DESIGN AND IMPLEMENTATION OF A SINGLE SYSTEM IMAGE OPERATING SYSTEM FOR AD HOC NETWORKS

Hongzhou Liu, Tom Roeder, Kevin Walsh, Rimon Barr, Emin Gün Sirer
Department of Computer Science, Cornell University
MobiSYS 2005
Outline

- Introduction
- System Overview
- Evaluation
Introduction
"The network is the computer"
Ad hoc networking applications differ from traditional applications in three fundamental ways.

- Ad hoc networking applications are inherently distributed.
- Ad hoc networks are typically highly dynamic and resource-limited.
- Ad hoc networks application are expected to outlast the lifetime of any one node.
Introduction

- Developing applications for ad hoc networks
- The current state
  - treat the network as a system of standalone systems.
- In this paper
  - A thin distributed operating system layer makes the entire network appear to applications as a single virtual machine.
  - MagnetOS
Introduction

MagnetOS provide two advantages

- Simplifies the development of applications substantially.
- Enables the underlying system to make energy-efficient placement and migration decision.
Introduction

The goals of MagnetOS
- Efficiency
- Adaptation
- Generality
- Extensibility
- Compatibility
MagnetOS applications
- comprised of event handlers that communicate with each other by raising well-typed events.

Event signatures
- specify the types of the arguments passed with the event, as well as the return type of the handler.
Introduction

Static partitioning service

converts regular Java applications into distributed components that communicate via events by rewriting them at the bytecode level.
Introduction

Present practical, online algorithms for finding an energy-efficient distribution of application components.
Introduction

Three Contributions

- It proposes a novel programming model for ad hoc networks
- Propose practical, adaptive, online algorithms for finding an energy-efficient placement of application components in an ad hoc network.
- Demonstrate that these algorithms achieve high-energy utilization, extract low overhead and improve system longevity.
System Overview
System Overview

– Static Partitioning

MagnetOS partitions applications based on programmer annotations and object boundaries.

This transformation at class boundaries preserves existing object interfaces, and inter-object invocations define events in MagnetOS.
System Overview

– Static Partitioning
System Overview
– Static Partitioning
System Overview

– Static Partitioning

The complex partitioning services need only be supported at code-injection points, and can be performed offline.

Since applications are verified prior to injection into the network.
System Overview

– Static Partitioning

Static partitioning extracts

- an event handler
- a dispatch handle
- an event descriptor
- a set of event global associated with the event handler

from each original class.
System Overview

– Static Partitioning

Event handler
- Is a modified implementation of the original class
- Stores the instance variables of the corresponding object.
System Overview
– Static Partitioning

Dispatch handles
- Used to invoke procedure calls on remote event handlers residing on other nodes.

Event descriptors
- Capture the event signatures that the original code exposes to the rest of the application.
System Overview

– Static Partitioning

OS

RPCs

intercepted

Dispatch handle

Raise event

handler
System Overview

- Dynamic Object Management

MagnetOS runtime services include:
- component creation
- inter-component communication
- Migration
- garbage collection
- Naming
- event binding
Object Creation

- An application contacts the local runtime and passes the requisite type descriptor and parameters for creation.
- The runtime then has the option of placing the newly created handler at a suitable location with little cost.
- Once created, the (remote) runtime simply initializes the handler by calling its constructor and returns a dispatch handle.
Remote Invocation and Migration

The runtime transparently handles invocations among the event handlers distributed across the network.

Each runtime keeps a list of local event handlers.

Dispatch handles maintain the current location of the corresponding handler, and processes raise events on behalf of application invocations.
System Overview

- Dynamic Object Management

MagnetOS can migrate both active and passive event handlers.

