Self-awareness and Adaptive Technologies: the future of operating systems?

Lamia Youseff*

fos team: Nathan Beckmann, Harshad Kasture, Charles Gruenwald III, Adam Belay, David Wentzlaff**, Lamia Youseff, Jason Miller, Anant Agarwal
And e-fos team in collaboration with the Angstrom team.

* Lamia worked on fos while she was a postdoctoral associate with the Carbon group at CSAIL, MIT.
** David worked on fos while he was a Ph.D. candidate with the Carbon group at CSAIL, MIT.
Emerging Computer Architectures Provide Opportunities for OS

- **Clouds**
  - Huge computing power
  - Scalability with a “click”

- **Multicore**
  - Large number of cores
  - Very fast on-chip core-to-core communication

All logos and trademarks appearing in this presentation are the property of their respective owners.
What is Wrong with Current OSs?

- Unclear how to scale contemporary OSs on Multicore and Manycore
- Programming on an Infrastructure as a Service (IAAS), Amazon EC2 style, Cloud is a challenge
Linux Does not Scale – Page Allocation Example

- Physical Page allocation (toy example)
  - Allocate physical pages as quickly as possible
- Examine the scalability of where the time goes
  - Precision timers & Oprofile
  - 16 core: quad – quad Intel server

ON EACH CORE:
```c
void main(void)
{
  char * big_array;
  long long count;
  big_array = malloc(MEMORY_SIZE_IN_MB * MB_IN_BYTES);
  for (count = 0; count < ((MEMORY_SIZE_IN_MB * MB_IN_BYTES) / PAGE_SIZE_IN_BYTES); count++)
  {
    big_array[count * PAGE_SIZE_IN_BYTES] = count % 256;
  }
}
```
Lock Contention Dominates Runtime

Cycles (in Billions)

Number of Cores

- get_page_from_freelist
- __pagevec_lru_add_active
- __rmqueue_smallest
- free_pages_bulk
- rmqueue_bulk
- main
- page_fault
- handle_mm_fault
- other
- clear_page_c

Lock contention

Architectural overhead

Useful work
Problem with Current Day Clouds

Independent Linux VM instances cobbled together in an ad-hoc manner at application layer
OSes Need to be Rethought

- Shared memory and locks lead to poor scalability
- Non-composable due to locks – hard to modify or add services
- OS and application fight for in-core cache
- OS relies on shared memory, which does not work across a cloud
- No resilience to faults; single failure causes OS to crash

Manycore moving from a few cores to 100’s of cores
Outline

- fos Introduction
  - Vision, Structure and Architecture
  - Anatomy of a malloc()
- Scalability in fos
  - Fleets of servers
  - Example: File System Stack
- Why adaptable and self-aware OS services?
- Adaptability in fos
  - Hybrid Messaging System
  - Elastic Fleets
- Enabling Automatic Self Awareness in fos
What is fos?

A novel elastic operating system for multicores and clouds

- Scalability and reliability for future multicores with 100’s to 1000’s of cores
- Provides single-system image across cloud machines
- Dynamically grow and shrink services to meet application demands

Provides typical OS services:
- Process management, scheduling of multiple applications, virtual memory management, protection, file systems, networking

Also provides special multicore services
- Process migration, fault resilience, dynamic goal optimization, elastic resource allocation/deallocation

An Operating System as a Service (OSaaS)
Internet-inspired structure, based on **messaging**.
- OS is collection of services (e.g. name service, page service PS, file service FS)
- Each service is implemented by a **fleet** of distributed servers
- Each server is bound to a core
- Application cores message a particular server core to utilize service
- Server cores collaborate/communicate to implement needed OS service
fos Architecture

- **Microkernel**
  - Executes on all cores
  - Provides protection mechanism but not policy
  - Provides memory management mechanism, not policy

- **Naming**
  - Translates symbolic names to physical destination
  - Standard high-level API hides location
  - Provides one-to-many maps
  - Provides redirection
  - Provides resilience when a server crashes
fos Architecture

- **System Servers**
  - Internet Inspired Servers
  - Fleets: Co-operating processes that provide a service
  - Communicate only via messaging
    - Facilitates transparent communication inter-machine or intra-machine
    - Easily move server processes anywhere

- **Messaging**
  - uk level messaging for core-to-core communication.
  - Light-weight user space messaging.
  - Inter-machine vs intra-machine communication.
Anatomy of a malloc Call in fos
Anatomy of a malloc Call in fos when Server is on Remote Machine in the Cloud
Scalability in fos

