The **nom** Profit-Maximizing Operating System

Muli Ben-Yehuda, Orna Agmon Ben-Yehuda, Dan Tsafrir

Lightbits Labs

Technion—Israel Institute of Technology
This talk is about money
In the clouds
It all began a few years ago, ...
3 years on average: buying hardware
Months: web hosting
Hours: EC2 on-demand (pay-as-you-go)
5 minutes: EC2 Spot Instances (pay-as-you-go)
3 minutes: GridSpot
1 minute: Profitbricks, Google Compute Engine
...
Amazon allows clients to dynamically add and remove I/O devices
Amazon allows clients to set a desired I/O rate on a per-block-instance basis
GridSpot, ProfitBricks and CloudSigma let their clients compose a flexible bundle—with prices depending on current cost of resources
The Only Cloud Service Provider with a Price/Performance Guarantee

ProfitBricks offers an easy-to-understand pricing model and minute-based billing and it's based on 4 individual items:
- CPU Cores + RAM + Storage + Traffic

ProfitBricks Cloud Price/Performance Guarantee:
We guarantee that any workload deployed on ProfitBricks will cost less than the same workloads running at the same performance level on the IaaS platforms of Amazon, Google or Microsoft.

Plus, All ProfitBricks Cloud Computing IaaS Accounts Include:
- ✔️ Free 24/7 Phone & Email Support
- ✔️ Free Inbound Traffic
- ✔️ Free IOPS
- ✔️ Free Firewalls

Ready to Experience Cloud Computing 2.0?  |  Go to 14 DAY TRIAL
RaaS leads to the need in automatic economic mechanisms inside the machine, and it’s happening today: CloudSigma’s burst pricing
CloudSigma’s burst prices, a typical day in July 2012

Image by Kristof Kovacs, taken from
http://kkovacs.eu/cloudsigma-burst-price-chart
CloudSigma’s burst CPU prices, May 2015–Jan 2016
Dynamic pricing is great...
Except for the fly in the ointment
Your operating system

SUCKS.
For traditional operating systems, performance is everything
Costs? They don’t care about no costs
Dynamic pricing clouds mandate change

Resource ownership and control  the OS is no longer sole owner
Economic model  Cloud provider vs. various clients
Resource granularity  fine-grained resource allocation
Architectural support  physical and virtual hardware
A profit maximizing operating system
Maximize profit  optimize for both performance and cost
Expose resources  move the kernel out of the way
Isolate applications  . . . with a little help from hardware friends
nom
Design and Implementation
Networking: private application stacks

Applications run with private network stacks and device drivers.

**nom** provides default TCP/IP stack and a device driver for the virtio virtnet virtual device.

Applications configure the stack and driver independently.

Potentially: application-specific optimizations to stacks and drivers.

Applications choose polling vs. interrupt-driven operations: trade off CPU cycles and power for better performance.
Application controls network stack and device driver behavior on receive and transmit via the batching delay

Batching delay 0: no delay, every packet is run to completion

Batching delay \( W \mu\text{sec} \): batch packets together but don’t let any packet wait more than \( W \mu\text{sec} \)

Balancing throughput, latency, and jitter

Throughput increases with \( W \) (good) up to a point

Latency and jitter increase with \( W \) (bad!)

What’s the right batching delay? Depends!
Utility of network bandwidth = Income - Expenses

Goal: Maximize the profit of a server running in the guest OS.
Income: depends on its performance and client SLA.
Expenses: depend on its resource consumption and on current prices.
Given: price of bandwidth and application load
Means: throughput, latency, and jitter
**Assumptions on Utility**

Assumption: network server’s utility **increases** as throughput rises

1. Can serve more users
2. Can serve users faster

Assumption: server’s utility **decreases** as latency and jitter rise

1. Lower quality of service
2. More likely to violate SLO’s

Assumption: server’s utility **decreases** as the price of bandwidth rises

1. Bandwidth becomes more expensive
2. Expenses rise $\Rightarrow$ lower profits
The bonus utility function

\[ U_{\text{bonus}} = T \cdot (\alpha + \frac{\gamma}{L} - P) \]

Key idea: users give the server bonuses for better latency

- \( T \): throughput in \( \frac{\text{Gbit}}{s} \) or application operations/second
- \( \alpha \): server’s valuation of useful bandwidth (\$/Gbit or \$/operation)
- \( \gamma \): size of the bonus
- \( P \): bandwidth price in \$/Gbit or \$/operation
The refund utility function

\[ U_{\text{refund}} = T \cdot (\max(0, \alpha - \beta \cdot L) - P) \]

