Management Elements

A Robust Management Element (RME):

- is replicated to ensure fault-tolerance
- tolerates continuous churn by automatically restoring failed replicas on other nodes
- maintains its state consistent among replicas
- provides its service with minimal disruption in spite of resource churn (high availability)
- is location transparent, i.e., RME clients communicate with it regardless of current location of its replicas

Solution Outline

- Finite state machine replication
- An algorithm for changing replica set (reconfiguration/migration)
- Our decentralized algorithm to automate reconfiguration
  - Structured Overlay Network (SON) to monitor nodes hosting replicas
  - Replica placement scheme to select/locate nodes that host replicas

End Result

Decentralized algorithms for Robust Management Elements (RMEs) that can be used to build robust management!

Solution Outline

- Finite state machine replication
- An algorithm for changing replica set (reconfiguration/migration)
- Our decentralized algorithm to automate reconfiguration
  - Structured Overlay Network (SON) to monitor nodes hosting replicas
  - Replica placement scheme to select/locate nodes that host replicas

End Result

Decentralized algorithms for Robust Management Elements (RMEs) that can be used to build robust management!

Solution Outline

- Finite state machine replication
- An algorithm for changing replica set (reconfiguration/migration)
- Our decentralized algorithm to automate reconfiguration
  - Structured Overlay Network (SON) to monitor nodes hosting replicas
  - Replica placement scheme to select/locate nodes that host replicas

End Result

Decentralized algorithms for Robust Management Elements (RMEs) that can be used to build robust management!

Solution Outline

- Finite state machine replication
- An algorithm for changing replica set (reconfiguration/migration)
- Our decentralized algorithm to automate reconfiguration
  - Structured Overlay Network (SON) to monitor nodes hosting replicas
  - Replica placement scheme to select/locate nodes that host replicas

End Result

Decentralized algorithms for Robust Management Elements (RMEs) that can be used to build robust management!

Solution Outline

- Finite state machine replication
- An algorithm for changing replica set (reconfiguration/migration)
- Our decentralized algorithm to automate reconfiguration
  - Structured Overlay Network (SON) to monitor nodes hosting replicas
  - Replica placement scheme to select/locate nodes that host replicas

End Result

Decentralized algorithms for Robust Management Elements (RMEs) that can be used to build robust management!

Solution Outline

- Finite state machine replication
- An algorithm for changing replica set (reconfiguration/migration)
- Our decentralized algorithm to automate reconfiguration
  - Structured Overlay Network (SON) to monitor nodes hosting replicas
  - Replica placement scheme to select/locate nodes that host replicas

End Result

Decentralized algorithms for Robust Management Elements (RMEs) that can be used to build robust management!

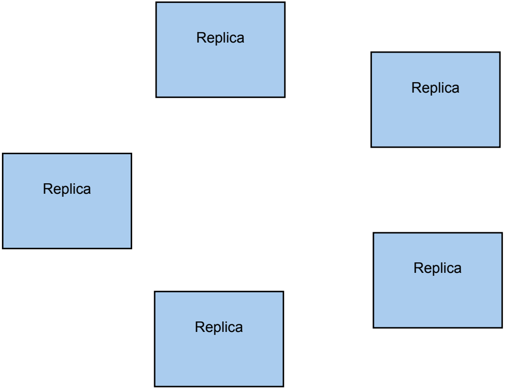
Replicated State Machine (RSM)

Replicated State Machine (RSM)

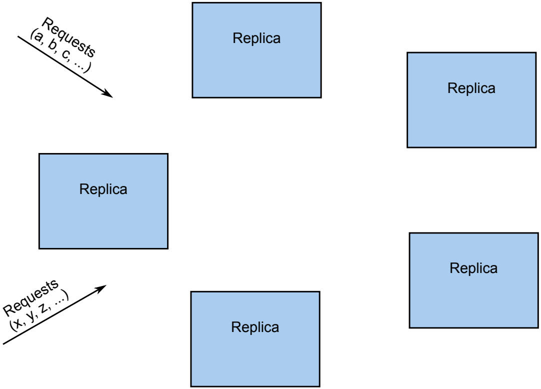
Replicated State Machine (RSM)

Replicated State Machine (RSM)

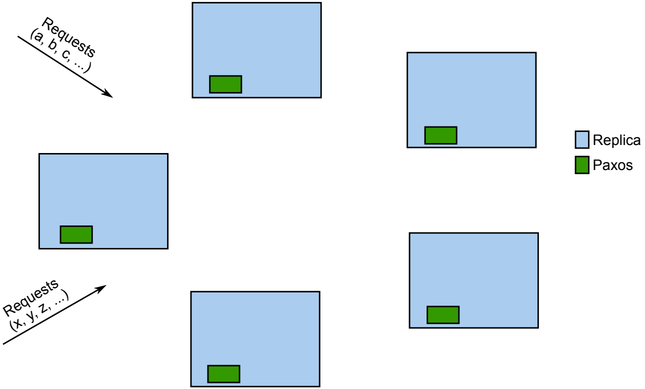
Replicated State Machine (RSM)



