An Operating System for Multicore and Clouds

Mechanisms and Implementataion

David Wentzlaff, Charles Gruenwald III, Nathan Beckmann, Kevin Modzelewski, Adam Belay, Lamia Youseff, Jason Miller, Anant Agarwal (CSAIL, MIT)

SOCC 2010

Problem Statement

- Traditional OS doesn't scale well
- Current IaaS push complexity to the user (Management of VMs)
- Need for a redesigned OS to scale - effectively harness unprecedented computational power potentially provided by clouds and multi-core processors

Challenges

- Heterogeneity: Current OS requires same ISA for all cores
- Scalability: Locks, locality aliasing reliance on shared memory
- Variability of Demand: Active number of live cores instead (Space partitioning instead of time partitioning)
- Faults: Hardware (Including Performance Impact from other entities) & Software (due to programming complexity).
- Programming challenges: Lock-based code, Resource management to be done by the cloud

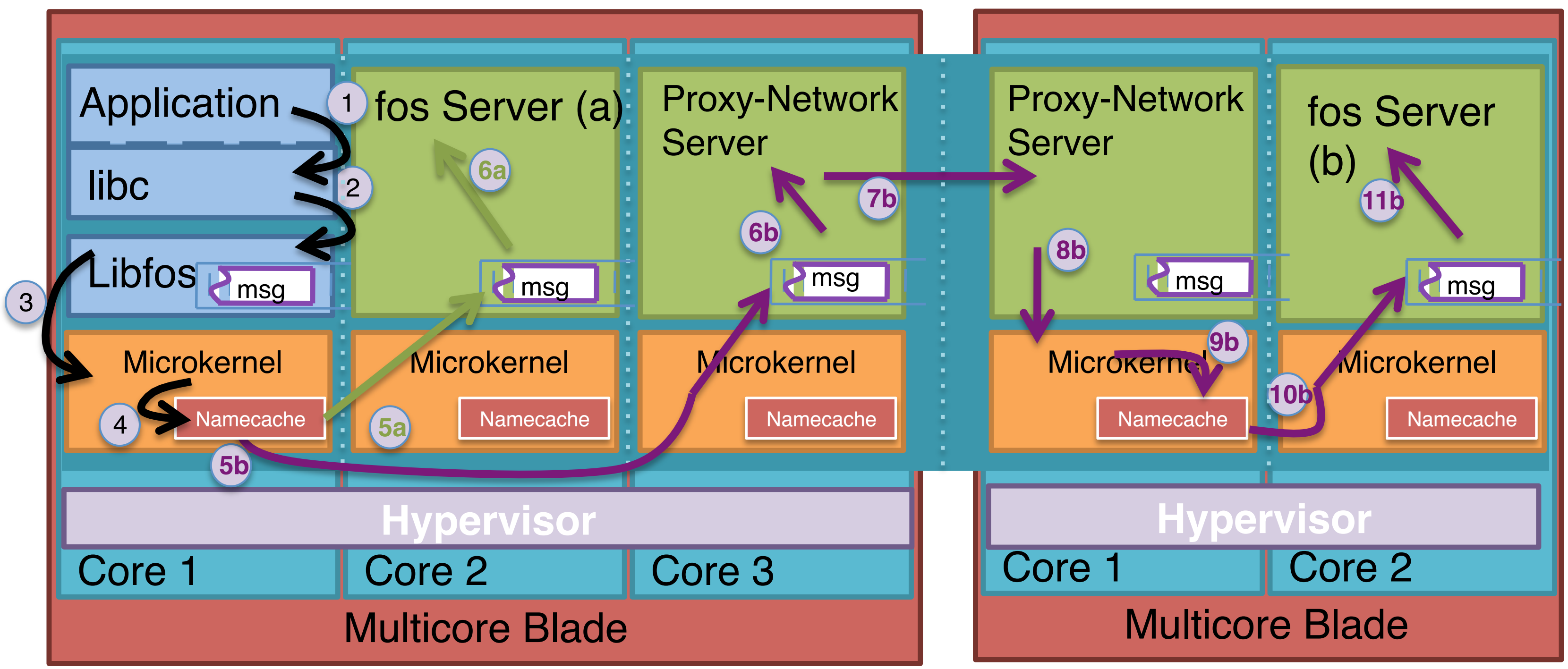
fos – Factored Operating System

- Single System Image across the cores
- MircroKernel, most of OS services provided in user space
- Function-specific services – Each one is distributed, e.g. File System services, paging service.
- Message passing – common communication primitive, across cores within the same machine and across machines

Advantages

- Ease of administration – OS Updates etc
- Transparent sharing – paging across the cloud? How efficient?
- Informed optimizations, OS has low-level knowledge, One system image
- Consistent view – load balancing, process migration
- Fault tolerance – global view

