國立臺灣大學電機資訊學院資訊工程學系
碩士論文
Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Taiwan University
Master Thesis

# 智能M2M中介軟體的容錯系統研究
# Research on Fault Tolerant Distributed M2M Systems

蘇適
Penn H. Su

指導教授：許永真 博士
Advisor: Jane Yung-Jen Hsu, Ph.D.

中華民國 102 年 3 月

March, 2013

國立臺灣大學
資訊工程學系

碩士論文

# 智能M2M中介軟體的容錯系統研究

蘇適撰

102
3

# 國立臺灣大學碩士學位論文
# 口試委員會審定書

## 智能M2M中介軟體的容錯系統研究
## Research on Fault Tolerant Distributed M2M Systems

本論文係蘇適君 (R99922157) 在國立臺灣大學資訊工程學研究所完成之碩士學位論文，於民國 102 年 3 月 14 日承下列考試委員審查通過及口試及格，特此證明

口試委員：

_____

_____

_____

_____

_____

所主任 _____

# Acknowledgments

I would like to express my greatest gratitude to the people who have helped and supported me throughout my graduate studies. I am grateful to my advisor Prof. Jane Yung-Jen Hsu for her continuous support, from initial advice, contacts in the early stages of conceptual inception, through ongoing advice and encouragement to this day. I am also grateful to my advisor Prof. Kwei-Jay Lin for his undivided attention to details, ongoing advice and encouragement.

To Dr. Yu-Chung Wang and Prof. Chi-Sheng Shih, for their patience and their kindness for answering my many requests.

To Niels Reijers, my colleague who helped me in completing the project with his vast knowledge, black VM magic tricks, ideas, and thoughts that made this journey a lot smoother.

To my colleagues and my friends, thanks for the support and fun. I want to thank iAgent group, George, Jya-Cheng, Joey, Leeheng, Farmer, Bo-Lung, and Sio, who have directly or indirectly help out this effort.

I wish to thank my parents for their undivided support and interest who inspired me and encouraged me to go my own way to this day, without whom I would be unable to complete my graduate studies.

# 中文摘要

隨著探測器以及執行器的多樣化，分散式系統的應用程式端也隨之變複雜。維護以及修理這類系統就變得非常困難。所以設計一套無需人介入分散式容錯系統變得有必要。

這個論文探討了這個問題。目前有很多其他方式，技巧來解決這個問題不過都是在不同的假設以及不同系統需求導致有不同的保證，沒有一個方法可以有效的解決所有在其他分散式系統相關的問題。在一個多元化、多種類的網路，當每一個節點都不一樣，而且可以執行多種服務以及無狀態的部件應用程式底下，我們提出了一個新的作法來解決這個問題。這個作法不需要任何訊息排序的認同，以及只需要線性的訊息量複雜度 $O(n)$

# Abstract

As the variety of sensors and actuators increase, applications for distributed systems are getting more complex. Repairment becomes difficult to perform manually. It is appealing to design a system that could achieve fault tolerance without require human intervention.

This thesis investigated this problem. There are many different techniques exist to solve this problem under different assumptions with varying guarantees, what works best for one instance of a problem does not always apply to the others. This thesis proposed a new technique to solve the problem under the assumption of stateless application and heterogeneous network with a varity of hardware specifications where some nodes could perform multiple services at a time. The technique we proposed does not require any strong message ordering properties, and it requires only linear message complexity

# Contents

# List of Figures

# Chapter 1

# Introduction

This chapter provides an overview of the thesis. First, we describe why WSN deployment and maintenance are still difficult and then we introduce the reconfigurable fault tolerance system problem and approaches to solving this problem. Next, we describe some related work on a reconfigurable fault tolerant distributed system. Lastly, we present our proposed solution to this problem.

## 1.1 Wireless Sensor Network Deployment is Hard

Wireless sensor networks are areas filled with network of tiny, resource limited sensors communicating wirelessly. Each sensor is capable of sensing the enviornment in its proximity. Wireless sensor networks are typically employed in a variety of applications ranging from home automation to millitary.

Sensor networks offer the ability to monitor real-world phenomena in detail and at large scale by embedding devices into the environment. Deployment is about setting up an sensor network in a real-world environment. Deployment is a labor-intensize and cumbersome task since environmental influences or loose program logic in code might trigger bugs or sensor failures that degrade performance in any way that has not been

observed during pre-deployment testing in the lab.

The real world has strong influences of the function of a sensor network that could change the quality of wireless communication links, and by putting extreme physical strains on sensor nodes. Laboratory testbed or simulator can only model to a very limited extent of those influences.

There have been several reports on sensor network installations where they encountered problems during their deployment[3][10][2][14][9][12][15][16].

Testbed in laboratory environment can still not model the full extents of the influences a real world enviroment could do. Deployment still a big problem in wireless sensor network applications.

## 1.2 Redundancy architecture

A distributed system usually consists of hosts that host services that clients or other services could read or write with some associated communication frequencies according to the application requirements. The problem of partial failures makes service redundancy a fundamental technique to distributed systems as it improves availability, eliminiates single points of failure. A system that is hardwired to get data from node X will fail when X fails. The problem of designing a system with replicas where node Y, which can provide the same service as X, can take over when X fails. To design such system, it is important to have a clear definition of a service such that it could be replicated onto heterogeneous hosts. It is also essential that the system could track and manage available replicas in the network. Autonomy is an important attribute of distributed systems since most of them would be left unattended for a long period of time; systems should be able to reconfigure and recover themselves in the event of failure.