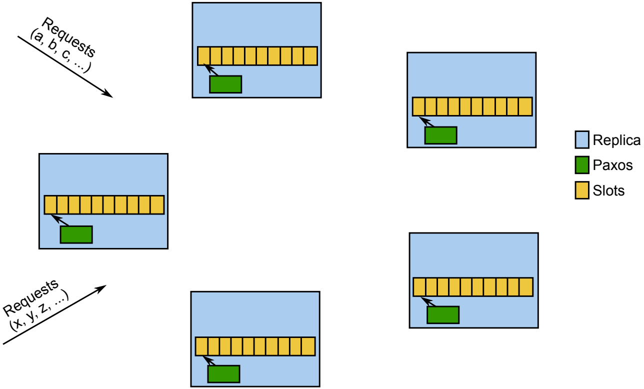
Replicated State Machine (RSM)



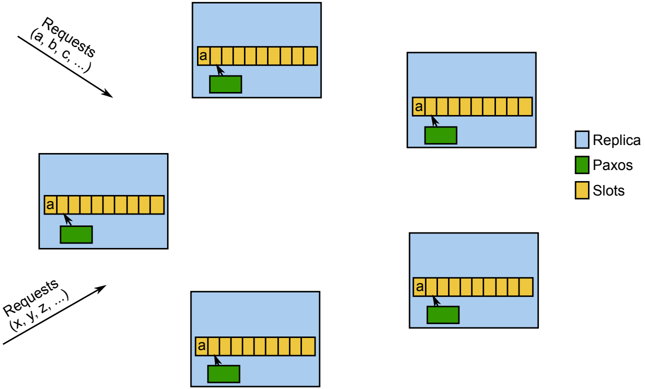
Replicated State Machine (RSM)



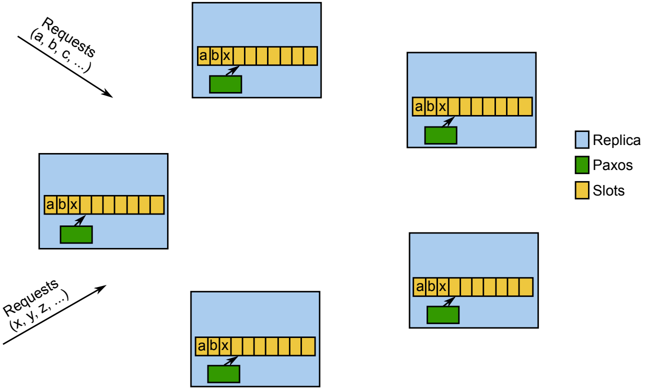
Replicated State Machine (RSM)



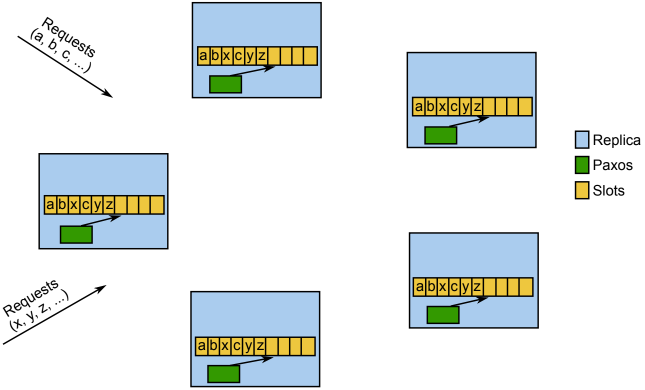
Replicated State Machine (RSM)



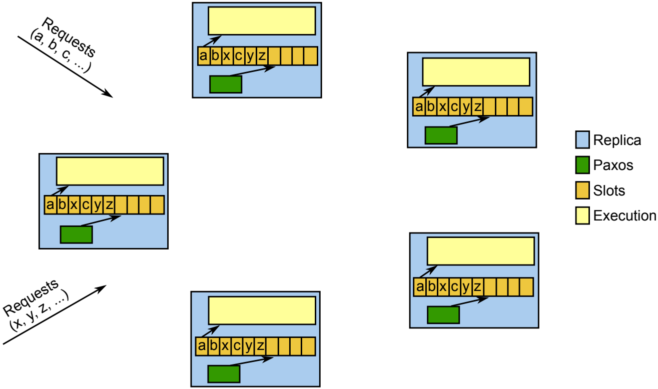
Replicated State Machine (RSM)



