CS162
Operating Systems and Systems Programming
Lecture 27

ManyCore OS and Peer-to-peer Systems

December 10, 2007

Prof. John Kubiatowicz

http://inst.eecs.berkeley.edu/~cs162

Requests for Final topics

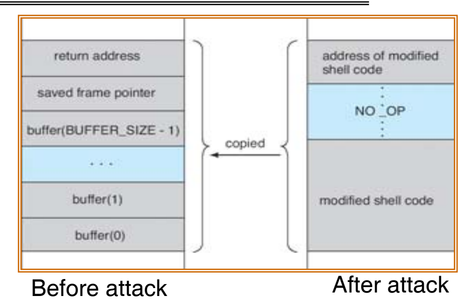
- Some topics people requested:
  - Dragons: too big of a topic for today
  - ManyCore Systems
  - Parallel OSs
  - Embedded OSs
  - Peer-to-Peer Systems (OceanStore)
  - Virtual reality/enhancement
  - Quantum Computing
- Today:
  - A couple of topics to finish from last time
  - ManyCore/Parallel OS
  - Embedded OS (realtime systems)
  - Peer-to-Peer Systems (OceanStore)
- Other Topics:
  - Come look for me at office hours (Or any other time)

Security Terms

- Virus:
  - A piece of code that attaches itself to a program or file so it can spread from one computer to another, leaving infections as it travels
  - Most attached to executable files, so don't get activated until the file is actually executed
  - Once caught, can hide in boot tracks, other files, OS
- Worm:
  - Similar to a virus, but capable of traveling on its own
  - Takes advantage of file or information transport features
  - Because it can replicate itself, your computer might send out hundreds or thousands of copies of itself
- Trojan Horse:
  - Named after huge wooden horse in Greek mythology given as gift to enemy; contained army inside
  - At first glance appears to be useful software but does damage once installed or run on your computer

Security Problems: Buffer-overflow Condition

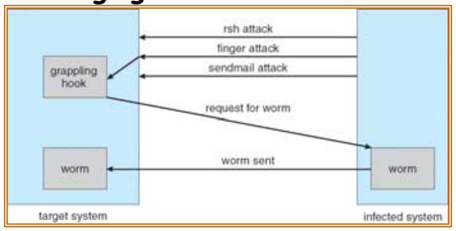
```
#define BUFFER SIZE 256
int process(int argc,
            char *argv[])
{
  char buffer[BUFFER SIZE];
  if (argc < 2)
      return -1;
  else {
      strcpy(buffer,argv[1]);
      return 0;
  }
}
```



- Technique exploited by many network attacks
  - Anytime input comes from network request and is not checked for size
  - Allows execution of code with same privileges as running program – but happens without any action from user!
- How to prevent?
  - Don't code this way! (ok, wishful thinking)
  - New mode bits in Intel, Amd, and Sun processors
    - » Put in page table; says "don't execute code in this page"

The Morris Internet Worm: The beginning of chaos

- Internet worm (Self-reproducing)
  - Author Robert Morris, a first-year Cornell grad student
  - Launched close of Workday on November 2, 1988
  - Within a few hours of release, it consumed resources to the point of bringing down infected machines



- Techniques
  - Exploited UNIX networking features (remote access)
  - Bugs in *finger* (buffer overflow) and *sendmail* programs (debug mode allowed remote login)
  - Dictionary lookup-based password cracking
  - Grappling hook program uploaded main worm program

Timing Attacks: Tenex Password Checking

- Tenex – early 70's, BBN
  - Most popular system at universities before UNIX
  - Thought to be very secure, gave "red team" all the source code and documentation (want code to be publicly available, as in UNIX)
  - In 48 hours, they figured out how to get every password in the system
- Here's the code for the password check:

```
for (i = 0; i < 8; i++)
  if (userPasswd[i] != realPasswd[i])
    go to error
```

- How many combinations of passwords?
  - 256^8?
  - Wrong!