## 1.3 Problem Definition

The problem of designing a distributed system that could handle failures and increase availability for all services is different depending on the system and application requirements. However, some generalities can be established. There will usually be a set of components which will be assigned to a set of hosts where components are associated with some communication frequencies and are reading/writing data from other components. Hosts could fail, and there is a time constraints on the recovery process. The objective is to detect and handle failures with mimimum communication and memory overhead.

In this thesis, the problem of reconfigurable fault tolerant system based on WuKong component based architecture is considered. The problem is interesting and worth solving by itself as this problem existed in all kinds of distributed systems.

### 1.3.1 WuKong: Intelligent Middleware for Flexible Sensor Configurations in M2M Systems

WuKong: Intelligent Middleware for Flexible Sensor Configurations in M2M Systems [11] consists of frameworks that supports flexible configurations of application specifications from flow-based programming. It uses component based architecture where services could be represented by software components which will be deployed to a set of hosts, and each host could hold more than one component.

### 1.3.2 Challenges

Distributed systems have some unique properties that makes this problem really hard is that communication between nodes are not reliable and the ordering of the messages

received could be out of order or dropped. Therefore most existing work treat the problem as a consensus problem where, mostly a variant of the solution proposed by Leslie Lamport in his paper [4] where acceptors need to come to a consensus for the value of a particular variable made by the proposers, and the learners will have to learn of that decision. Under the assumption of a finite state machine for every process, the ordering properties is really important among the acceptors so none of them could reach a different state thus deviating from consensus. However, we will show that this is unnecessary in our approach.

It is a challenege designing a solution that is scalable and also pertaining to the limitations when typical distributed systems such as wireless sensor networks have tight resource constraints and usually deploy in large quantity which dooms the thought of storing or maintaining any additional states, resouces has to be used economically.

## 1.4 Approach

This thesis proposed an original approach that hasn't been done before. Previous work on the problems has been considered the use of consensus protocols with some sort of configurations that contain a set of members, and for every failure the configuration will be reconfigured using the said consensus protocol to reach consensus in the system. However, the results haven't shown to work under heterogeneous network with nodes that could carry multiple components. This thesis proposed a novel algorithm with the use of a distributed data structure to maintain the list of members in order that provides a way to track redundancies and recovery from failures when nodes could potentially be both a backup and a service provider, and it also eliminates the needs for messages ordering to reach consensus.

### 1.4.1 Related Work

The problem of a distributed fault tolerant system has been addressed in many literature [8, 4, 6, 5, 13, 7], however none of the results considered the case of nodes that could carry multiple components and applications with complicated structure with sensors, computation, and actuators. In contrast, most of their assumptions are based on homogeneous network with nodes with reasonable large memory and computation constraints and services with states.

## 1.5 Thesis Organization

Our work overlaps many diverse but interconnected domains, each topic being itself a subject of advanced research and abundant literature. Chapter 1 gives an introduction to the problem and outline of the approach used to solve the problem. Chapter 2 gives a brief background overview of the topic that this work based on. We start by describing wireless sensor networks. Then we go on to discuss fault tolerant design for distributed systems, it's objectives and recent developments. Then we proceed to talk about component model based middleware and WuKong. Chapter 3 described our work on a reconfigurable component based fault tolerance system. In this chapter we give detail description of our method and algorithms.Chapter 4 discussed the tradeoffs and one possible direction in future research. Finally, chapter 5 presented some conclusions of the work, list of contributions and future work.

# Chapter 2

# Background

## 2.1 Wireless Sensor Networks

Sensor nodes are equipped with low-power, low-cost, and failure-prone sensors or actuators. Sensor networks are networks of sensor nodes that connect to the physical space that are intrumented to produce data that could be meaningful for further research. They collaborate to collect, process and disseminate environmental information[1].

Sensor network could be homogeneous, meaning all nodes are identical with same sensors, actuators and hardware setup. Sensor networks could also be heterogeneous where nodes have different sensors, actuators and hardware setup. Heterogeneous networks require higher level management and organization resources. Wireless sensor networks are nodes that communicate through air by sending electronic signals. Wireless communications aren't stable, as it is highly influenced by environmental factors.

### 2.1.1 Redundancy

Sensor networks are usually deployed in large scale and unattended in long period of time. Sensor networks communicate with low-power wireless radios to aid scientists in collecting spatial data that could lead to more understanding of the environment. However,

several challenges such as node failures, message loss, and sensor calibration leaves the effectiveness of sensor networks in question. With the assumption of spare homogeneous resources, redundancy is used in sensor networks to increase fault tolerance against node failures. The system is designed with backup nodes that could automatically recover and replace should one node fail.

## 2.2 Component Based Middleware

Middleware enables communication and management of data that simplfies complex distributed applications.

As most applications for wireless sensor network involves management of data and communication between network of nodes, middleware is integral in providing a unified experience for implementing more complex architecture such as service-oriented architecture.

However, the separation of design abstractions between low-level hardware and high-level application logic has not been successful in sensor based systems.

It is also not successful in terms of making them adapatable and evolvable for new services in new environments.

Next section presents an instance of a component based middleware.

