2010 Fourth International Conference on Sensor Technologies and Applications

Automatic Application Object Migration in Sensor Networks

Paul Hunkin
Department of Computer Science
Waikato University
Hamilton, New Zealand
Email: pwh4@cs.waikato.ac.nz

Tony McGregor
Department of Computer Science
Waikato University
Hamilton, New Zealand
Email: tonym@cs.waikato.ac.nz

***Abstract*—Object migration in wireless sensor networks has the potential to reduce energy consumption for a wireless sensor network mesh. Automated migration reduces the need for the programmer to perform manual static analysis to find an efficient layout solution. Instead, the system can self-optimise and adjust to changing conditions. This paper describes an automated, transparent object migration system for wireless sensor networks, implemented on a micro Java virtual machine. The migration system moves objects at runtime around the sensor mesh to reduce communication overheads. The movement of objects is transparent to the application developer. Automated transparent object migration is a core component of Hydra, a distributed operating system for wireless sensor networks that is currently under development. Performance of the system under a complex performance test scenario using a real-world dataset of seismic events is described. The results show that under both simple and complex conditions the migration technique can result in lower data traffic and consequently lower overall energy cost.**

***Keywords*-Wireless Sensor Networks; Operating Systems; Distributed Systems**

I. Introduction

Recent years have seen the development of wireless sensor network hardware and software systems, and a variety of operating system, middleware and programming methods [10]. However, application programming for sensor networks is still a challenging and difficult task. Developers must contend with the complexities of both embedded and distributed systems, while often coding using very platform-specific systems. In many cases, developers must be explicitly aware of a number of complex issues. These may include limited energy and other hardware resources, wireless vs wired communications, asynchronous event processing, differences between different types of sensor node hardware.

Because wireless sensor networks are a relatively new area and one that has significantly different requirements than most other areas of software development, no standard paradigm for programming WSNs has emerged. Many systems have been developed to run on wireless sensor nodes, most of which can be divided into the following categories.

- Single node operating systems, such as TinyOS [5] and Contiki [2].
- Virtual Machines such as Mate [4], that allow easier programming and hardware abstraction along with energy savings from potentially smaller bytecode programs.
- Group-level abstractions such as SpatialViews [7], that provides a method for dividing up the mesh of sensor nodes into logical groups
- Network-level abstractions such as TinyDB [6] and Cougar [3], that abstract away the entire mesh.

In this paper we propose an automatic object migration methodology that relocates objects to reduce energy costs. The system forms an essential component of Hydra, a new distributed operating system designed to run on wireless sensor networks.

II. Background

Existing wireless sensor network application programming paradigms each have benefits and drawbacks. The single-node programming model provides the maximum flexibility in terms of writing applications that run inside the sensor mesh. As the range of uses of wireless sensor networks is always increasing, this flexibility is a major advantage. However the drawback of this approach is the increased complexity caused by requiring the programmer to have knowledge of the sensor mesh topology and to manage the interactions between nodes. Conversely, the group and network level abstractions simplify WSN applications, but sacrifice programming flexibility. In many cases, systems are limited to simple data retrieval.

A distributed operating system makes a collection of individual computers (or 'nodes') appear as one logical computer to the user. Hydra provides the end-user with a virtual machine that conceptually treats an entire mesh of sensor nodes (or other compatible devices) as one logical machine with access to the resources of the wireless sensor network, such as its sensors and processing capacity. User applications are programmed in a subset of Java.

This strategy allows the sensor network developer to create complex applications easily, without being aware of the details of the underlying network communications or the typical distributed systems issues that arise with low-level sensor network programming techniques. The use of Java means that many of the programming constructs, APIs and development tools that are commonly used in desktop

978-0-7695-4096-2/10 $26.00 © 2010 IEEE
DOI 10.1109/SENSORCOMM.2010.29

or server development can be utilized on sensor networks. This lowers the barrier to entry for new WSN users.

A Java programs can be thought of as a collection of objects, where each object contains a number of methods and/or references to other objects. When the Java bytecode for an object method requires an operation on another object, the operation is invoked either locally or remotely, conceptually similar to a standard Java remote method invocation. Remote object invocation is transparent to the programmer.

To support object migration, the data consumed by each object to a remote node is recorded. The amount of data used by each object is periodically tallied and the object may be dynamically migrated to another location in the mesh.

For example, in an application that detects movement via accelerometers that are spread across a sensor mesh, the program objects that are performing the monitoring would migrate to the nodes that are experiencing the most sensor activity. If the majority of the activity moved to another part of the mesh, then the system can migrate the objects to the new center of activity provided resources are available.