Architecture



Architecture

- Small microkernel on each core, to provide basic messaging between applications and servers, *capabilities* to restrict access into the microkernel
- Space partitioning - Belief that there will soon be a time where the number of cores in the system exceeds the number of active processes
- Name mapping kept by distributed set of proxy-network servers (Cached by each microkernel) – Leveraging P2P solutions, WIP
- Applications communicate through libfos to interact with OS services

Messaging

- Basic communication primitive
- Operating system services also implemented using messaging, need to communicate with the servers
- Messaging medium can be network or shared memory
- Intra – machine communication uses shared memory
- Across the cloud, shared memory is first used to send messages to the local proxy server which then uses the network

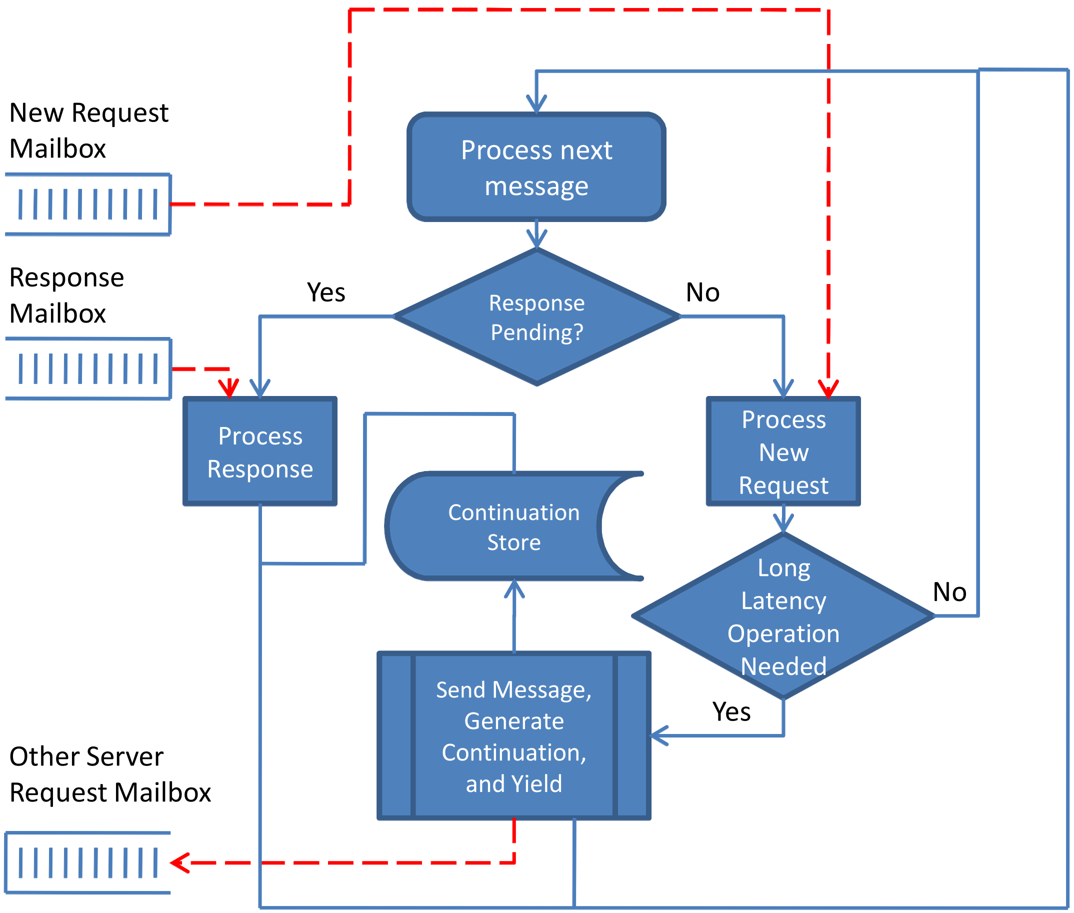
Naming

- Processes register a particular name for a mailbox
- OS service provided by several independent processes, these form a fleet, Nameserver picks a member of the fleet on receiving a request
- How this is cached, consistency etc. still in works – needs to be extremely low latency but still maintain consistent and global view of namespace

OS Services

- Fleet of servers – each service implemented as a set of processes
- File System Fleet, naming fleet, scheduling fleet, paging fleet etc.
- Uses a server model – RPC semantics (Requires serialization / deserialization primitives)
- Parallel data structures – managing the state of the service among members

Main Loop of a server



File System as a Service

- Application client, fos file system server and block device driver server – (may) execute on separate cores
- fos intercepts file system call, and sends it to the file system server (requires name server lookup)
- File system server communicates with the block device driver server to provide Disk I/O Operations and access to the physical link.