## 2.3 WuKong: The intelligent middleware for M2M applications

### 2.3.1 Goal

Deployment and development for M2M applications are in its infancy today. As many applications are still single purpose in homogeneous networks with specific network

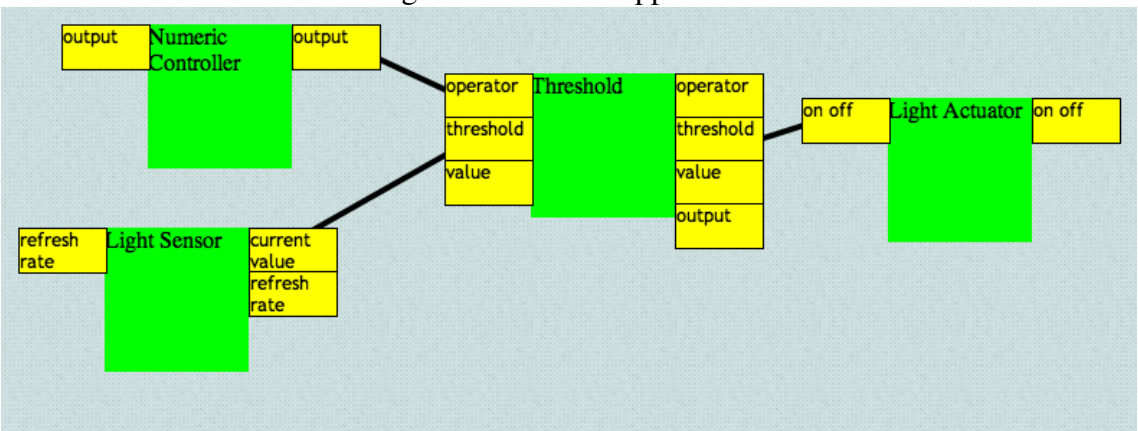
protocols. The hardware has a fixed range of sensors, and the applications cannot be easily ported to other platforms.

The existing middleware support that decouple high-level application design abstractions and low-level hardware has not been successful.

In Intel-NTU Center Special Interests Group for Context Analysis and Management (SIGCAM), we have been collaborating on a project, called WuKong, aiming to develop an intelligent middleware for developing, and deploying machine-to-mahicne (M2M) applications with ease. The main contribution of this project is to support inlligent mapping from a high-level flow based program (FBP) to self-identified, context-specific sensors in a target environment[11].

### 2.3.2 Flow Based Programming

Figure 2.1: A FBP application




M2M applications are by definition distributed where the application requirements involve a network of nodes collaborating for some common goals. M2M applications are typically defined by its flow of information between components, as opposed to more

traditional applications that focus more on local information processing.

Flow Base Programming is best suited for describing M2M applications as it allows the developers for the applications to focus more on the abstraction meaning of the components instead letting the unimportant details such as the hardware to stick right in the face. The result application will contain all necessary information for the framework to construct low-level details to implement the flow.

Applications are designed and constructed on FBP canvas by dragging a set of abstract components from the library as illustrated in Figure 2.1 Each component is illustrated by a green block, each block has a set of properties, each with different access modes, such as readonly, writeonly, readwrite. Properties on the left of the greenblocks are properties that could be written, and properties on the right are readable. Components are connected by links, which is drawn by linking two properties in different components.

Some components represent physical hardware such as a sensor, or an actuator while some other components could represent virtual processes such as mathmatical computations, comparisons, etc. However, the final physical implementations of the components are only made during application deployment by the Master but not during FBP construction.

Components expose their interface through properties. A link is only made with properties with matchinkg data type. The FBP applciation in Figure 2.1 illustrates a simple scenario where the light actuator will turn on the light if light level drops below some value. The Numeric Controller component will be assigned to a user input device used by users to set its desire light threshold, which its output is sent to Threshold component. The light value is sensed from Light Sensor component and sent to Threshold. If the light value sensed is below the threshold value, Threshold will output a boolean to set the on off property of Light Actuator to turn the device, which will be deteremined during deployment, that it is represented by on or off.

### 2.3.3 Sensor Profile Framework

While FBP defines the logical view of an application, WuKong profile framework allows tracking, identification of physical resources within the Sensor Network. There are a range of sensors which provide similar functionality with different level of quality, it could model the sensor capability to enable handling heterogeneous sensors and provide a common abstraction for the logical view.

There are two main concepts in Sensor Profile Framework, WuClasses and WuObjects. WuClasss model components by exposing a number of properties describing, and allow access to, a specific resource represented by the class. Drawing from the example in Figure 2.1, the on off property of Light Actuator component is boolean writeonly. WuClass also implements an update function to describe a component's behavior. For example, Threshold has four properties: operator, threshold, value, output. The output value is determined from the previous 3 properties that it returns true when the value is lower or higher than the threshold which depends on the value of the operator, and it returns false otherwise.

WuObjects are the main unit of processing that are hosted on the nodes. Each WuObject is an instance of WuClasses. It allows the framework to achieve 4 responsibilities:

1. Allow the Master to discover the current status of a node with the list of WuClasses and WuObjects it has.

2. Create new WuObject instances on a node to start receiving data and doing local data processing.

3. Trigger executions in WuObjects, either periodically or as a result of changing inputs.

4. Propagate changes of properties between linked properties in different components, which may be hosted locally or remotely.