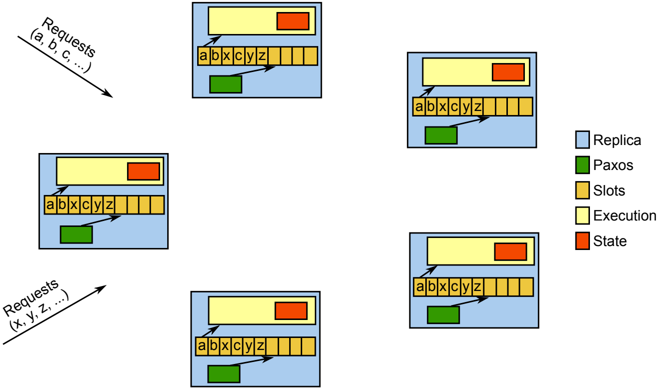
Replicated State Machine (RSM)



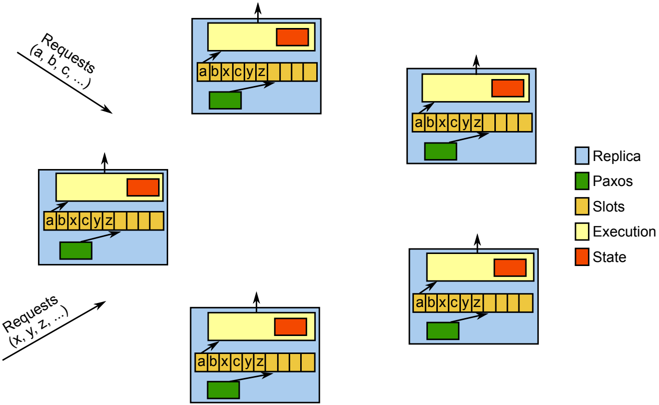
Replicated State Machine (RSM)



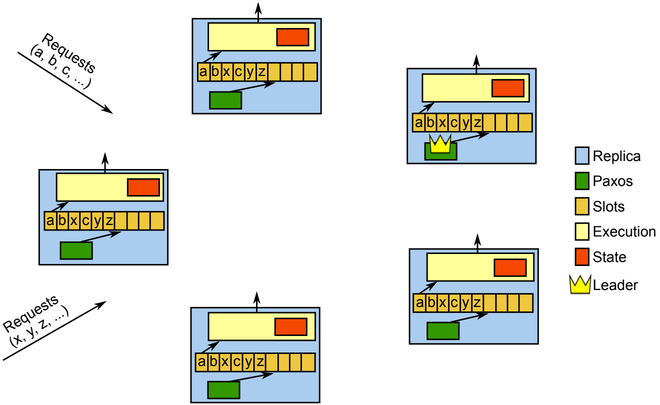
Replicated State Machine (RSM)



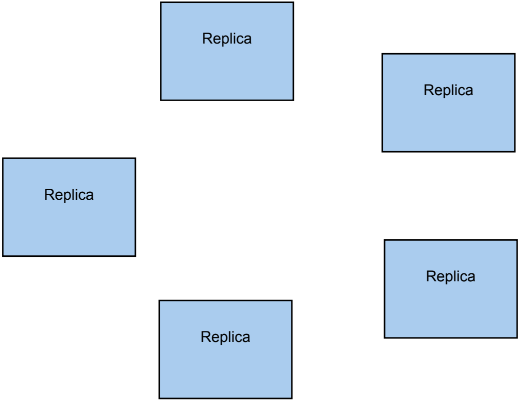
Replicated State Machine (RSM)



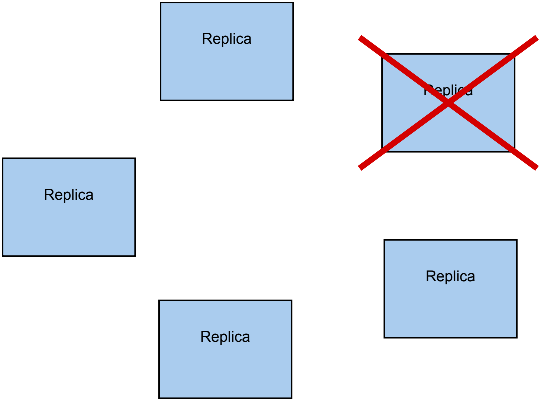
Replicated State Machine (RSM)



