The Multikernel: A new OS architecture for scalable multicore systems
Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schupbach and Akhilesh Singhania
[SOSP ‘09]

Sameer Wadgaonkar
Dept of Computer & Information Sciences
University of Delaware
Overview

1. Introduction
2. Motivations
3. The Multikernel Model
4. Implementation: Barrelfish
5. Evaluation
6. Conclusion
• Computer hardware is changing faster than system software.
• There are many components changing at once
• Having so many changing components leads to difficulties with scaling operating systems.
• Since the hardware is changing fast, it’s better to use a general purpose design than target specific hardware components.
Introduction

- In order to adapt with changing hardware, treat the computer as networked components using OS architecture ideas from distributed systems.
- Treat the machine as a network of independent cores.
- No inter-core sharing at the lowest level.
- Move traditional OS functionality to a distributed system of processes.
- Communication done through message passing.
- Goal is to be easily scalable.
Motivations

- A general purpose Operating System must perform well on an increasingly wide range of system designs.
- Hardware has different performance characteristics.
- Unlike large high performance systems, you don’t know what hardware pieces you’ll be using so you cannot optimize for specific hardware at the design stage.
  - The Sun Niagara processor uses a banked L2 cache that a reader-writer lock can exploit by trying to write to it twice in a row to see if there are any readers.
Motivations

- Processor cores are also diverse
- Some machines have a mix of different kinds of cores.
  - Possibly with different instruction sets.

Node Layout of an 8x4-core AMD system
Motivations

• Interconnect is the term used to describe the connection between different components
• Communication between hardware components resembles a message passing network.
• There are already programmable peripherals like Network Interface Cards and Graphical Processing Units that do not maintain cache coherence with CPUs.
Motivations

- Messages cost less than shared memory
- Traditionally, most communication has been done through shared memory
- The costs grow approximately linearly with the number of threads and how many lines are modified in the cache.
- When 16 cores are modifying the same data it takes almost 12,000 extra cycles to perform the update.

Comparison of the cost of updating shared state using shared memory and message passing.
Motivations

- Cache Coherence
  - As the number of cores and the subsequent complexity of the interconnect grows, hardware cache-coherence protocols will become increasingly expensive.

- Messages are easy
  - There are two principal concerns, not being able to access shared data, and the event-driven programming style that results from asynchronous messaging.
  - The knowledge required for effective sharing of a particular data structure is encoded implicitly in its implementation.
  - OS developers are more accustomed to thinking in terms of state machines and message handlers than other programmers
  - Active research area with promising results that seem a natural fit for the Multikernel model
The Multikernel Model

- Structure the Operating System as a distributed system of cores that communicate using message passing and share no memory.
- Three Design Principles:
  - Make all inter-core communication explicit
  - Make the Operating system structure hardware-neutral
  - View state as replicated instead of shared
The Multikernel Model

- Make all inter-core communication explicit:
  - All communication is done through messages
  - Use pipelining and batching
    - Pipelining: Sending a number of requests at once
    - Batching: Bundling a number of requests into one message and processing multiple messages together

- Make the Operating System structure hardware-neutral
  - Separate the OS from the hardware as much as possible
  - Only things that are specific are:
    - Interface to hardware devices
    - Message passing mechanisms
  - This lets the OS adapt to future hardware easily.
The Multikernel Model

• View state as replicated:
  • Traditional Operating Systems maintain state
    • Windows dispatcher, Linux scheduler queue, etc.
    • Must be accessible on multiple processors
  • Information passing is usually done through shared data structures that are protected by locks
  • Instead, replicate data and update by exchanging messages
  • Replication allows data to be shared between cores that do not support the same page table format
  • Brings data closer to the cores that process it.
Implementation

• Barrelfish:
  • A research operating system built from scratch.
  • Exploring OS structure for multi-core, scalability, and hardware diversity.

• Goals:
  • Give comparable performance to existing commodity operating systems.
  • Demonstrate evidence of scalability to large numbers of cores.
  • Exploit message passing abstraction to achieve good performance.
  • Exploit the modularity of the OS to place OS functionality according to hardware topology.
  • Be retargeted and adapted for different hardware.
Figure 5: Barrelinfish structure
Implementation

- **System Structure:**
  - Put an OS instance on each core as a privileged-mode CPU driver
    - Performs authorization
    - Delivers hardware interrupts to user-space drivers
    - Time-slices user-space processes
  - Shares no data with other cores
    - Completely event driven
    - Single-threaded
    - Nonpreemptable
  - Implements an lightweight remote procedure call table

- **Monitors**
  - Single core user-space processes, schedulable
  - Performs all the inter-core coordination
  - Handles queues of messages
  - Keeps replicated data structures consistent
    - Memory allocation tables
    - Address space mappings
  - Can put the core to sleep if no work is to be done
Processes:
- Collection of dispatcher objects
  - One for each core it might execute on
- Communication is done through dispatchers
- Scheduling done by the local CPU drivers
- The dispatcher runs a user-level thread scheduler

Inter-core communication:
- Most communication done through messages
- For now it uses cache-coherent memory
  - Was the only mechanism available on their current hardware platform
Implementación

- Memory Management:
  - The operating system is distributed
  - User-level applications and system services can use shared memory across multiple cores
  - Must ensure that a user-level process can’t acquire a virtual map to a region of memory that stores a hardware page table or other OS object
  - All memory management is performed explicitly through system calls
  - All virtual memory management is performed by the user-level code
Shared Address Space:
- Barrelfish supports the traditional process model of threads sharing a single virtual address space
  - Shared across dispatchers (cores)
- Provides coordination over:
  - Virtual address space
  - Capabilities
  - Thread management
    - Thread schedulers exchange messages
      - Create and unblock threads
      - Move threads between dispatchers (cores)
    - Allows dispatchers to gang schedule threads to run together.
Evaluation

- This is just one way to use a Multikernel model
- Separating the CPU driver and monitor is not optimal
  - Doing so requires context switches to go back and forth
  - Lots of overhead
- Barrelfish is designed to be a proving ground for experimenting with non-monolithic systems with the idea of moving away from shared data as a communications system.
TLB shootdown:
- Invalidating page entries requires global coordination
- Linux and Windows:
  - The core that wants to change a page writes the operation to the shared location and sends an interrupt to all other cores that have that page.
  - Each core traps, writes to a shared variable to notify it got the change, changes the TLB, and continues execution.
  - The first core can continue when it sees that all the other cores received its change.
  - The is fast but disruptive.
Evaluation

- TLB shootdown:
  - In Barrelfish:
    - Uses messages instead of interrupts to pass on the changes.
    - Waits for a reply before continuing
      - Requires waiting, messages get passed when it's convenient
    - Allows optimization of messaging mechanism
      - Broadcast: good for AMD/Hyper-Transport which is a broadcast network
      - Unicast: good for small number of cores
      - Multicast: good for shared, on-chip L3 cache
      - NUMA-Aware Multicast
        - Send messages to highest latency nodes first
Comparison of TLB shootdown protocols
Unmap latency on 8x4-core AMD
Evaluation

- Messaging Performance: Two-Phase Commit
  - Used for changing memory ownership and usage.
  - Expensive operation, but can be pipelined to decrease cost.

Two-phase commit on 8x4-core AMD
Compute-bound workloads on 4x4-core AMD
Conclusion

- Computer hardware is changing fast
  - Multicore machines resemble complex networked systems
- Scalability is important to utilize future hardware.
- Message based communication proves to be a viable alternative.
- Barrelfish performed reasonably well in comparison to existing, mature, monolithic kernels.
Questions?
Thank You.