**Property Propagation**

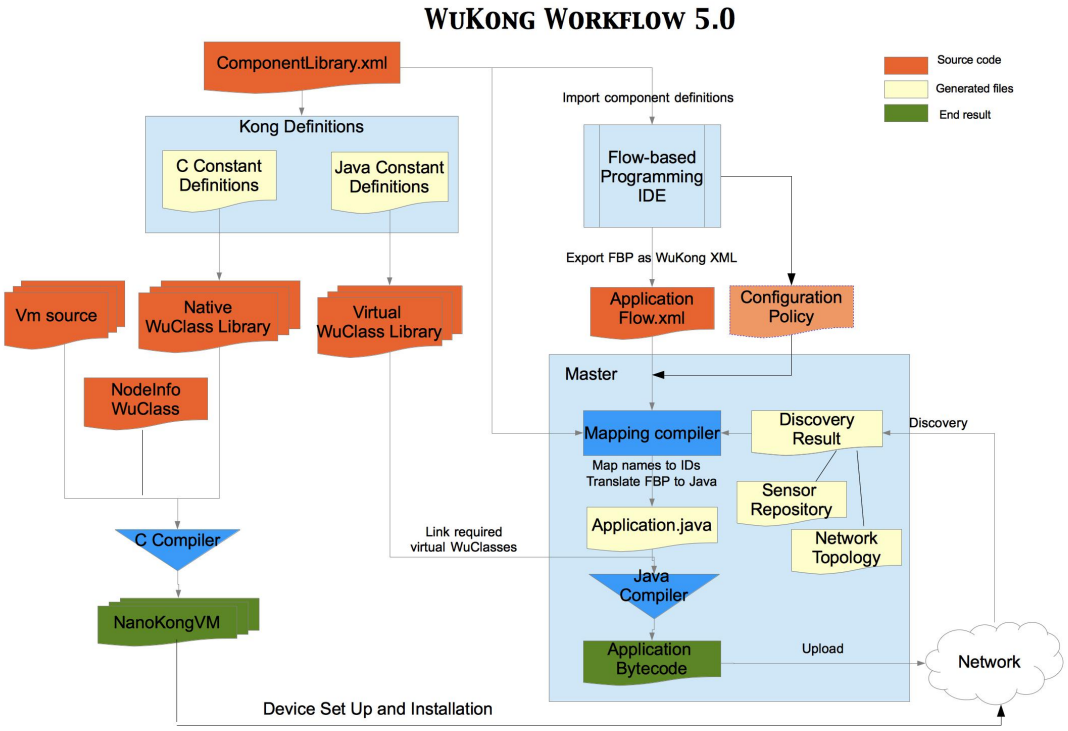
The profile framework is in charge of communication between WuObjects as well, which are not necessarily on the same nodes. Profile Framework monitors the changes in properties and propagate the changes to the connected WuObjects. For example, if a Temperature WuObject is connected to a Threshold WuObject, the changes in Temperature current value property will trigger propagation from the Profile Framework to propagate the new value in current value to the Threshold WuObject connected property, and since Threshold WuObject could be on a different node, the framework will take care of this by initiating a wireless connection between the nodes to send the data over. Once a new value has been set, Threshold WuObject will also trigger its update() function to recompute its output properties which in turn would cause another chain of propagation to the linked WuObjects.

### 2.3.4 Compilation and Mapping

Figure 2.2 shows the overview of WuKong's build flow. The left part represents the build process for NanoKong VM which will be installed on the sensor nodes as part of the WuKong framework. The top part represents the build process for generating component libraries and Virtual WuClass library which will be used in other parts of the process. The right part illustrates the build process for FBP applications from being drawn in the IDE to Java bytecode that will be transmitted to the nodes.

The FBP program from the IDE will be exported as XML to the Master, the Master

Figure 2.2: WuKong application build flow




will then take this XML and passed to Mapper to generate a Java program that will be executed on the nodes. Finally, the compiled code is then wirelessly uploaded to the nodes in the network.

The Java code consists of many parts from different phases of the mapping process. First, the Java code contains information about links between components that were taken from the FBP XML passed in earlier from the IDE. A link contains the source component id, destination component id. The library code for components corresponding to the component ids are stored in the node if it is written in native language, or uploaded as part of the Java bytecode if it is written in Java language. Second, it contains information about the mapping from application component ids to actual node identifications and

positions. The purpose of a mapping which separates from the actual link makes it easier to substitude the actual host of the WuObject later during reconfiguration from the Master. This mapping is created by the Master during discovery phase that probe the network for node's capabilities in terms of available WuClasses, then mapper will decide the final candidates that will be hosting for a component. If no native version of a component is found on the nodes, mapper will substitute with a Java version of it.

### 2.3.5 System Progression Framework

There are a few popular wireless communication protocols in M2M applications: ZigBee, ZWave. It is expected that in the future more diverse wireless nodes equipped with radios that support protocols such as low-power bluebooth and WiFi that all have one or more powerful gateway to connect to the outside world. In WuKong system, one of the gateways will take on the role of higher management decision maker called *Master* to making the decisions for deployment and producing the configuration of wireless sensor networks.

# Chapter 3

# Reconfigurable Atomic Service for Component Objects

This chapter presents the Reconfigurable Atomic Service for Component Objects, RASCO in short. First, section 3.1 described profile framework that this work based. Then, section 3.2 describes a new redundancy abstraction called Strips to track and handle redundant WuObjects dynamically. Lastly, section 3.3, 3.4 and 3.4.2 presents the distributed algorithm used to solve the problem.

## 3.1 Profile Framework

In this work, we build upon WuKong, a loosely-coupled component based architecture for M2M systems. WuKong uses profile framework to enable the handling of physical resources on heterogeneous sensor nodes, and for higher abstractions of software component capabilities. As future M2M systems could consist of many heterogeneous sensor nodes and actuator nodes, two main concepts in profile framework, namely WuClass and WuObject, was introduced to allow WuKong to track, and manage physical resources in the network. [11] However WuKong has no support for fault tolerance. We proposed a solution to track, manage and maintain consistency among nodes based on concepts of WuClasses and

WuObjects from profile framework. Nodes duplicate WuObjects, such as WuClass ids but not their respective ports on their respective hosts, and link information. Among the hosts that have the same WuObjects, only one is active and is the primary service provider in the eyes of other services and clients.