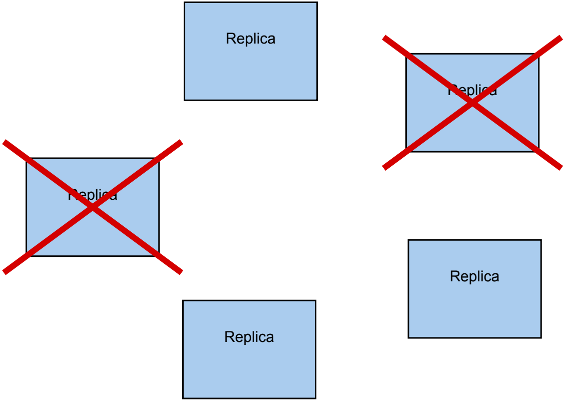
Replicated State Machine is Not Enough



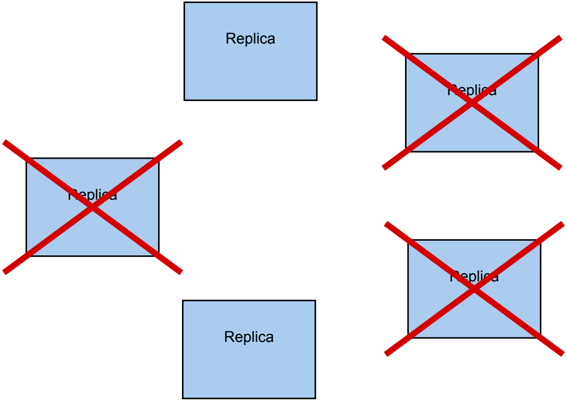
Replicated State Machine is Not Enough



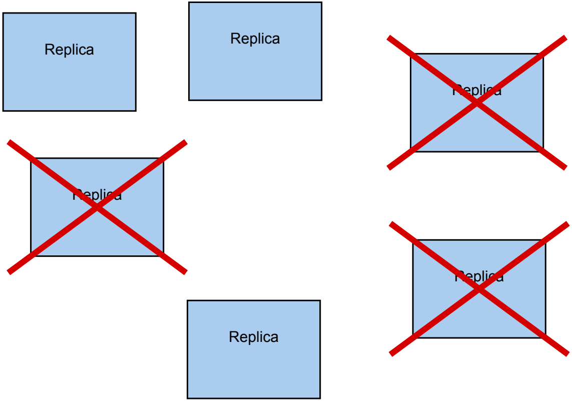
Replicated State Machine is Not Enough



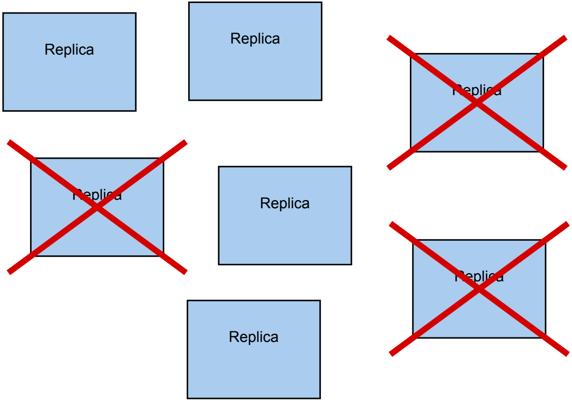
Replicated State Machine is Not Enough



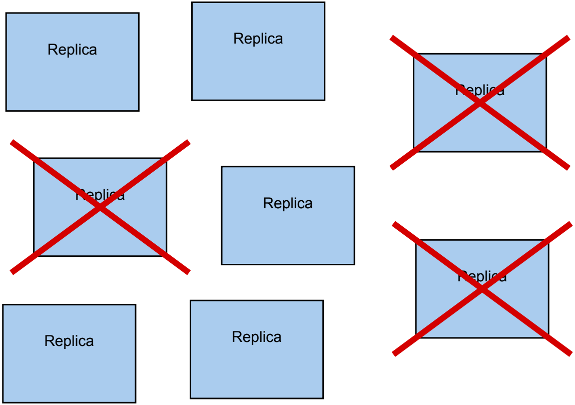
Replicated State Machine is Not Enough



Replicated State Machine is Not Enough



Replicated State Machine is Not Enough



Migration/Reconfiguration: Basic Idea

- A configuration is the set of replicas
- Replicas include the configuration as part of the state
- A special request that changes the configuration
- We used the SMART algorithm

Migration/Reconfiguration: Basic Idea

- A configuration is the set of replicas
- Replicas include the configuration as part of the state
- A special request that changes the configuration
- We used the SMART algorithm

Migration/Reconfiguration: Basic Idea

- A configuration is the set of replicas
- Replicas include the configuration as part of the state
- A special request that changes the configuration
- We used the SMART algorithm

Migration/Reconfiguration: Basic Idea

- A configuration is the set of replicas
- Replicas include the configuration as part of the state
- A special request that changes the configuration
- We used the SMART algorithm

