UCLA
Posters

Title
Kairos: Reliable and Efficient Programming Abstractions for Sensor Networks (SYS 4)

Permalink
https://escholarship.org/uc/item/2233b9pd

Authors
Ramakrishna Gummadi
Nupur Kothari
Ramesh Govindan
et al.

Publication Date
2006
Kairos: Reliable and Efficient Programming Abstractions for Sensor Networks
Ramakrishna Gummadi, Nupur Kothari, Ramesh Govindan and Todd Millstein

http://kairos.usc.edu

Introduction

Macroprogramming

- Allows all nodes to be programmed as a single unit
- Global program behavior captured as a single task on a centralized memory model
- Advantages: Centralized programs are easier to specify, code, understand and debug than hand-coded distributed versions. Also, no need for explicit parallelization or synchronization code
- Challenge: Designing the compiler and runtime components that generate and implement an equivalent concurrent distributed version

Kairos: Language Constructs

- Node naming
  - node and node_list types that facilitate topology independent programming
  - get_neighbors() to obtain current one-hop neighbors of a node
- Node-Local variables
  - var@node: One copy of var at each node
  - Ordinary variables stored centrally, known as central variables
- Sensors and Timers
  - node-local variables with sensor/timer attributes
  - select() used to wait for timers or to read sensors
- Concurrency model
  - cfor to provide serializability

Our Approach: Kairos

- Provides reliable and efficient concurrent, distributed execution of a central program written in the Kairos language, an extension of C
- The Kairos compiler partitions the central program into nodecuts, which are node-level tasks executed entirely at a single node
- The Kairos runtime orchestrates the execution of successive and concurrent nodecuts, spawning them at different nodes and fetching the remote variables needed by them
- A prototype has been implemented for TinyOS

Kairos: System Design and Implementation

Kairos Compiler

- Performs dataflow analysis to partition the Kairos program into nodecuts
  - A nodecut represents a unit of work for an individual node to execute
  - Before execution must be able to determine the location of all node-local variables accessed, so as to be able to pre-fetch them
- Converts each nodecut into a nesC task
- Handles the cfor constructs correctly. Generates code to fork the thread of control into multiple concurrent executions, one per each iteration of the cfor loop
- Built as an extension to the CIL infrastructure for C analysis and transformation

Kairos Runtime

- The runtime provides the infrastructure to reliably and efficiently execute the nesC tasks generated by the compiler
- To ensure serializability of cfor, we use a hierarchical locking scheme with a novel deadlock detection protocol
- The Thread Manager controls
  - which nodecuts to next execute threads
  - whether to migrate a thread to another node
  - where to spawn the threads created by a cfor
- The Variable Manager
  - Obtains, maintains, and forwards variable locks
  - Fetches and writes back remote variables needed by a thread
  - Manages node-local variables

Results, Ongoing and Future Work

- Completing Kairos implementation on TelosB motes, and initial results are promising
- Generic Failure Recovery: Automated recovery mechanisms in presence of various classes of failures
- Exploiting Heterogeneity, Hierarchy, and User-level Energy and Resource Management