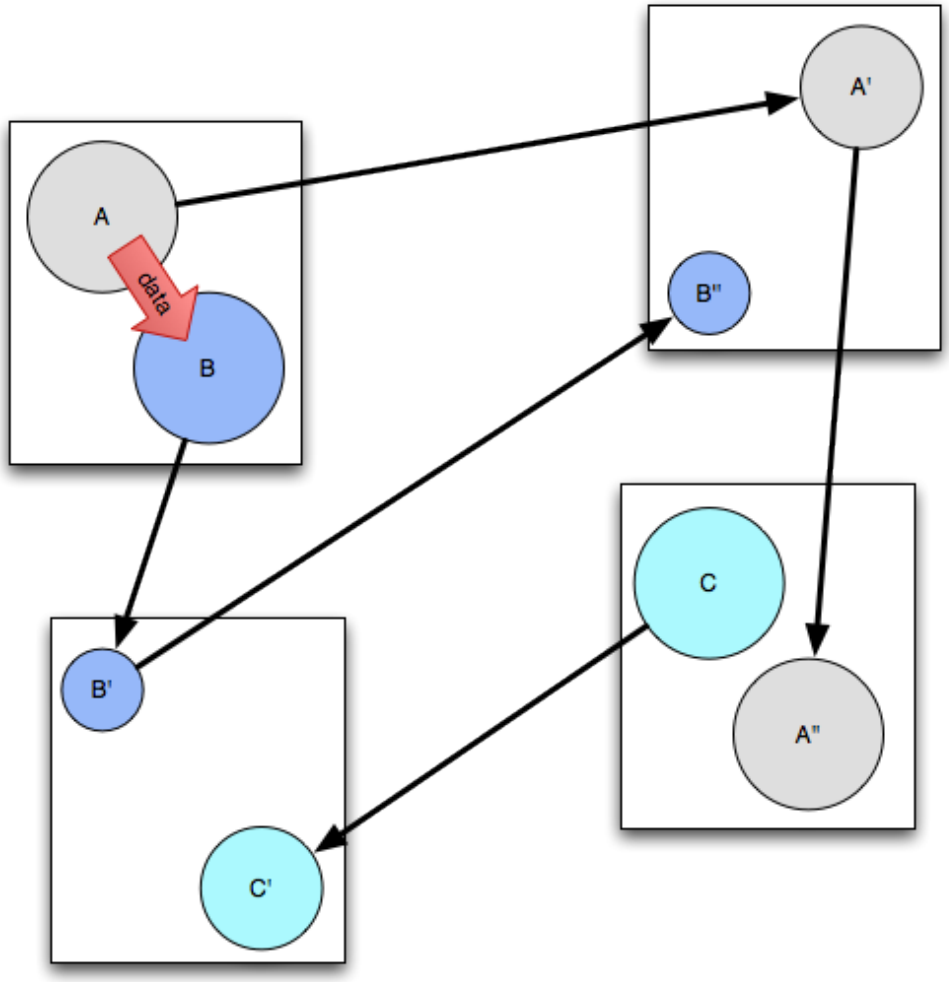
## 3.2 Strips

This work proposed an algorithm that uses strips. Strips are a sequence of WuObjects on different nodes where the node holding the first WuObject is active in the system while the rest are backups. The order of a strip is typically determined by the system that deploys the application. Strips consists of members which are chained together in series to the next that when one member failed, the next one will take over, except the last one. For example, for a strip constructed like this $\rightarrow 1-2-3-4-5$, when 3 failed 4 will take over the place of 3 and shift all the objects after it forward, and the new chain will look like this: $\rightarrow 1-2-4-5$. Now if 1 failed, 2 will take over and members after 2 (including 2) will shift one position forward that would result in $\rightarrow 2-4-5$. When a backup becomes the head of the strip, it will become active in the system.

In a heterogeneous network, each host could carry more than one WuObjects. Since each strip represents a specific component in the application, there will typically be many of these strips present in the network where each of them could crisscross with one and others. So it is typical that a node could be carrying inactive WuObjects and active WuObjects at the same time.

A node stores membership information of the strips where it is a member of. As later sections will also introduce to the heartbeat protocols that each node will be monitoring some other nodes, each node will also store the membership information of the strips

Figure 3.1: An example network with several strips



of the nodes it is monitoring. Each stripe is stored as a list with membership address information in the same order as the order of the strips so for example, the current head of the strip is the first element of the list. Nodes use the information stored to track and notify the nodes for any changes in the strips.

Figure 3.1 illustrates a network with many strips as strips crisscross and layout in the network. Each block represents a host, and each circle is a WuObject. Notice that only the head (ones with no arrows pointing at) of the strip is active and will have data flow through them as represented by a thick arrow. The name of the component represents the type of the component, and duplications have the same name as the component but with an apostrophe next to it. As shown in the figure 3.1, each strip could crisscoss and could have duplication residing with another component within the same host.

## 3.3 Decentralized Failure Detection

The system assumes a fully connected heterogeneous network with sensors and actuators where each node in the network could host multiple components. This work uses heartbeat technique, a popular technique widely used to detect failures in high-availability distributed systems. Each node would send a heartbeat message to its detector periodically in a fixed interval until it's unable to send messages anymore. Each node is therefore suspected dead when it stops sending messages after a period of time.

There are many related work on heartbeat protocols to ensure high-availability whether it uses star topology, or a ring topology; the main purpose for a heartbeat protocol is to detect failure within a network as fast as possible. Our work assumed a ring topology heartbeat protocol such that a node A would send heartbeat to node B and so on, but the last node would send heartbeat back to node A.