Defeating Password Checking

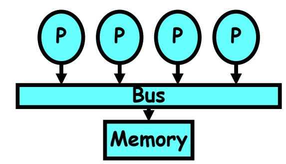
- Tenex used VM, and it interacts badly with the above code
  - Key idea: force page faults at inopportune times to break passwords quickly
- Arrange 1st char in string to be last char in pg, rest on next pg
  - Then arrange for pg with 1st char to be in memory, and rest to be on disk (e.g., ref lots of other pgs, then ref 1st page)

```
                a|aaaaaa
                 |
page in memory| page on disk
```

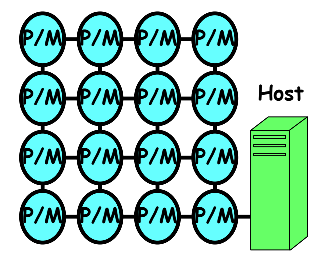
- Time password check to determine if first character is correct!
  - If fast, 1st char is wrong
  - If slow, 1st char is right, pg fault, one of the others wrong
  - So try all first characters, until one is slow
  - Repeat with first two characters in memory, rest on disk
- Only 256 * 8 attempts to crack passwords
  - Fix is easy, don't stop until you look at all the characters

Types of Parallel Machines

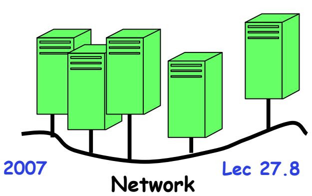
- Symmetric Multiprocessor
  - Multiple processors in box with shared memory communication
  - Current MultiCore chips like this
  - Every processor runs copy of OS



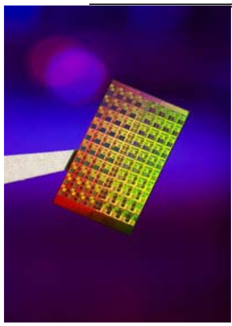
- Non-uniform shared-memory with separate I/O through host
  - Multiple processors
    - » Each with local memory
    - » general scalable network
  - Extremely light "OS" on node provides simple services
    - » Scheduling/synchronization
  - Network-accessible host for I/O



- Cluster
  - Many independent machine connected with general network
  - Communication through messages



ManyCore Chips: The future is on the way



- Intel 80-core multicore chip (Feb 2007)
  - 80 simple cores
  - Two floating point engines /core
  - Mesh-like "network-on-a-chip"
  - 100 million transistors
  - 65nm feature size

Frequency	Voltage	Power	Bandwidth	Performance
3.16 GHz	0.95 V	62W	1.62 Terabits/s	1.01 Teraflops
5.1 GHz	1.2 V	175W	2.61 Terabits/s	1.63 Teraflops
5.7 GHz	1.35 V	265W	2.92 Terabits/s	1.81 Teraflops

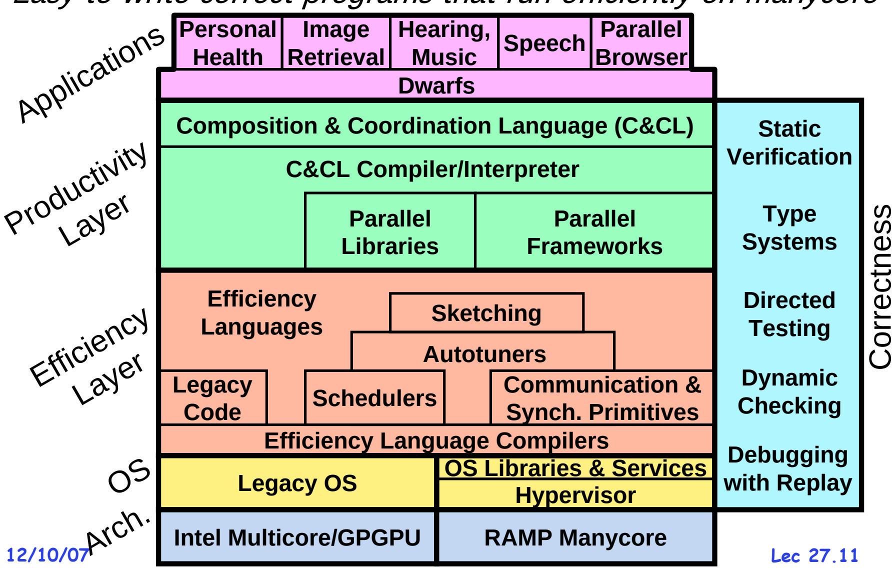
- "ManyCore" refers to many processors/chip
  - 64? 128? Hard to say exact boundary
- How to program these?
  - Use 2 CPUs for video/audio
  - Use 1 for word processor, 1 for browser
  - 76 for virus checking???
- Something new is clearly needed here…