Our Approach to Automate Reconfiguration

Goals

- Automatically maintain configuration in a decentralized way
- Select resources, detect failures, and decide to migrate
- Clients find service without central repository

Our Approach to Automate Reconfiguration

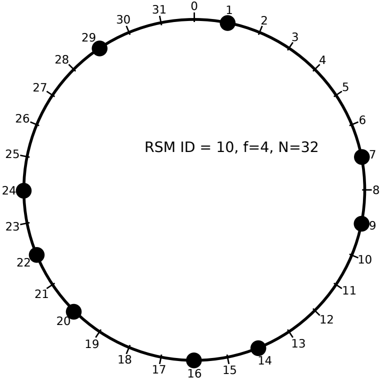
- Structure Overlay Network to monitor nodes hosting replicas
- Replica placement scheme (such as symmetric replication) to select nodes that will host replicas
- RSM receives monitoring information and uses it to construct a new configuration and to decide when to migrate.
- A decentralized algorithm that automates the reconfiguration of the replica set in order to tolerate continuous resource churn.

Outline

1. Introduction
2. Automatic Reconfiguration
3. Evaluation
4. Conclusions and Future Work

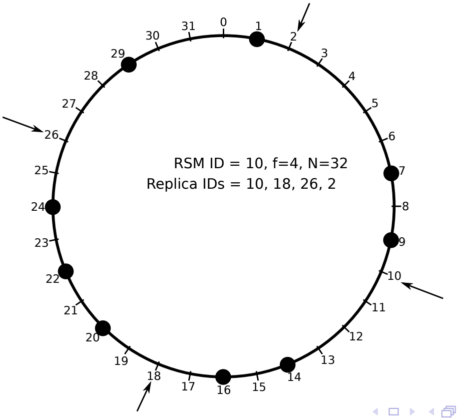
Creating a Replicated State Machine (RSM)

Any node can create a RSM. Select ID and replication degree



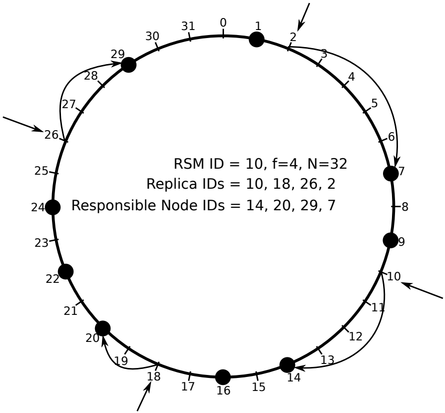
Creating a Replicated State Machine (RSM)

The node uses symmetric replication scheme to calculate replica IDs



Creating a Replicated State Machine (RSM)

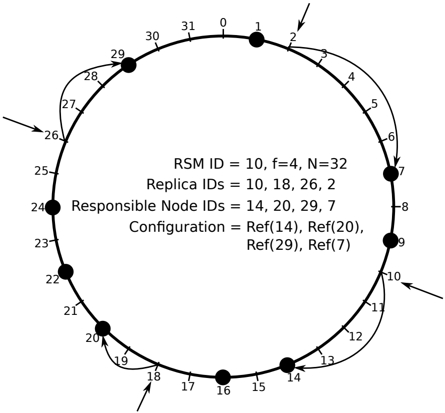
The node uses lookups to find responsible nodes . . .



RSM ID = 10, f=4, N=32
Replica IDs = 10, 18, 26, 2
Responsible Node IDs = 14, 20, 29, 7

Creating a Replicated State Machine (RSM)

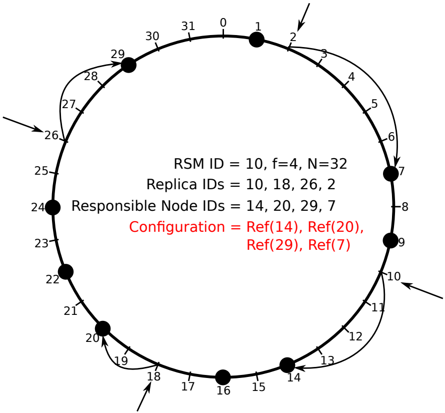
. . . and gets direct references to them



RSM ID = 10, f=4, N=32
Replica IDs = 10, 18, 26, 2
Responsible Node IDs = 14, 20, 29, 7
Configuration = Ref(14), Ref(20), Ref(29), Ref(7)

Creating a Replicated State Machine (RSM)