However, the heartbeat protocol used in the network is not directly related to the ordering of the strips. The heartbeat protocol is a layer below the strips as a support for network fault detection, the algorithm above will take advantage of the given information from the layer below to recover the system.

One of the reasons this is good for the system is extensibility and separation of concerns, by excluding both domains into separate layers each layer could optimize on its own record without being binded by another layer which might hinder each other. The separation of concerns would make each layer modular and be able to switch and plug in for a different algorithms that might result in a better performance for a particular deployment. Besides, because every node could carry more than one WuObject, and the distribution of the WuObjects might not always be ideal and evenly spread out to the network, heartbeat protocol designed around current strip assignments will not be able to cover the whole network or to be optimized in communication overhead.

## 3.4 Failure Recovery

When a failure is detected, there are two tasks that the system will have to do to recover from failures. First it has to make sure all members that carries the strips in the failure nodes will have consistent view of the strips. Second, it would need to propagate the changes to reconfigure other parts of the system that depend on the locations of the heads of the affected strips in order to function. The work built on a stateless system as defined by WuKong middleware with FBP applications where none of the services need to store any states or to remember any history. For example, WuKong applications don't have services that require to store past values of some variables. Therefore strips members (except the head) with inactive WuObjects are not the same replication as defined in other

related work, but simply as backups that will take over when the active WuObject failed.

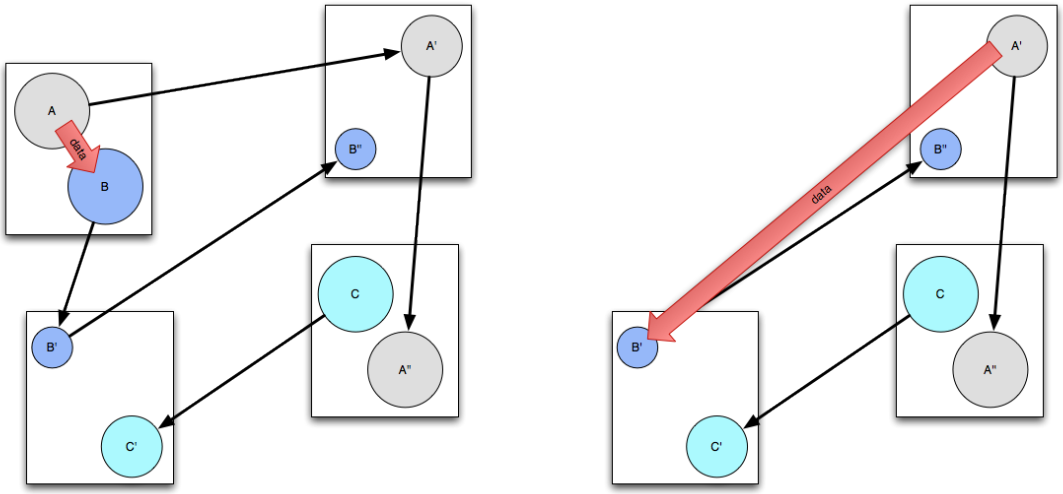
### 3.4.1 Consistency among strip members

Typically several strips would be cut off in the event of failure, and if the failed node carries some active WuObjects from some strips, the system would not able to continue to function. RASCO will attempt to recover by letting the detector of the failure to initiate the recovery algorithms. Since the detector will be responsible for recoverying for the failed node, every node needs to have membership knowledge of the strips from the nodes it is monitoring. For example, if node A is monitoring node B, A would know the members of all strips in node B in addition to its local strips. Strips only specifies the order of recovery, it is not correlated with the network structure for the fault detection, in other words, a strip with A and B doesn't mean B is monitoring A, as B could be monitored by C which depends on the structure of heartbeat protocol layer. In the initial algorithm, the detector node will prepare a update message to inform all members of the strips with which the failed node is associated with. Assuming that every node that monitors other node will have knowledge of the strips that it contains and the members that the strips pertain. The node would send out a marker message first to confirm the nodes which are still functioning, and once all acknowledges have been received, it will proceed to send the update message to update their local knowledge of the strips to reach a consensus. The ordering of the messages wouldn't matter since the end state of any failure sequence for any strip would be the same. For example, given a strip of three members $\rightarrow 1-2-3$, if the updated failure sequence is given in any permutation by $[1, 2]$ or $[2, 1]$, the end results would be the same $\rightarrow 3$ since the remaining members from those two failure sequence is the same and the relative order of the members would stay the same. Therefore there

is no need for extra communication overhead to maintain ordering to gaurantee level of consistency between members since they will all come to the same conclusion given each receiver receive the same messages. The overhead are messages required to update each member's internal membership information.

### 3.4.2 Reconfiguration

Figure 3.2: A reconfiguration of a Network




Even though consensus of the new configuration for each affected strip has been reached, some nodes with component that acts as a client to another component for data or the reverse would also have to have consensus on the updated primary holder of each affected component. And the node that is monitoring the node monitored by the dead node would also need to update on its knowledge of its monitored node in order to recover the

next possible failure.

RASCO will initiate a reconfiguration service to reconfigure the network to adapt to the new strip configurations. Reconfiguration service is implemented by a distributed algorithm which is inititated by the detected nodes. First the initiator node would have to identify the components that are reading/writing data with the components carried by the dead host. This would be done by requesting the link information between components at the higher level provided by the application. Then each member of the strips of the connected components would be updated with the information about the change in the host of the failed components with a *reconfig* message containing the change in the location of the specific WuObjects. Right after receiving the *reconfig* message, if the connected nodes whose WuObjects are pushing data to the WuObjects on the failed nodes, they will force to initiate a data push to update and bring the recovered nodes up to date, and vice versa, so the nodes whose WuObjects are receiving data from the failed nodes will initiate a data pull after receiving the *reconfig* message.