Berkeley PARLab

- Parallel processors have been around for a long time
  - So, what is different now?
    - » Industry is on a growth path – massively parallel processors will soon be widespread
    - » Communication between cores very low overhead
  - Challenge is still how to program them
- Caught attention of Berkeley (and many others)
  - New research laboratory: PARLab
  - New approach: vertically integrated programming environment
  - Combine lessons of last 20 years with application-driven approach
- Berkeley researchers from many backgrounds meeting since Feb. 2005 to discuss parallelism
  - Krste Asanovic, Ras Bodik, Jim Demmel, Kurt Keutzer, John Kubiatowicz, Edward Lee, George Necula, Dave Patterson, Koushik Sen, John Shalf, John Wawrzynek, Kathy Yelick, …
  - Circuit design, computer architecture, massively parallel computing, computer-aided design, embedded hardware and software, programming languages, compilers, scientific programming, and numerical analysis

Par Lab Research Overview

Easy to write correct programs that run efficiently on manycore



PARLab approach to parallel programming

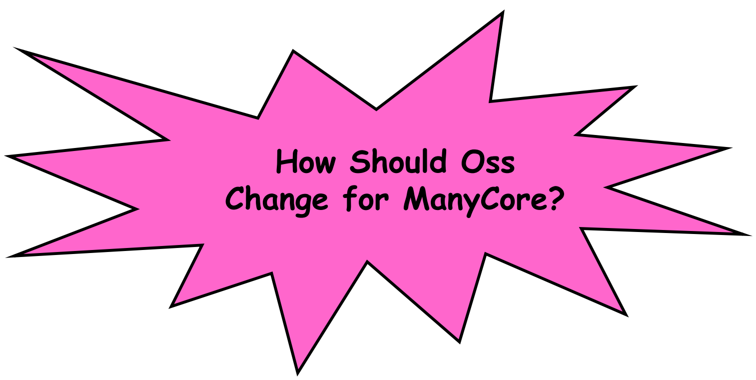
- 2 types of programmers ⇒ 2 layers
- Efficiency Layer (10% of today's programmers)
  - Expert programmers build Frameworks & libraries, Hypervisors, …
  - "Bare metal" efficiency possible at Efficiency Layer
- Productivity Layer (90% of today's programmers)
  - Domain experts / Naïve programmers productively build parallel apps using frameworks & libraries
  - *Frameworks & libraries* composed to form applications
- Effective composition techniques allows the efficiency programmers to be highly leveraged ⇒
  - Create language for Composition and Coordination (C&C)

Traditional Parallel OS

- Job of OS is support and protect
  - Need to stay out of way of application
- Traditional single-threaded OS
  - Only one thread active inside kernel at a time
    - » One exception – interrupt handlers
    - » Does not mean that that there aren't many threads – just that all but one of them are asleep or in user-space
    - » Easiest to think about – no problems introduced by sharing
  - Easy to enforce if only one processor (with single core)
    - » Never context switch when thread is in middle of system call
    - » Always disable interrupts when dangerous
  - Didn't get in way of performance, since only one task could actually happen simultaneously anyway
- Problem with Parallel OSs: code base already very large by time that parallel processing hit mainstream
  - Lots of code that couldn't deal with multiple simultaneous threads ⇒One or two locks for whole system

Some Tricky Things about Parallel OSs

- How to get truly multithreaded kernel?
  - More things happening simultaneously⇒need for:
    - » Synchronization: thread-safe queues, critical sections, …
    - » Reentrant Code – code that can have multiple threads executing in it at the same time
    - » Removal of global variables – since multiple threads may need a variable at the same time
  - Potential for greater performance⇒need for:
    - » Splitting kernel tasks into pieces
- Very labor intensive process of parallelizing kernel
  - Needed to rewrite major portions of kernel with finer-grained locks
    - » Shared among multiple threads on multiple processors⇒ Must satisfy multiple parallel requests
    - » Bottlenecks (coarse-grained locks) in resource allocation can kill all performance
- Truly multithreaded mainstream kernels are recent:
  - Linux 2.6, Windows XP, …

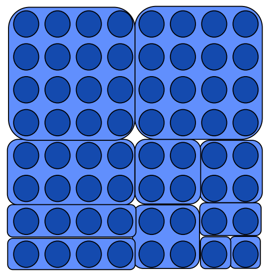


ManyCore opportunities: Rethink the Sink

- Computing Resources are *not* Limited
  - High Utilization of *every* core unnecessary
  - Partition *Spatially* rather than *Temporally*
- Protection domains not necessarily heavyweight
  - Spatial Partitioning⇒ protection crossing as simple as sending a message from partition to partition
  - Opportunity: hardware support for label-based access control (ala Asbestos) for messages
- I/O devices *not* limited and *do not need to be* heavily multiplexed
  - High bandwidth devices available through network
  - FLASH or other persistent storage yields fast, flat hierarchy
  - Monolithic file system view outdated: give applications access to persistent chunks of storage
  - Allocate cores for I/O – yields performance *and* security