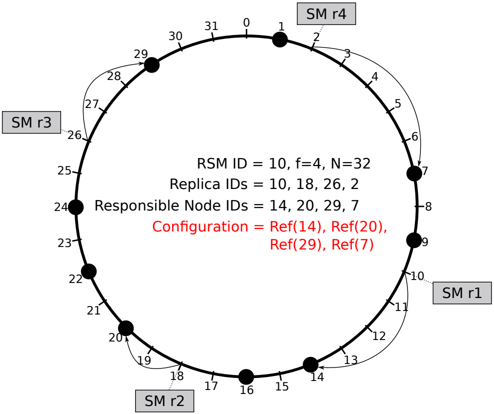
The set of direct references forms the configuration



RSM ID = 10, f=4, N=32
Replica IDs = 10, 18, 26, 2
Responsible Node IDs = 14, 20, 29, 7
Configuration = Ref(14), Ref(20), Ref(29), Ref(7)

Creating a Replicated State Machine (RSM)

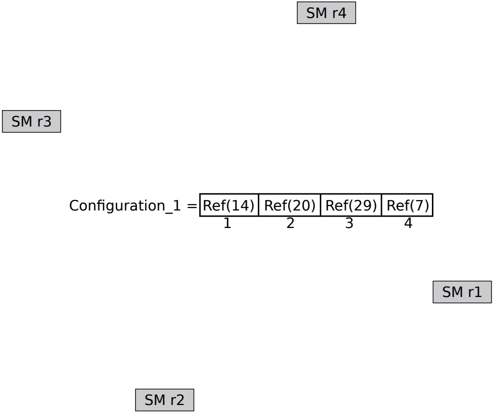
The node sends a *Create* message to the configuration



RSM ID = 10, f=4, N=32
Replica IDs = 10, 18, 26, 2
Responsible Node IDs = 14, 20, 29, 7
Configuration = Ref(14), Ref(20), Ref(29), Ref(7)

Creating a Replicated State Machine (RSM)

Now replicas communicate directly using the configuration



SM r4

SM r3

Configuration_1 =

Ref(14)	Ref(20)	Ref(29)	Ref(7)
1	2	3	4

SM r1

SM r2

Clients Interaction

- We do not need a configuration repository
- A client need to know only the RSM_ID and replication degree
- The client uses symmetric replication scheme and lookups to calculate the configuration
- Due to lookup inconsistency the calculated configuration and the real configuration may be different
    - We assume that they overlap for clients to be able to send requests
    - Otherwise the request will fail and the client will have to retry later

Why to Migrate?

- To fix Lookup inconsistencies
- To handle resource churn

Handling Lookup Inconsistency

- Because of lookup inconsistency the configuration may contain incorrect nodes
- The inconsistency is detected when a node receives a request targeted at a replica that the node does not have but should be responsible for
- In this case the node issues a configuration change request asking the RSM to replace the incorrect node in the current configuration with itself

Handling Churn

- Similar to handling churn in a DHT
  - When a node joins, it gets a list of replicas (RSM_ID and rank) it is responsible for from its successor
  - When a node leaves, it hands over replicas to its successor
  - When a node fails, its successor uses symmetric replication schema and range-cast to find replicas it should be responsible for
- After getting the list of replicas the node issues a configuration request to each RSM to replace incorrect node with itself

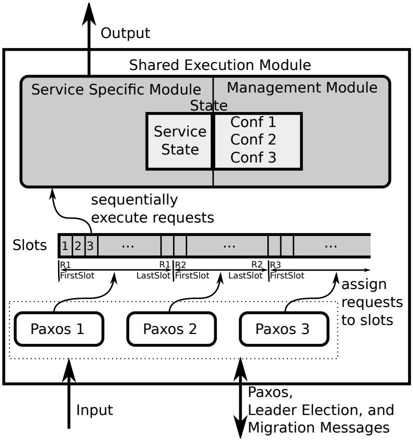
Changing the Configuration (Migration)

- In reconfiguration algorithms the admin sends a configuration change request that contains all nodes in the new configuration
- We can not do the same in a decentralized fashion (to avoid conflicts)

Changing the Configuration (Migration)

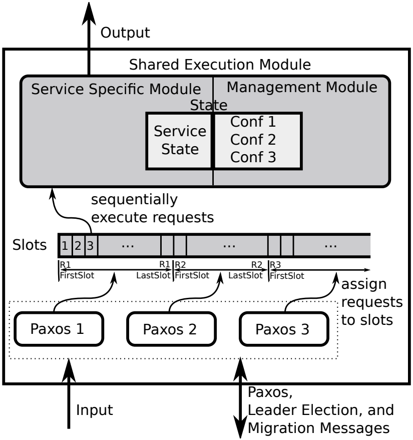
- In our approach the request does not contain the entire configuration.
- It contains only monitoring information (a request to replace a particular node)
- The RSM is extended to use this information to construct a new configuration and to decide when to migrate
- This is done in a deterministic and consistent way (guaranteed by the state machine)