As shown in figure 3.2, the network on the right has a node failed that bring both current heads of strips A, and B down. As shown on the right, after recovery, the nodes that have the active WuObjects connecting to each new heads in both strips A and B both will be updated on the current location of the heads of both strips. Because the bottom left node contains the new head of strip B and is connected to strip A, therefore it will be updated on the location of the new head of strip B, and vice versa.

Reconfiguration service has message complexity of $O(m)$ where m is the number of strips where its components are linked to the failed components.

# Chapter 4

# Deployment of Reconfigurable Atomic Service for Component Objects

This chapter discussed the tradeoffs in determining optimal deployment for fault tolerance system such as RASCO. First it gives an overview of why it is a problem that we need to reconfigure the underlying network in a larger scale and how it is a problem to the system in itself in the long run. Then we will describe the deployment problem in the next section. Lastly, we identify certain tradeoffs in deployments for fault tolerance that could influence the system in a big way.

## 4.1 Reconfigurable Atomic Service for Component Object

RASCO allows for continuous and large changes in the underlying network. Providing consistent shared objects in a dynamic network is what RASCO is developed for. RASCO achieves this by introducing the use of strips. Strips enable the system to track, maintain replicas and maintain consistency in the presence of failures. However, the number of failures RASCO can handle is still limited by the number of spare nodes that could provide required services. It is possible after a large change in the network, none of the nodes providing a service is there anymore, so it is impossible for the newly joined nodes to

tap in and take over. To handle this large and permanent changes, WuKong supports a system progression framework that could handle such large change in the underlying network with dynamic *reconfiguration*. WuKong reconfiguration reassigns associations between software components and hosts. But it does not have a solution to what the new assignment would look like. This deployment problem is a gap that should be considered.

In the case of RASCO where deployment has to be continuously adjusted, the deployment problem is more of a online problem, so there is a tighter limit on time available to compute the assignment and strips, which makes the problem a lot more interesting.

## 4.2 Optimal deployment

The problem of deploying a specific distributed system onto a network structure typically consists of mapping the components of the system onto the hosts of the network. The mapping is subject to constraints. In the case for RASCO, the constraints are whether a node supports certain service to host certain components, and how much communication overhead would induce from the assignment to maintain consistency for the strips, and from the perspective of WuKong, some components need to seaparate from other components to achieve fault tolerance, and some needs to place together to function properly.

Determining such an optimal deployment is a *combinatorial optimization* problem. Combinatorial optimization problems are generally extremely challenging computationally. However, the problem is ubiquitous as it happens all around our lifes, whether it is companies trying to assign limited resources to meet certain objectives, or institutions allocating resources to its staff members to minimize cost, are examples of combinatorial optimization problems. However it is difficult to predict what will and what will not work. It is unlikely that a single approach will be effective on all problems or instances of the

same problems. As we also want the system to come up with a solution within a time limit, there is a time constriant on the algorithms. So finding a good balance between the *quality* of a solution of the *time* it takes to come up with a good enough solution is critical.

# Chapter 5

# Conclusion

Previous chapters have described the work that has been done on WuKong platform with WuObjects. It is useful to reflect on what has been accomplished and place them in the broader context of the more general fault tolerance problem as well as the specific contributions of this work.

## 5.1 Discussion

We introduced RASCO, a reconfigurable fault tolerance system for tracking, managing duplicated WuObjects, or software components where multi-model nodes are assumed in the network. We also described Strips, a redundancy abstraction, on how they are used in RASCO to achieve consistent views among strip members with no need of any message ordering guarantees. Then we described algorithms that made this kind of architecture recovery from failures and reconfigure other parts of the network in linear message complexity. The solutions developed for WuKong system performs very consistently, as the recovery time is always under two secs on average with a handful of sensors. However, occationally heartbeats could report erroneous failures and result in a quick false recovery, and left the partitioned nodes behind, but the system as a whole still

function properly.

The developed methods adds new and useful solutions to build a fault tolerant system that could be reasoned easily and with a performance as expected on average. This method might not be superior to other methods in terms of completeness and complexity, but it serves as a quick, easy and unbias solution to provide decent fault tolerance for distributed systems.

## 5.2 Future Work

We have shown that it is possible to design a reconfigurable fault tolerant system without any message ordering properties to achieve consensus compared to other related works with minimum communication overhead. Strips makes it really easy to describe a component system with redundancy and it scales with well with complexity. The reconfiguration algorithm is also shown to take linear message complexity to recover from failures. However, there is clearly more work to be done. This section will address some directions future research can take.

Even though when failure occurs, RASCO does not do lock step to prevent other parts of the system to halt while doing the recovery, RASCO can only handle one failure at a time, since without a distributed locking mechanism, simultaneous failures occurring across the network would put detectors in discordance and causes confusion as they all assume the rest of the network stays the same as before the failure. It is possible to be clever about when to relax the requirements to only update the critical component holders first, before synchronizing the rest, so when failures occurs recovery could be done more quickly and leave more time for the detector to figure out and update the rest that are still in inconsistency.