Spatial Partitioning

- Groups of processors acting within hardware boundary
  - Shared memory and/or active messages within partition
  - Protected message passing between partitions
  - Time multiplexing of computing resources *not* required
  - Quality of Service guarantees provided on resources such as memory and network bandwidth
- Deconstructed OS
  - Only hypervisor present on every partition
  - Functionality of traditional OS split amongst partitions:
    - » Legacy Device drivers wrapped and isolated on individual partitions
    - » File systems handled by server partitions
    - » Interrupts and other events delivered to free partitions
  - Parallel applications given "bare metal"
    - » free to deploy whatever scheduling is most advantageous



Spatial Partitioning and Applications

- Many possibilities for mapping applications to partitions:
  - *Within partition*: shared memory and user-level active messages freely exchanged for parallel apps
  - *Between partitions:* user-level active messages
- Since spatial partitions represent security contexts:
  - One application per partition
    - » Obvious division
  - Many partitions per applications:
    - » Great for pipe/filter type of computations
    - » Insecure plug-ins isolated from primary application
    - » Communication between partitions via messages
- Should spatial partitions be virtualized?
  - Probably
    - » Danger of reintroducing scheduling artifacts, but….
    - » Gives more flexibility for dividing up applications

Minimalism

- Hypervisor is only universally resident code
  - Handles basic resource allocation
    - » Very thin layer
  - Manages spatial partitions/initiating application execution
- Major system facilities replaced by libraries/servers
  - Thread generation and scheduling ⇒ *user-level libraries*
  - I/O system calls ⇒ messages to servers on other cores
  - Servers for filesystems/etc run at user level as well
- "Bare-Metal" partitions for applications
  - Parallel apps given complete control of processor partition
    - » *User-level* runtime scheduling system
    - » Exclusive use of partition-wide synchronization network
    - » Exclusive use of shared memory, virtual memory hardware
    - » Direct access to performance monitoring hardware
  - Any temporal multiplexing is *infrequent* and *partition-wide*

User-Level Protected Messaging

- *Crossing protection domain ⇒ sending a message*
- User given direct ability to send and receive messages
  - Direct, protected access to network interface
  - Message send/receive simply writing/reading registers
  - Access to DMA also at user level
- User-level Messages for crossing protection domains
  - Rather than a two-level hierarchy (user+root), have a partially ordered set of Security contexts
  - Taint tracking
    - » Partitions and/or processes labeled with security contexts
    - » Data from one source is "tainted" with label from source
    - » Message dropped if dest not authorized to receive it
    - » Example: data from partition with label X cannot leak to any other partition unless has appropriate label
- Messages can invoke handlers on receiver in hardware
  - Full support for fast exception handling
    - » Exceptions handled entirely at user level

Fault Isolation and Optimistic Concurrency

- Mechanisms for Optimistic Concurrency
  - Partition-level checkpoint/restore
  - Permits ability to back up to consistent point across ManyCore partitions
- Dependency tracking
  - Track which speculative executions depend on each other
  - Dependencies can be transferred through messages
  - Speculative rollback of groups of dependent executions
    - » Example: Transaction-based cached file system; simply roll-back application if cache discovered out of date
- Fault-Tolerance
  - Checkpoint/restore triggered via information from compiler/frameworks
  - Idea: framework knows when to
    - » Trigger checkpoints
    - » When and how to check consistency of computation

Realtime OS/Embedded Applications

- Embedded applications:
  - Limited Hardware
  - Dedicated to some particular task
  - Examples: 50-100 CPUs in modern car!
- What does it mean to be "Realtime"?
  - Meeting time-related goals in the real world
    - » For instance: to show video, need to display X frames/sec
  - Hard real-time task:
    - » one which we must meet its deadline
    - » otherwise, fatal damage or error will occur.
  - Soft real-time task:
    - » one which we should meet its deadline, but not mandatory.
    - » We should schedule it even if the deadline
  - Determinism:
    - » Sometimes, deterministic behavior is more important than high performance

