fos: Building a Self-Aware Operating System for Multicore and Clouds

Anant Agarwal
CSAIL, MIT

Team: David Wentzlaff, Charles Gruenwald III, Nathan Beckmann, Harshad Kasture, Kevin Modzelewski, Adam Belay, Lamia Youseff, Jason E. Miller, and Anant Agarwal

Live fos web server - http://fos.csail.mit.edu

http://groups.csail.mit.edu/carbon
The World has Changed since Multics/Unix

That was then

CPU
The World has Changed since Multics/Unix

That was then
The World has Changed since Multics/Unix

That was then
The World has Changed since Multics/Unix

That was then  

This is 2010
The World has Changed since Multics/Unix

That was then

This is 2010

1000 cores per chip in 2016
The World has Changed since Multics/Unix

That was then

1000 cores per chip in 2016

This is 2010
The World has Changed since Multics/Unix

That was then

This is 2010

1000 cores per chip in 2016
The World has Changed since Multics/Unix

That was then

This is 2010

1000 cores per chip in 2016
1,000,000 cores per cloud!

But our operating systems haven’t!
Problems with Today’s Clouds
E.g., Web Serving Cloud Environment

Web Surfing Clients

Internet

Load Balancers

Apache

Network Switches

memcached

Database

Cloud manager
E.g., Eucalyptus

Apps writers need to become systems gurus to manage very complex systems.
Factored OS: Space Sharing Replaces Time Sharing

Space sharing translates to three design principles

- Inspired by Internet, OS is collection of services
- Each service implemented as fleet of cooperating servers
  - Lockless
  - Communicate via messaging
- OS servers bound to cores

fos is also self-aware

[fos, SOCC 2010]
fos Structure

Core

User App

I/O

PS FS PS FS PS

PS PS FS PS
fos Structure

Page Server

Fleet

I/O

Core

User App

Page Server Fleet
fos Operation

Need new page (message)
fos Operation

Need new page

User App

Core

I/O
fos Operation

Need new page

User App

Core

I/O
User Application needs to access file on Processor Module 2

1. Application contacts local File System Server FS
2. FS uses Name Service to determine destination of other File System Server on Machine 2
3. File System Server messages Machine 2 via Inter-Chip Proxy Network Servers
5. File System Server returns file
fos brings standardized single system image to cloud computing.

fos provides abstraction of a single system with many processors running one OS.
Contrast with Current Day Clouds

Independent Linux VM instances cobbled together in an ad-hoc manner at application layer
Is fos a PaaS, IaaS, or Other?

- **Platform as a Service (PaaS)**
  - Application Platform provided as a service
  - Scale - up managed for one programming model

- **OS as a Service (OaaS)**
  - fos is a Cloud aware OS
  - Provide PaaS scalability to wider range of applications

- **Infrastructure as a Service (IaaS)**
  - VMs and Datacenter provided as a service
Is fos a PaaS, IaaS, or Other?

- Platform as a Service (PaaS)
  - Limited to one programming model (e.g., Hadoop)
Is fos a PaaS, IaaS, or Other?

- Platform as a Service (PaaS)
  - Limited to one programming model (e.g., Hadoop)

- Infrastructure as a Service (IaaS)
  - VMs and Datacenter provided as a service
fos is an OaaS

- Platform as a Service (PaaS)
  - Limited to one programming model (e.g., Hadoop)
- OS as a Service (OaaS)
  - fos is a Cloud *aware* OS
  - “Universal” platform
  - PaaS type scalability to wide range of applications for virtually any programming model
- Infrastructure as a Service (IaaS)
  - VMs and Datacenter provided as a service
fos Components

Name cache finds the relevant closest live server for a given service

Name caches optimize name server requests
Anatomy of a read Call
Anatomy of a read Call

File read

Application
libc
libfos

File System
Server

Paging
Server

Microkernel
Namecache

Microkernel
Namecache

Microkernel
Namecache

Hypervisor

Core 1
Core 2
Core 3

Multicore Blade
Anatomy of a read Call

File read

Application
libc
libfos
msg
File System
Server
read
Microkernel
Namecache
Microkernel
Namecache
Microkernel
Namecache
Hypervisor
Core 1
Core 2
Core 3
Multicore Blade
Anatomy of a read Call

File read

- Application
- libc
- libfos
- msg
- Microkernel
  - Namecache
- File System Server
  - read
- Paging Server
  - Microkernel
  - Namecache
- Hypervisor
- Core 1
- Core 2
- Core 3
- Multicore Blade
Anatomy of a read Call

File read

Application
libc
libfos
msg

File System Server
read

Paging Server

Microkernel
Namecache

Core 1
Core 2
Core 3
Multicore Blade

Hypervisor
Anatomy of a read Call

File read

- Application
  - libc
  - libfos
- File System Server
  - read
- Paging Server
- Microkernel
  - Namecache
- Hypervisor
- Core 1
- Core 2
- Core 3
- Multicore Blade
Anatomy of a read Call

File read

Application
libc
libfos
msg
Microkernel
Namecache
Microkernel
Namecache
Microkernel
Namecache
Hypervisor
Core 1
Core 2
Core 3
Multicore Blade
File System
Server
Paging
Server
Anatomy of a Remote read Call