Passive event handler
- consists purely of static data associated with an event handler
- migrated at run time by serializing handler state and moving it to a new node.
When event handlers migrate

MagnetOS informs dispatch handles of relocation lazily

- In order to reduce the energy cost of migration

- This notification is accomplished through forwarding references left behind
System Overview

- Dynamic Object Management

- Dispatch handles
- Event handler

forwarding references left behind
Migrating active event handler

- Is a costly operation
- When node battery level is below a critical threshold required to offload all event handlers
- Requires effectively capturing the current state of the computation
- Static partitioning service injects code to periodically check a flag in each basic block.
System Overview

Dynamic Object Management

Handling Failures

Since attempting to preserve single-system application semantics in the presence of such failures is futile

- due to well-known impossibility results,

MagnetOS provides reasonable defaults for common cases and exposes the unrecoverable errors to applications.

System provide at-most-once semantics for event invocation.

Each invocation carries a unique identifier
System Overview

Dynamic Object Management

- Failure detection can be deferred until a node requires the results of a computation executed.

- The system uses keep-alive messages for long-running synchronous event invocations.

- Unrecoverable errors are reflected to applications as a special run time exception.
MagnetOS performs health checks only when a node is waiting on the result of a computation from another node.

The migration layer uses forwarding pointers to track objects that have moved, when the pointer chain is broken due to a failure, falls back to a broadcast query for the target event handler.
The MagnetOS runtime provides an explicit interface by which application writers can manually direct component placement.

This interface allows programmers to establish affinities between event handlers and ad hoc nodes.
System Overview

- Dynamic Object Management

Two levels of affinity

- **strong affinity**
  - anchors the code to that node.

- **weak affinity**
  - immediately migrates the component to the named node
  - allows the automated code placement techniques
System Overview

- Event Handler Placement

All of algorithms share the same basic insight:

- they shorten the mean path length of data packets by moving communicating objects closer together.

They do this by profiling the communication pattern of each application in discrete time units, called epochs.
System Overview

Event Handler Placement

In each epoch, every runtime keeps track of the number of incoming and outgoing packets for every object.

At the end of each epoch, the migration algorithm decides whether to move that object, based on its recent behavior.
System Overview

- Event Handler Placement

LinkPull

- Collects information at physical link level
- Migrates components one hop at a time
- Keep a cont of the message sent to and from each neighboring node.
- Move object along the link with greatest communication
System Overview

— Event Handler Placement

PeerPull

- Operates at the network level
- Migrate components *multiple hop at a time*.
- Finds the host with which a given object *communicates the most* and migrates the object directly to that host.
System Overview

- Event Handler Placement

**NetCluster**

- Nodes that shares the same next or last hop on the route are considered to be in the same cluster.
- Finds the cluster that a object *communicates with the most*.
- Migrate the object to a *randomly chosen* node within that cluster.
System Overview

- Event Handler Placement

TopoCenter(1)

- Migrates components according to a partial view of the network built at each node.
- Each node attaches its single-hop neighborhood to outgoing packets.
- Nodes on the path extract the topology information from packets they forward.
System Overview

- Event Handler Placement

TopoCenter(Multi)

- Similar to TopoCenter(1)
- Nodes attach **all the topology information** they know to outgoing packets.
System Overview

– MagnetOS API

Node

- Encapsulates the notion of a physical host running a MagnetOS instance.
- Enables applications to name and query nodes.
- Node handle can be used to:
  - Migrate application components
  - Anchor an object to a specific location
  - Query node
System Overview

– MagnetOS API

**Link**
- Refers to a physical link between two nodes

**NeighborSet**
- Is the set of single hop neighbors of a given node

These two abstraction enable applications to discover the network topology
System Overview

– MagnetOS API

**Energy**

- Enables applications to query the energy level of the underlying platform.

**Timer**

- Enables applications to schedule events to be invoked at a predetermined time in the future.
- Can migrated between nodes.
System Overview

– MagnetOS API

**Lock**

- Is a remote synchronization object
- Analogous monitor in Java
- Acquisitions and releases are converted into remote operations on the event handler.
System Overview

– MagnetOS API

**Thread**

- In MagnetOS is an execution context that can perform a sequence of synchronous event raised.
- Can migrate from one node to another.
System Overview

- Implementation Support for Ad Hoc Networks

- Runs on any platform that supports Java JDK1.4.