The Hydra virtual machine can be run on WSN hardware as well as on desktop or server Linux systems, communicating with the sensor node virtual machines over a network link. We have tested Hydra on 16-bit MSP430-based Scatterweb nodes, and 32 or 64-bit Linux. Platform independence allows for hybrid approaches to sensor network hardware–for example, a sensor network that requires mass storage as part of its application might associate with a Linux-based JVM that allows use of its hard drive. As an example, a Hydra mesh might contain a large number of battery powered sensor nodes, a smaller number of solar powered one-board PC nodes for processing capacity and an externally powered PC with a disk for data storage.

In another scenario, a variety of hardware models of sensor nodes could belong to the same mesh. As long as they have some method of communication, the hardware abstraction given by the JVM means that they can share data and code seamlessly.

III. Background

A. Hydra - A distributed WSN OS

The WSN distributed operating system technique has several advantages to wireless sensor networks:

- Resource sharing: resources on any individual node are available from any place in the system. Sensor node programs can request data from sensors elsewhere in the mesh in order to optimize performance. Hydra applications use this to abstract away the details of sensor locations.
- Dynamic adaptability to changing conditions: Node failure is a common issue in sensor networks due to their limited power and cheap hardware. If a node becomes overloaded, or indicates the imminent failure due to situations such as power failure, the system can handle this by relocating tasks away from that node. Likewise if the environment or network layout changes, the system can adapt.
- Scalability: Scaling the system for increased workloads can be supported in many cases by adding more nodes at runtime. For example, a mesh can be extended to a new area. Hydra will automatically detect the new resources and begin to move tasks to them as needed.
- The programming and effort required to optimise a WSN is reduced because analysis and runtime power optimisation is handled by Hydras object migration.

Most distributed operating system projects are implemented on relatively sophisticated hardware with high energy cost requirements–unlike wireless sensor networks. Building a distributed operating system that operates on the very limited hardware resources of a wireless sensor network is one of the major challenges of this project.

For most wireless sensor networks, power is very limited. Using the radio to communicate is a heavy power drain. This is potentially problematic for distributed operating systems–communication is a fundamental part of the concept. As a consequence, Hydra must minimize communication.

Object migration is the primary mechanism used to provide efficiency in the Hydra distributed operating system, making more efficient use of limited power. Unlike mobile agents, Hydras application components are not aware of their movements - movement is managed by the lower level operating system layer.

B. Communication minimization

Hydra requires a general purpose virtual machine bytecode that can be changed at runtime. A general purpose instruction set VM bytecode is typically still more compact than machine code, as well as being hardware-agnostic–the same bytecode can be executed under many different environments. The system is free to modify the bytecode at runtime if needed, as long as this does not modify overall application behaviour.

To perform partitioning, applications are divided into various logical objects and operations are performed on these objects instead of on the entire process. This allows a complex program that deals with a variety of different sensor sources to be automatically and transparently split into components. These components may execute on different nodes. This finer granularity makes much more efficient power optimisation possible and, at the same time, reduces the amount of program data that will be sent as the individual objects will be smaller than the entire program. Application partitioning forms the basis of the mobile agent payload–each object can be thought of as an individual ‘agent’, that communicates with other agents to perform the goal of running the program. The end-user has no knowledge of the individual node hardware or network layout.

At first sight the use of Java with wireless sensor networks may appear unusual. However with careful implementation, a micro JVM can be built for WSNs, and provides several advantages.

1) The resulting bytecode is able to be split into chunks by Java Object boundary.
2) The instruction set and associated standard library is general purpose, but not so complex that it cannot be implemented on a wireless sensor node.
3) Suns official compiler and linker toolchain exists that can turn an application written in a high level language into the bytecode–this decreases the work necessary for the project.

While some micro JVMs do exist for sensor networks (such as Darjeeling [1]), at the time of writing their code is not available for general use. Some other open-source JVM implementations such as LeJOS [8] were also evaluated, but were not suitable for the task at hand due to their focus on non-sensor-network tasks. Most of the other JVM implementations are not suitable to run on sensor network hardware and require more resources than our target hardware platform has available. Consequentially a JVM was developed specifically to support Hydras requirements.

C. Remote Object Characteristics

Java programs are strongly object orientated - everything in a Java program inherits from the base class java.lang.Object. This makes object instances a convenient mechanism for dividing programs into independent sections.

To support object migration, object methods must be able to be invoked on objects that were local but have now been migrated on a remote node. When the JVM detects that the currently executing bytecode is attempting to invoke a method on an object, the JVM invokes the method locally or remotely depending on where the object is located in the mesh using a remote procedure call. This process is completely transparent to the user application.