Replica Architecture



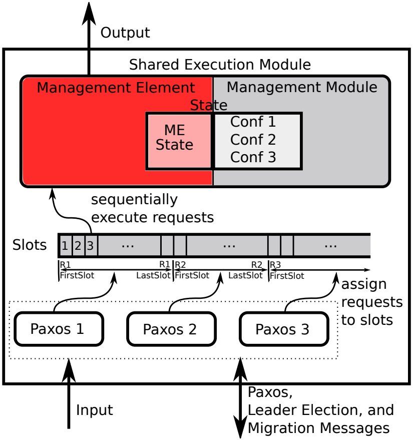
Robust Management Elements

- Our approach is generic and can be useful for many services
- We use it in Niche to implement Robust Management Elements
- Replace the service specific part of the execution module with a management element



Robust Management Elements

- Our approach is generic and can be useful for many services
- We use it in Niche to implement Robust Management Elements
- Replace the service specific part of the execution module with a management element



Outline

1 Introduction

2 Automatic Reconfiguration

3 Evaluation

4 Conclusions and Future Work

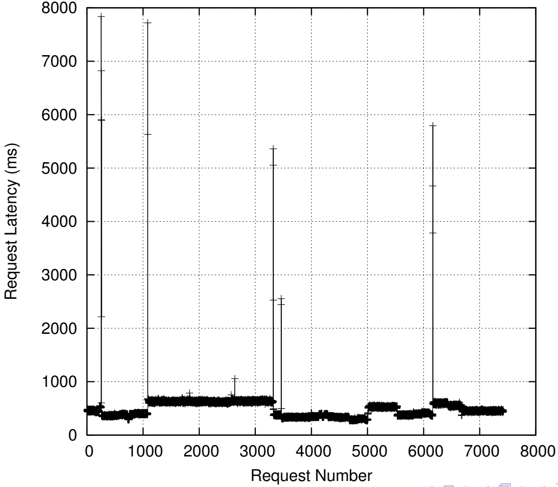
Methodology

- Simulation-based performance evaluation
  - Request latency
  - Number of messages
- Focused on the effect of the churn rate and replication degree on request critical path and failure recovery
- Built a prototype implementation of RME
  - service is implemented as an aggregator that accumulates values from clients
  - A client represents both sensor and actuator
- Used the King latency dataset
  - Measures latencies between DNS servers

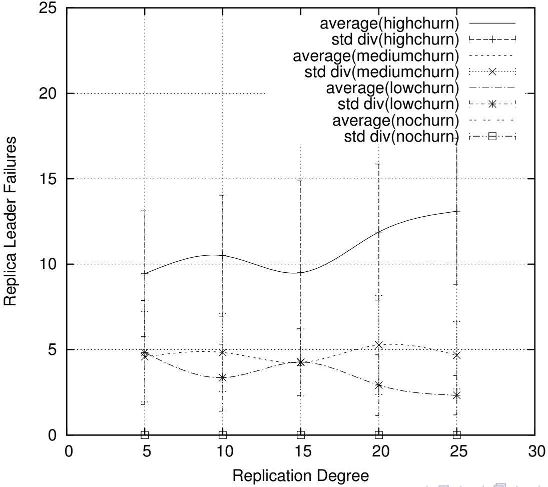
Some Parameters

- Overlay size: in the rage of 200 to 600 nodes
- Replication degree: varies from 5 to 25
- Failure threshold: varies from 1 to strictly less than half of the number of replicas
- Lifetime-based node failure model with Shifted Pareto distribution of node lifetime. We modeled three levels of churn:
    - high churn rate (mean lifetime of 30 minutes)
    - medium churn rate (90 minutes)
    - low churn rate (150 minutes)