Discussion

- Is this the OS that the Datacenter needs?
  - Resource Sharing, Data Sharing
  - Programming abstraction?

- Provisioning at the granularity of VMs – not at the level of cores. Is the Hypervisor necessary? (NO). Can the single system image OS itself provide required isolation?

Discussion

- Tessellation also exploits the idea of Space-Time partitioning but designed for many-cores only. Can this scale? Seems to target multi-core systems with more emphasis on performance prediction. Hardware support?

- Microkernel vs. Multikernel (Barrelfish)
  - Make inter-core communication explicit (No single abstraction)
  - State is replicated instead of shared

Multikernel

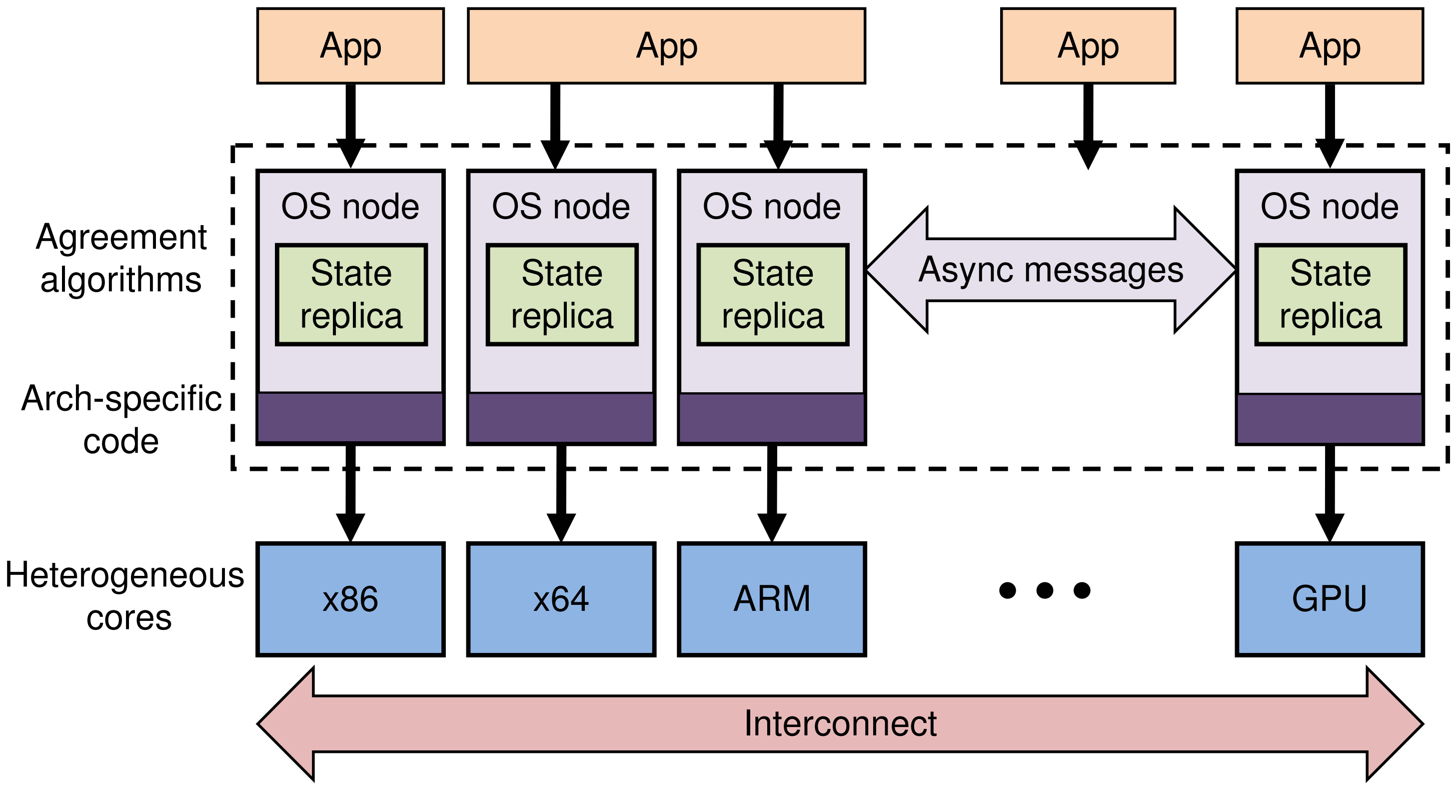


Figure 1: The multikernel model.

Discussion

- Per – Node efficiency?
  - Is efficiency really not a concern? Paging across the cloud?
  - Akaros – Many core processes (MCP) – gang scheduled
  - Present applications with finer details of the resources they are allocated (or provisioned)
  - Incremental – Data-intensive / processing intensive nodes to run Akaros while others can run general purpose OS

Thanks!

- Gautam