- On Linux
  - use an efficient in-kernel AODV implementation.

- On other platforms
  - use a user-level version of AODV.

- Routing algorithm is independent from the rest of the runtime.
System Overview

– Implementation Support for Ad Hoc Networks

RMI are inadequate when operating on multihop ad hoc network.

Built a custom RPC package based on a reliable datagram protocol (RDP).

Higher-level policies in MagnetOS require information on component behavior to make intelligent migration decisions.
MagnetOS keeps a cumulative sum per component per epoch

Periodically informs the migration policy in the system of the current tally.
Evaluation
Evaluation

Benchmarks & Workload

Three representative applications

- SenseNet
  - Run on a 14 by 14 grid of sensors.

- Publish-Subscribe
  - Consists of ten channels each with four subscribers.

- FileSystem
## Evaluation

### Simulation Methodology

#### Simulation
- Implemented a significant part of the MagnetOS system in sns.

#### Physical test
- Implement these benchmarks on top of system that support x86/Windows, x86/Linux and StrongArm/PocketPC
Evaluation

Simulation Methodology

Comparison strategies

- **Static Centralized**
  - Assign all movable objects to a single, central node in the network.

- **Random**
  - Selects a random neighbor as destination for each movable object at each epoch.

- **Past Optimal**
  - Assumes that every node in the network has full knowledge of the network topology
  - Move objects to an optimal position based on the communication pattern in the last epoch.
Evaluation

Result and Discussion

Figure 3: Automatic migration significantly extends system lifetime in the SenseNet application. Bars represent 25th and 75th quartiles.
Evaluation

Result and Discussion - SenseNet

![Graph showing total field energy over time for different methods]

- Static
- Random
- LinkPull
- PeerPull
- NetCluster
- TopoCenter(1)
- TopoCenter(Multi)
- PastOptimal

Adaptive are more shallow
Evaluation

Result and Discussion - SenseNet

b. Nodes Drained Over Time

- Number of Nodes Drained vs. Time (s)
Evaluation
Result and Discussion – publish-Subscribe

a. Operations Completed Over Time

Throughput (percent)

Time (s)
Evaluation
Result and Discussion – publish-Subscribe

b. Nodes Drained Over Time

Number of Nodes Drained

Time (s)
Evaluation
Result and Discussion – publish-Subscribe

c. Total Field Energy Over Time

Remaining Field Energy (J)

Time (s)
Evaluation
Result and Discussion – FileSytem

a. Operations Completed Over Time

Throughput (percent) vs Time (s)
Evaluation
Result and Discussion – FileSytem

b. Nodes Drained Over Time

Number of Nodes Drained

Time (s)
Evaluation
Result and Discussion – FileSytem

c. Total Field Energy Over Time

- Static
- Random
- LinkPull
- PeerPull
- NetCluster
- TopoCenter(1)
- TopoCenter(Multi)
- PastOptimal

Remaining Field Energy (J)

Time (s)
Evaluation
Result and Discussion – Node Mobility
Evaluation
Result and Discussion – Validation

![Graph showing energy consumption over event number. The graph compares Experiment (Static), Simulation (Static), Experiment (PeerPull), and Simulation (PeerPull).]
## Evaluation

### Result and Discussion – time overhead

<table>
<thead>
<tr>
<th></th>
<th>Java RMI</th>
<th>MagnetOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null</td>
<td>403±16</td>
<td>172±6</td>
</tr>
<tr>
<td>Int</td>
<td>446±9</td>
<td>180±8</td>
</tr>
<tr>
<td>Obj. with 32ints</td>
<td>991±35</td>
<td>174±4</td>
</tr>
<tr>
<td>Obj. with 4int,2obj</td>
<td>884±21</td>
<td>177±7</td>
</tr>
</tbody>
</table>

Table 1: Remote method invocation comparison. All times in μs, average of 1000 calls.
The End