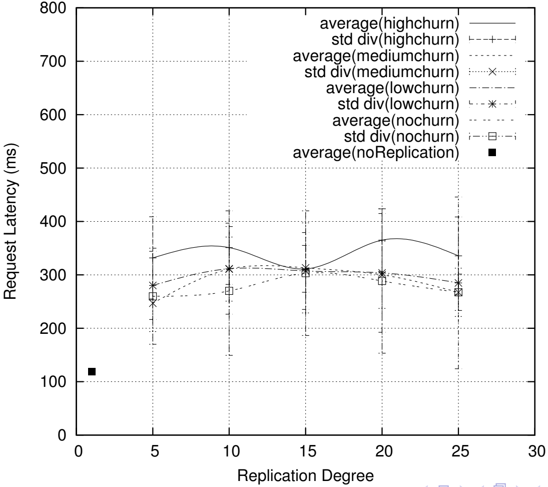
Request latency for a single client



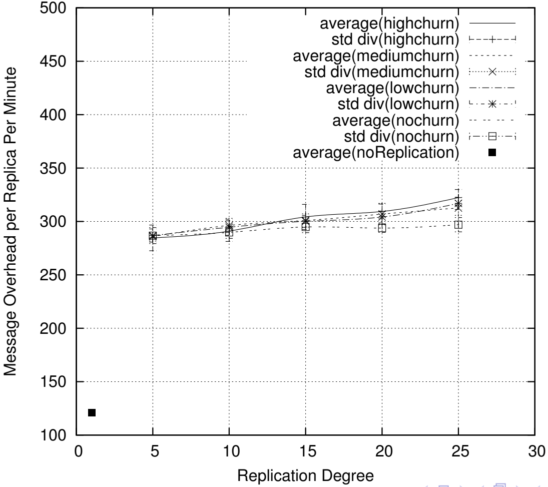
Leader failures vs. replication degree



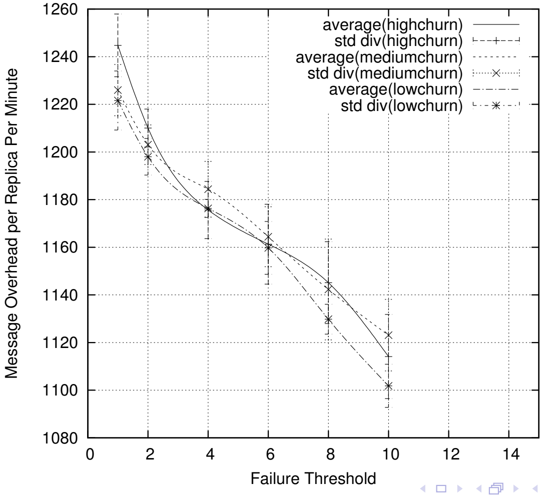
Request latency vs. replication degree



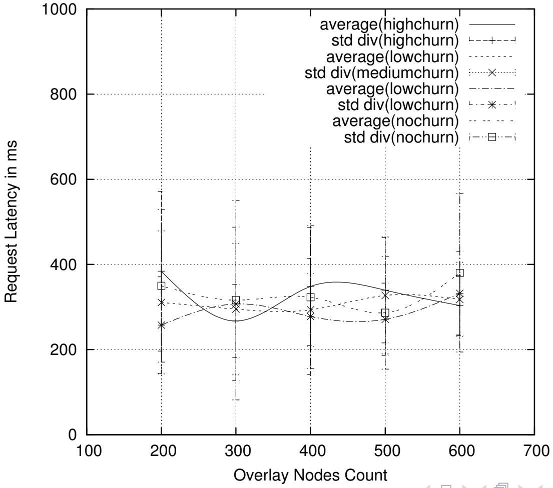
Messages/minute vs. replication degree



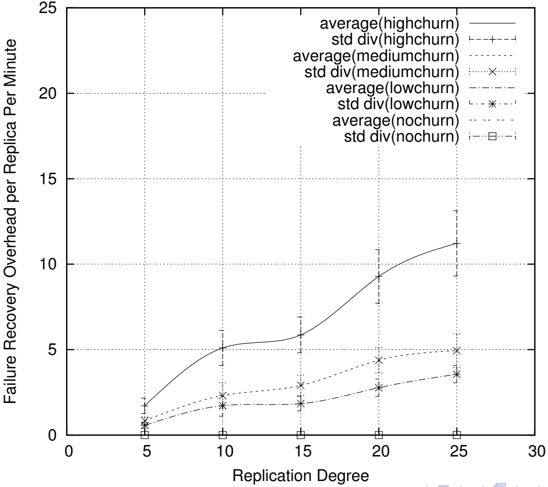
Messages per minute vs. failure threshold



Request latency vs. overlay size



Recovery messages vs. replication degree



Outline

1 Introduction

2 Automatic Reconfiguration

3 Evaluation

4 Conclusions and Future Work

Conclusions

- Proposed the concept of Robust Management Elements (RMEs)
- Separate the issue of robustness of management from the actual management mechanisms
- Achieve RMEs by automatically reconfiguring a replicated state machines

Conclusions

- Used a replica placement scheme to decide on the placement of replicas and uses SON to monitor them
- The replicated state machine is used to process monitoring information and to decide when and where to migrate
- We have developed a prototype and conducted various simulation experiments which have shown the validity and feasibility of our approach

Future Work

- Evaluate our approach on larger scales and extreme values of load and churn rate
- Optimise the algorithms in order to reduce the amount of messages and improve performance
- Implement our approach in the Niche platform to support RMEs in self-managing distributed applications
- Try to apply our approach to other problems in distributed computing

Thank you for careful listening :-)

Questions?