On object initialisation, each instance of an object is assigned a unique identifier that is used to refer to this object across the mesh. Each JVM then maintains a mapping of objectid to remote node location or local object instance. Each JVM does not need to know the location of every object, however. A simple example: JVM-1 moves an object to JVM-2. JVM-2 now has all the necessary information to perform operations on that object. In the event that an object moves more than once, the record of where an object is may be out of date. In this case, a request for a remote operation on the moved object goes from JVM-3 to JVM-1, JVM-1 will return a message to JVM-3 indicating the new location of the object–JVM-2.

Object IDs are assigned based upon the ID of the node it was created on. This allows a simple mapping of objectID to probable location of the object. In the case that the object has moved from its origin, the origin node should be able to provide the updated location. In our testing to date, this algorithm performs effectively. However, it is susceptible to losing objects if nodes fail. The primary purpose at this stage of our work is to demonstrate that our approach to object migration is feasible and effective in a WSN environment. The naming and location algorithm described here is satisfactory for this purpose. More effective naming and migration algorithms are the subject of ongoing work.

IV. Migration in Hydra

The Hydra JVM tracks the data sent between each object and remote nodes. This data is used to trigger migration of objects between nodes. At this point, two reference algorithms ('none' and 'perfect'), one simplistic algorithm ('naive') and one heuristic algorithm ('Routing') based on routing and visibility data has been implemented.

A. Reference

These provide benchmarks we use for evaluation purposes.

- 'None' - No migration occurs. Object positions are static throughout.
- 'Perfect' - This algorithm uses full knowledge of the mesh layout (which is available under the simulation environment, but not in real life). This is also an expensive (exponentially so) algorithm, and so is unsuitable to run on CPU-limited sensor devices.

B. 'Naive'

This is a heuristic algorithm that monitors the communications an object has with other nodes. Objects are then migrated to the node they exchange the most data with.

```
costToMove = (objSize * objWeight) - objLocal
costToStay = objRemote[i]
if(costToStay - costToMove > threshold){
        move(object, i)
}
```

C. 'Routing'

The 'Routing' algorithm is the approach we are currently investigating. It is a heuristic algorithm that attempts to find an intelligent placement, based on the individual nodes knowledge of the mesh.

Each timeslice, each node considers the adjacent nodes (ie: one hop away) as candidates to move objects to. For each object, the node attempts to predict the resulting saving in communication cost if the object was moved to the candidate adjacent node. The object is then moved accordingly. The accumulated savings knowledge of the object is also transmitted, so the receiving nodes knowledge of predicted savings is updated.

The predicted saving is calculated using several pieces of knowledge

- That the cost of communication with a node will decrease by the cost of the link if the node is moved over that link
- That the maximum increase in the cost of the rest of the communications to objects which we use a different path to communicate to is the current cost plus the cost of the link to the new home node being considered.
- Reachability and cost data that may have been passed from other nodes as part of of a previous object migration.

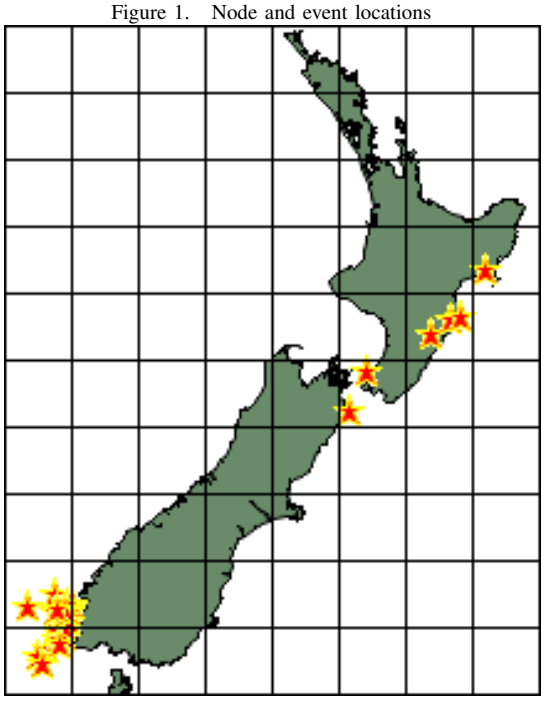
V. Testing

Initial testing of the Hydra prototype has proved promising. Development of Hydra includes both Scatterweb MSP430-based nodes, a Linux VM, and a Linux-based test harness. The Linux test environment aids in data collection and processing.

A. Earthquake Data Test

This test demonstrates the system in a complex layout, with real event data. The event data consists of a chronological set of seismic measurements taken during a recent earthquake event in Fiordland, New Zealand [9].