File read
Anatomy of a Remote read Call

File read

Application —— libc
libfos —— msg

Microkernel —— Namecache

Hypervisor
Core 1
Core 2
Core 3

Multicore Blade

fos
Server

read

Proxy — Network Server

msg

Proxy — Network Server

File System Server

fos
Server

File read
Anatomy of a Remote read Call

File read
Anatomy of a Remote read Call

File read
Anatomy of a Remote read Call

File read
fos is Self-Aware

[Heartbeats, ICAC2010, PPoPP 2010]
Name Service (and name cache) enables service fleets to grow elastically with demand.
fos Name Service Enables Elasticity

Name Service enables service fleets to grow elastically with demand

File System Service Elasticity Example

1. Name server (and name cache) points to File Server 1
Name Service enables service fleets to grow elastically with demand.
fos Name Service Enables Elasticity

File System Service Elasticity Example

1. Name server points to File Server 1
2. Application request rate rises; File Service spawns a new File Server 2
3. Name Server directs file system accesses to the new server transparent to application

Name Service enables service fleets to grow elastically with demand
fos Name Service Enables Elasticity

File System Service Elasticity Example

1. Name server points to File Server 1
2. Application request rate rises; File Service spawns a new File Server 2
3. Name Server directs file system accesses to the new server transparent to application
4. Name server will update name caches to reflect the change

Name Service enables service fleets to grow elastically with demand
A synthetic application makes requests at a varying rate
 fos File System Service Elasticity

- filesystem with 1 server cannot meet rising demand
filesystem with 2 servers meets rising demand, but wastes resources when demand is low
fos File System Service Elasticity

- fos filesystem grows from 1 to 2 servers to match demand, then shrinks to 1 server as demand lessens
H.264 Video Encode: Procedural
H.264 Video Encode: Self-Aware using Heartbeats plus Heuristic Approach
Results

fos prototype is functional
Results on 48-core machine
Results – Syscall Time

End-to-end system call test

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Intra-Machine (us)</th>
<th>Inter-Machine (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux system call</td>
<td>0.418</td>
<td>274 (Approx.)</td>
</tr>
<tr>
<td>fos system call (via µk)</td>
<td>3.76</td>
<td>353</td>
</tr>
<tr>
<td>fos system call (via User Messaging)</td>
<td>0.628</td>
<td>353</td>
</tr>
</tbody>
</table>

Diagram showing the flow of data between different components such as Application, File System Server, Microkernel, Namecache, libc, Hypervisor, Core 1, and Core 2.
# Results – Network

## Network Benchmarks (ping, webserver)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Ping Response (us)</th>
<th>Webserver Response (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>32</td>
<td>180</td>
</tr>
<tr>
<td>fos</td>
<td>31</td>
<td>178</td>
</tr>
</tbody>
</table>
Results – Process Spawning

- Spawning a new process

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Intra-machine (ms)</th>
<th>Inter-Machine (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>fos starting new process</td>
<td>1.54</td>
<td>4.07</td>
</tr>
</tbody>
</table>
Page Allocation Fleet

Page allocator fleet

App 1

allocate

Page Allocation Server

Local page pool

App 2

allocate

Page Allocation Server

rebalance

Page allocator fleet

Process Mgmt Server

allocate

Page Allocation Server

free

Free pages

allocated pages
Results – Page Allocation

![Graph showing throughput (pages / million cycles) vs. number of clients for different configurations of servers.]

- fos: 8 servers
- fos: 4 servers
- fos: 2 servers
- fos: 1 server

Linux
Results – Page Allocation Per Core

Number of cores vs Throughput (pages / million cycles)

Linux

fos

Num. servers in best configuration
Results – File System

OS requests to closest server versus to random server

- Throughput (trans / million cycles)
- Performance Improvement (%)
- Number of clients

Graph showing the throughput and performance improvement as the number of clients increases.
EE meets CS
A control theory for self-aware systems
fos is Self-Aware

Heartbeats API

fos: A Self-Aware Operating System

Analysis & Optimization Engine

Observe

 Decide

Act

Scheduler
Memory Manager
File System
Device Drivers

App 1

App 2

App 3

Perf, Models

Learner

Cache size, associativity

System call

Scheduler

Memory Manager

File System

Device Drivers

Core

DRAM

Disk

I/O

PEP Core

Activity, power, temp

Voltage, freq, precision

Miss rate

Cache

Core

App 2

App 1

App 3

Algorithm

Heartbeat, goals

System call

Core

DRAM

Disk

I/O

PEP Core

Speed

Power
ODA Control Loop

[Control Theory for Self-Aware Computing, ICDC2010]

Observe
Heartbeats API

Decide
Alternatives:
- Control system
- Learner
- Heuristic models

Act
System Parameters
- Cores
- Memory
- Power
A Control Theory for Self-Aware Computing

Desired Heart Rate $\bar{r}$

Error $e(k)$

SEEC Controller

Speedup $s(k)$

Application

Observed Heart Rate $r(k)$
User Can Dial in Chosen Behavior