- The key three design guidelines
  - Inspired by the Internet, OS is built as collection of services (e.g., naming and resource discovery, scheduler, file system)
  - Each service is fully distributed with no global shared state or locks – implemented as a collection of cooperating servers
  - OS servers are bound to cores
    - Eliminates interference between OS and application
fos Scalable Fleet example: File System Fleet
Outline

- fos Introduction
  - Vision, Structure and Architecture
  - Anatomy of a malloc()
- Scalability in fos
  - Fleets of servers
  - Example: File System Stack
- New Challenges for today’s OS
- Adaptability in fos
  - Hybrid Messaging System
  - Elastic Fleets
- Enabling Automatic Self Awareness in fos
New Challenges for today’s OS

- Unprecedented amounts of resources and variables
  - Increased number of resources => increased number of failures

- Unprecedented variability in demand
  - For the same application, its performance characteristics and requirements usually change during its runtime (e.g. allocated number of core)
  - Different simultaneously-executing applications have different performance requirements (e.g. a memory-intensive vs a computational intensive app).
Building Adaptability and Self-awareness into fos

FOS: A Self-Aware Operating System

- Analysis & Optimization Engine
- Heartbeat
- App 1
- App 2
- App 3

Observe
- Perf. Models
- Learner
- Decide
- Act

Scheduler
Memory Manager
FileSystem
Device Drivers

Core
Cache
DRAM
Disk

Heartbeat, goals
Algorithm
System call

Activity, power, temp
Voltage, freq, precision
Miss rate
Cache size, associativity

Speed
Power

I/O Devices
Building Automatic self-awareness into fos

- fos Analysis and Optimization Engine.
  - A closed feedback loop of ODA.
  - Each component provides a new OS service:
    - The Observer provides vital signs services.
    - The Decision engine provides performance models and AI learning engine.
    - The actuators changes system status.
  - Each component is either implemented as a fos fleet or integrated into another fleet.

- Analysis & Optimization Engine
  - Vital Signs (e.g. heartbeats)
  - Decision Engine (e.g. performance models, control system, AI learner)
  - Actuators
    - Applications (e.g. algorithms)
    - OS sub-systems (e.g. growing fleets)
    - Hardware components (e.g. frequency scaling)
Sefos: Building self-awareness into fos

**Vital Signs Fleet**
- Observes the status of software and hardware components.
- Provides a global knowledge-base of the system.
- Implemented as a fleet, storing the status information in a distributed data object
  - key-value store
- Collects measurements from:
  1) Applications
     - e.g. Apps heartbeats, App. specific measurements (fps in a video encoder or flops in a scientific app).
  2) OS subsystems
     - e.g. Utilization ratio of the file system fleet.
  3) Hardware components
     - e.g. temperature, core frequency, power, cache miss rate.
Sefos: Building self-awareness into fos

**Decision Engine**
- A new OS service in fos.
- Implemented through a fleet of servers for scalability.
- Process input from the vital signs service.
- Provides three approaches to runtime decision making:
  - Machine learning
  - Classical control theory
  - Performance Models

Uniform API across the mechanisms allowing them to be switched at runtime & composable.
**Actuators**

- Action that allows the system to adapt at runtime.
- Three sets of actions, based on their impact:
  - Application actions
    - Allocating or de-allocating cores to an application
    - Switching between algorithms
    - Migrating processes for better data locality and cache usage
  - OS services actions
    - Growing and shrinking fleets
    - Migrating servers
  - Hardware actions
    - Frequency scaling to save power
  - Implemented as extension to fos fleets, actions for hardware through OS tools or application-specific actions.

**Vital Signs**

- Decision Engine

**Actuators**

- Applications
- OS sub-systems
- Hardware components (e.g. frequency scaling)
Adaptability in fos

- Fos adapts to the varying demand in multicore and clouds:
  - Hybrid Messaging
  - Elastic Fleets
I. Messaging in fos

- Messaging in fos provides IPC through message-passing abstraction
  - Mailbox-based, each associated with name and capability
  - Can be implemented via a variety of mechanisms, multiplexed via *libfos* library layer
  - Transparent to the application
  - fos currently supports three mechanisms
Messaging Mechanisms in fos

- **Kernel messaging**
  - Implemented in microkernel via shared memory
  - Messages are sent by trapping into the microkernel

- **User-level messaging**
  - A channel-base mechanism via URPC
  - Runs entirely in user-space
  - Lower messaging latency at the cost of initial overhead for channel creation

- **Inter-machine messaging**
  - Message goes through the proxy server
  - Proxy server routes the message to the correct machine
Hybrid Messaging

- fos adapts automatically using various messaging mechanisms
II. Elastic Fleets

- Fleets can grow and shrink to meet varying demand
  - Fleet can grow
    - New servers are spawned on new cores
  - The naming service provides load-balancing
    - Requests directed to the new fleet member
  - Fleet can shrink
    - Load is distributed to other fleet members
Name Service enables service fleets to grow elastically with demand
fos services can grow and shrink to match demand