RASCO also could not account for network partitions where a group of nodes could be disconnected from the network for a period of time and then come back as RASCO encourages the system to recovery as quickly as possible given the current events, it leaves no room of tolerance for nodes partitioning away from the network. But given how often those partitioning occurs, it is important to be able to tell from a partition or not, or employ a tolerance period to compensate when the nodes are gone, and revert back when they are back. But in a partition scenario, two partitions of the network could both be thinking each other's dead, therefore both will try to be active and take over the network, this is typically described as the split brain problem in literature. Therefore it is also important that the split brain problem will be prevented when the split occurs. One possible direction is enhancing failure model. Heartbeat is a useful heuristic to detect possible failures in a distributed network, however the binary failure model supporting this heuristic is usually too simple in the context of most distributed systems that it usually gives false positive results on the health of the monitored nodes, thus it couldn't detect possible network partition as a result. There are existing work on more accurate failure detection models that could shine a light on this problem.

As we have stated in previous chapters, this thesis assumed a stateless application where no services need to store any past states, such as a transactional database. There is also a direction this research can take is to take Strips and make it work on applications with states, as a field of applications require services that store past records in a certain order.

That beings us to another important element, which is the applicability of the developed methods on other types of distributed systems. Even though we have shown how the developed method works under systems with stateless applications. There are other problems that might emerged from different time or frequency communciation requirements

between nodes that would require a more rigorous methods to handle.

Another important element is the deployment problem as we have stated in chapter 4 when reconfiguration needs the assistent of external mastermind such as WuKong Master. As there are limitations to the number of failures one network could take, the optimal deployment problem is definitely one direction this research could take.

The major limitation on the developed methods is the scale of the distributed systems on which they are applied. Many real world deployment is much larger in size which consist of more nodes with components. Currently, our method cannot scale well enough for such network. As the network size increases, the memory used for Strips will exceed the limits and reconfiguration time will explode. A challenging and interesting direction is to investigate the possibilities of scalability with Strips, and reconfiguration algorithms. A possible option is to divide a large network into small partitions which can be deployed. But whether it could still maintain atomic services, and to what extend this approach can lead to desired performance still remains something to be investigated.

# Bibliography

[1] V. A. Archana Bharathidasan. Sensor Networks: An Overview.

[2] A. Arora, P. Dutta, S. Bapat, and V. Kulathumani.... A line in the sand: A wireless sensor network for target detection, classification, and tracking. *Computer Networks*, Jan. 2004.

[3] G. Barrenetxea, F. Ingelrest, G. Schaefer, and M. Vetterli. The hitchhiker's guide to successful wireless sensor network deployments. In *Proceedings of the 6th ACM conference on Embedded network sensor systems SenSys 08*, volume D, pages 43–56. ACM Press, 2008.

[4] L. Lamport. Paxos made simple. *ACM SIGACT News*, 2001.

[5] H. Liu, A. Nayak, and I. Stojmenović. Fault Tolerant Algorithms / Protocols in Wireless Sensor Networks. In *Guide to Wireless Sensor Networks Computer Communications and Networks 2009*, pages 261–291. 2009.

[6] A. D. Luna, S. Aknine, and J. Briot. Dynamic resource allocation heuristics for providing fault tolerance in multi-agent systems. *Proceedings of the 2008 ACM symposium on Applied computing SAC '08*, 2008.

[7] N. Lynch and A. A. Shvartsman. RAMBO: A Reconfigurable Atomic Memory Service for Dynamic Networks. *DISC '02 Proceedings of the 16th International Conference on Distributed Computing*, pages 173–190, 2002.

[8] J. Neumann, N. Hoeller, C. Reinke, and V. Linnemann. Redundancy Infrastructure for Service-Oriented Wireless Sensor Networks. In *2010 Ninth IEEE International Symposium on Network Computing and Applications*, pages 269–274. IEEE, July 2010.

[9] P. Padhy, K. Martinez, A. Riddoch, H. L. R. Ong, and J. K. Hart. Glacial Environment Monitoring using Sensor Networks. *RealWSN*, pages 10–14, 2005.

[10] J. Polastre, R. Szewczyk, A. Mainwaring, D. Culler, and J. Anderson. Analysis of wireless sensor networks for habitat monitoring. In C. S. Raghavendra, K. M. Sivalingam, and T. Znati, editors, *Wireless Sensor Networks*, chapter 18, pages 399–423. Kluwer Academic Publishers, 2004.

[11] N. Reijers, K.-j. Lin, Y.-c. Wang, C.-s. Shih, and J. Y. Hsu. Design of an Intelligent Middleware for Flexible Sensor Configuration in M2M Systems. *Sensornets*, 2013.

[12] I. Stoianov, L. Nachman, S. Madden, T. Tokmouline, and M. Csail. PIPENET: A Wireless Sensor Network for Pipeline Monitoring. *6th International Symposium on Information Processing in Sensor Networks, 2007. IPSN 2007.*, pages 264–273, 2007.

[13] J. Sussman, I. Keidar, and K. Marzullo. Optimistic Virtual Synchrony. *Reliable Distributed Systems, . . .* , (914), 2000.

[14] J. Tateson, C. Roadknight, A. Gonzalez, T. Khan, S. Fitz, I. Henning, N. Boyd, C. Vincent, and I. Marshall. Real World Issues in Deploying a Wireless Sensor Network for Oceanography. *RealWSN*, 2005.

[15] G. Tolle, J. Polastre, R. Szewczyk, and D. Culler.... A macroscope in the redwoods. *Proceedings of the 3rd international conference on Embedded networked sensor systems SenSys '05*, Jan. 2005.

[16] G. Werner-Allen, K. Lorincz, and J. Johnson.... Fidelity and yield in a volcano monitoring sensor network. *Proceedings of the 7th symposium on Operating systems design and implementation OSDI '06*, pages 381–396, Jan. 2006.