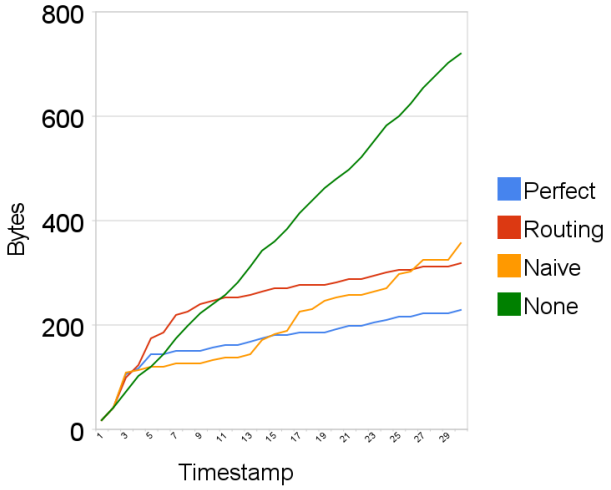
Figure 1. Node and event locations



The test scenario is a mesh of 8x10 nodes in a regular grid, representing a regular mesh covering. Nodes communicate only with their immediate neighbours above, below, left and right. They cannot communicate diagonally. Multihop networking is enabled. The timeline begins slightly before the main earthquake event. 40 seismic events are fed into the mesh, with the initial events are scattered across New Zealand. Soon they they become clustered around the earthquake epicentre, with occasional unrelated events over the rest of the country. The initial object location is placed in the central North Island.

The graph below shows the cumulative data usage of the entire mesh during the experiment, with each line representing a different migration algorithm.

Figure 2. Cumulative data usage



The graph shows that:

- Initially the performance of all algorithms is identical, as no migration has occured
- The ‘None’ algorithm performs poorly throughout, while the ‘Perfect’ reference algorithm performs the best
- At timestamp 3, the first migration event occurs
- The results of each algorithm begin to differ at timestamp 5
- Though the ‘Naive’ algorithm first appears to perform the best, it quickly deteriorates. It is also much less stable than the other algorithms, with sharp rises in data usage
- Though the ‘Routing’ heuristic initially performs worse than the naive algorithm, once enough placement knowledge has been transmitted, it begins to perform similarly to the ‘Perfect’ algorithm and begins to outperform the naieve approach.
- ‘Routing’ is also much more stable than ‘naieve’.

VI. Future work

More complex testing with larger-scale data that closely models other real world wireless sensor network deployments is underway. Testing on real-world sensor network hardware running Hydra is also planned. This will help explore the effectiveness of the system under more complex and realistic scenarios.

We also plan to address the issue of node failure and redundancy in a future publication.

VII. Conclusions

An essential component of our Hydra distributed operating system for wireless sensor networks is the ability to transparently migrate objects between nodes. This is particularly important in a WSN distributed OS because it allows communication costs to be reduced, hence reducing power usage. It can also improve performance and reduce the programmer effort required to create complex WSN applications. The object migration system developed for Hydra and described here supports fine grained transparent object migration, using our custom micro JVM that runs on a sensor network platform.

The earthquake test scenario demonstrates the application of the system to more complex real-world data events. Open work includes the investigation of alternative algorithms for object naming and location as well as migration triggers. Other aspects of the Hydra design are also being pursued, including better fault tolerance and replication.

References

[1] N. Brouwers, P. Corke, and K. Langendoen. A java compatible virtual machine for wireless sensor nodes. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 369–370, New York, NY, USA, 2008. ACM.

[2] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, pages 455–462, Washington, DC, USA, 2004. IEEE Computer Society.

[3] W. F. Fung, D. Sun, and J. Gehrke. Cougar: the network is the database. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 621–621, New York, NY, USA, 2002. ACM.

[4] P. Levis and D. Culler. Maté: a tiny virtual machine for sensor networks. In *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 85–95, New York, NY, USA, 2002. ACM.

[5] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. Tinyos: An operating system for sensor networks. pages 115–148. 2005.

[6] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[7] Y. Ni, U. Kremer, A. Stere, and L. Iftode. Programming ad-hoc networks of mobile and resource-constrained devices. In *PLDI '05: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, pages 249–260, New York, NY, USA, 2005. ACM.

[8] J. Solrzano. *leJOS*. URL: http://lejos.sourceforge.net/.

[9] stuff.co.nz. *More earthquakes jolt Fiordland*. Story from http://www.stuff.co.nz/national/2724208/More-earthquakes-jolt-Fiordland.

[10] R. Sugihara and R. K. Gupta. Programming models for sensor networks: A survey. *ACM Trans. Sen. Netw.*, 4(2):1–29, 2008.