A synthetic benchmark app makes requests at a varying rate;
- 2 Clients, each repeatedly opens a file, reads 1 KB and closes the file
- Clients increase request rate until midpoint then decrease

fos filesystem grows from 1 to 2 servers to match demand, then shrinks as demand lessens
fos File System Service Elasticity

- fos services can grow and shrink to match demand
- A synthetic application makes requests at a varying rate
- fos filesystem grows from 1 to 2 servers to match demand, then shrinks as demand lessens
fos File System Service Elasticity

- fos services can grow and shrink to match demand
- A synthetic application makes requests at a varying rate
- fos filesystem grows from 1 to 2 servers to match demand, then shrinks as demand lessens
fos File System Service Elasticity

- fos services can grow and shrink to match demand
- A synthetic application makes requests at a varying rate
- fos filesystem grows from 1 to 2 servers to match demand, then shrinks as demand lessens
An Early Prototype of a self-aware fs service

- **Observer**: Leverage fs fleet utilization to indicate the load of the OS subsystems.
- **Decision Engine**: Use simplified *high-* and *low-watermarks*.
- **Actions**: *grow* and *shrink* the fleet

![Graph showing system throughput and fleet size over number of transactions](image)
Conclusions

- New many-cores and clouds architectures present new OS challenges for the OS community:
  - Scalability
  - Varying demand
- fos is a new factored operating system that is:
  - Addressing scalability by the fleets design
  - Addressing varying demands through the optimization and analysis engine
- We designed the analysis and optimization engine as a new OS service in the form of a close ODA loop
- We demonstrated how adaptability and self-awareness can be achieved through the analysis and optimization engine in two toy prototypes
  - Hybrid messaging
  - Elastic fs fleet
Questions

live fos web server at

http://fos.csail.mit.edu
Hidden Slides
Elastic fleets example: fos file system

- 2 Clients, each repeatedly opens a file, reads 1 KB and closes the file
- Clients increase request rate until midpoint then decrease
Anatomy of a malloc Call in fos

Diagram:
- Application
- libc
- libfos
- Paging System Server
- Proxy-Network Server
- Namecache
- Hypervisor
- Core 1
- Core 2
- Core 3
- Multicore Blade
- fos Server (b)

Steps:
1. malloc call
2. System Server
3. msg
4. Namecache
5. msg
6. Network Server
Anatomy of a malloc Call in fos when Server is on Remote Machine in the Cloud
Overview of Sefos ODA

**O:** Heartbeats services (Sensors or Observers): i.e. knowledge-base of the system.
- Two sets of APIs:
  - Collecting status updates from apps and system services
  - Responding to queries about the apps and sys services’ status

**D:** Decision engine service (decision or controllers):
- Generalized form of the scheduling services
- Deploy Control theoretic and AI models to make decisions.

**A:** Variables (Actions or Actuators); e.g. leveraging:
- Elasticity: growing and shrinking fleets.
- Spatial locality: by migrating process.
- Improving performance: by controlling the number of cores allocated to a service/app.
- Changing cores frequency: to save overall power
O: heartbeats Service

- A new OS service that keeps the current status of the system
  - App processes statistics
  - Other system fleets’ statistics (utilization level, incoming queue length, etc)
  - Hardware components status (temp, cores freq, etc)
- Implemented as a fleet of servers.
- Functionality:
  1. Collects the heartbeats from different applications and other system service,
     Via heartbeats API, e.g.
  2. Responds to inquiries about overall system status or specific application heartbeats
     Via heartbeats_monitor API, e.g.
D: Decision Engine

- Another OS service that is a generalized form of the scheduling service.
  - Can maximize overall performance, minimize power or other goals.
- Query the heartbeats service for system status.
- Make decision using:
  - A closed feedback loop to decide on best course of action to achieve a certain goal
    - Works best for changing one variable at a time
    - Model can become very difficult to understand as number of goals and number of variables increase.
  - Alternatively, an AI model
    - Works best for changing several variables at a time
A: Actions

- Each sub-system provides a set of actions (variables) and
- Three types of actions, according to their direct impact...
  - App Actions; e.g.
    - Allocating or de-allocating cores to an application
    - Migrating App processes for better spatial allocation
      - To move process to data: improve cache efficiency.
      - To decrease communication overhead (in asymmetric machines)
  - System Services Actions; e.g.
    - Leverage fleet elasticity: grow and shrink the fleet
    - Migrate servers to improve spatial locality
  - Hardware Actions; e.g.
    - Frequency scaling to save power