Key idea: the server gives its users refunds as latency increases

- \( T \) throughput in \( \frac{\text{Gbit}}{s} \) or application operations/second
- \( \alpha \) server’s valuation of useful bandwidth (\$/Gbit or \$/operation)
- \( \beta \) extent of the refund
- \( P \) bandwidth price in \$/Gbit or \$/operation
The penalty utility function

\[ U_{\text{penalty}} = T \cdot (\alpha \cdot (1 - \min(1, X \cdot \mathcal{N}(L_0, L, \sigma)))) - P \]

Key idea: server pays penalties when it fails to meet SLAs

- \( T \) throughput in \( \text{Gbit/s} \) or application operations/second
- \( \alpha \) server’s valuation of useful bandwidth (\$/Gbit or \$/operation)
- \( X \) penalty factor for not meeting users’ SLOs
- \( L \) mean latency (\( \mu \text{secs} \))
- \( L_0 \) the maximal latency allowed by the SLAs
- \( \sigma \) the latency’s standard deviation (jitter)
- \( \mathcal{N}(L_0, L, \sigma) \) the probability that a latency sample will not meet the latency SLO
- \( P \) bandwidth price in \$/Gbit or \$/operation
Implementation

- x86-64 SMP virtual machines
- Direct access to I/O devices
- Runs on bare-metal too
- C and (a little) assembly
- Applications:
  - memcached key-value store
  - nhttpd web server
  - NetPIPE network benchmark
- Penalty, refund, and bonus utility
Evaluation

OUTSTANDING
Excellent
Very Good
Average
Below Average
Methodology: key questions

1. Does optimizing for cost preclude optimizing for performance?
2. Does optimizing for both cost and performance improve profit?
3. Is changing behavior at runtime important for maximizing profits?
**Methodology: experiment design**

- memcached, nhttpd, and NetPIPE running in Linux, OSv, and nom virtual machines with the same* resources
- Each experiment runs for 120 seconds
- First 60 seconds:
  - Serving many users together
  - Cloud provider charges $1/Gb for bandwidth
- Second 60 seconds:
  - Serving a single important user at a time
  - Cloud provider charges $10/Gb for bandwidth
- Results reported are averages of 5 runs
- nom applications know throughput, latency, jitter as a function of batching delay and change it when conditions change
Effect of batching on throughput and latency

Figure: memcached throughput and latency as a function of batching delay

1. Throughput rises with batching delay up to a point
2. Latency is best with batching delay 0 (no delay)
Figure: The Memcached throughput/latency Pareto frontier

1. No single “best” batching delay setting
2. \( \Rightarrow \) No single “best” stack or driver
3. \( \Rightarrow \) Applications can pick the right working point according to current price and load
How well does nom perform?

Up to 1.29x better throughput and latency than Linux

1.2x–3.9x better throughput and up to 9.1x better latency than Linux and OSv
What makes nom fast?

1. Hypervisor friendly: 10K-20K exits vs. 43K-90K for Linux and OSv
2. Polling: 1K interrupts vs. 12K-22K for Linux and OSv
3. Zero-copy transmit and receive, no allocations on data path
4. Application-specific configuration of stacks and drivers
Does nom profit more?

**memcached:** nom yields up to 11x more profit

**nhttpd:** nom yields up to 1.8x more profit
What makes **nom** profitable?

**memcached** static vs. adaptive

**nhttpd** static vs. adaptive

- Runtime adaptation is a must for maximizing profit
Conclusions

Clouds with dynamic pricing present challenges and opportunities.

Traditional OSes are a poor fit for such clouds because they ignore costs.

We present nom, a profit-maximizing operating system.

In nom there is no “best” software stack:

Application-specific stacks and drivers

Use utility functions to adapt runtime behavior and maximize profit.
Thank you! Questions?
Related work

- Userspace I/O and virtual machine device assignment
- The Exokernel [SOSP’95, SOSP’97, TOCS’02]
- Library operating systems [VEE’07, ASPLOS’11]
- Dune [OSDI’12] and IX [OSDI’14]
- Mirage [HotCloud’10], [ASPLOS’13]
- NoHype [ISCA’10]
- Arrakis [HotOS’13], [OSDI’14]
- RaaS [HotCloud’12], [VEE’14], [CACM]