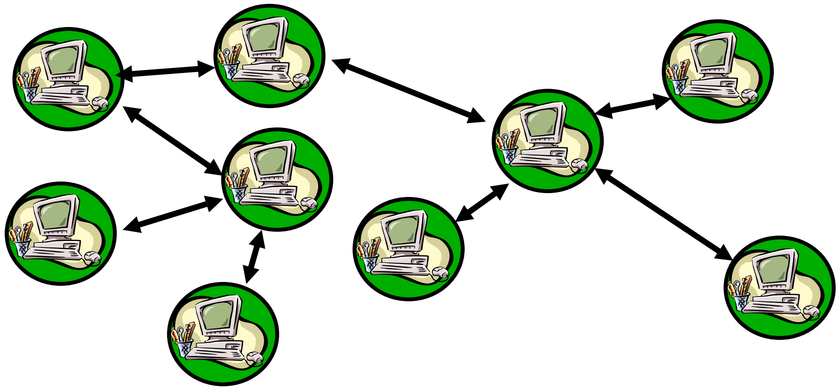
MultiCore and Realtime

- Realtime OS Details
  - Realtime scheduler looks at deadlines to decide who to schedule next
    - » Example: schedule the thread whose deadline is next
  - What makes it hard to perform realtime scheduling:
    - » Too many background tasks
    - » Optimizing for overall responsiveness or throughput is different from meeting explicit deadlines
- Why are Realtime apps often handled by embedded processors?
  - Because they are dedicated and more predictable
  - Idea: Only need to meet throughput requirements
    - » Might as well slow down processor (via lower voltage) as long as performance criteria met
    - » Power reduces as V^2!
- ManyCore
  - Opportunity to devote cores to realtime activities
  - "Bare metal" partitions: best of realtime and general Oss in one chip…!

Administrivia

- Midterm II
  - Still Grading! Should be done very soon.
  - I put up solutions already
- Project 4
  - Due Tomorrow, 12/11
- Final Exam
  - December 17th, 5:00-8:00pm
  - 10 Evans
  - Bring 2 sheets of notes, double-sided
  - All lectures – except today (this is a freebie!)

Peer-to-Peer: Fully equivalent components



- Peer-to-Peer has many interacting components
  - View system as a set of equivalent nodes
    - » "All nodes are created equal"
  - Any structure on system must be self-organizing
    - » Not based on physical characteristics, location, or ownership

Research Community View of Peer-to-Peer

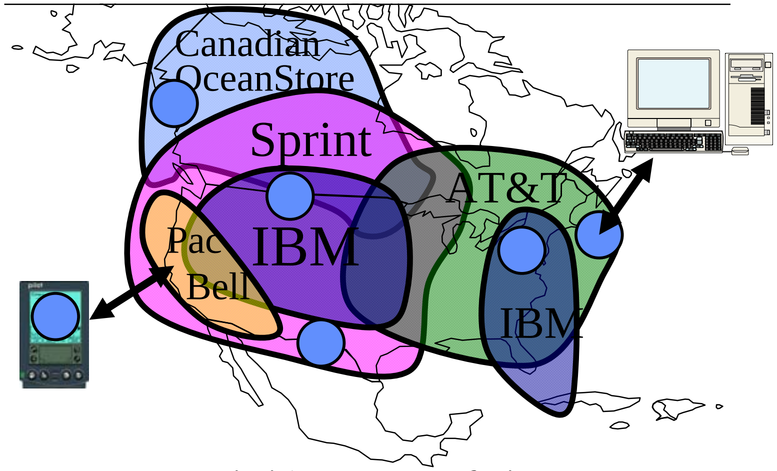
- Old View:
  - A bunch of flakey high-school students stealing music
- New View:
  - A philosophy of systems design at extreme scale
  - Probabilistic design when it is appropriate
  - New techniques aimed at unreliable components
  - A rethinking (and recasting) of distributed algorithms
  - Use of Physical, Biological, and Game-Theoretic techniques to achieve guarantees

Why the hype???

- File Sharing: Napster (+Gnutella, KaZaa, etc)
  - Is this peer-to-peer? Hard to say.
  - Suddenly people could contribute to active global network
    - » High coolness factor
  - Served a high-demand niche: online jukebox
- Anonymity/Privacy/Anarchy: FreeNet, Publis, etc
  - Libertarian dream of freedom from the man
    - » (ISPs? Other 3-letter agencies)
  - Extremely valid concern of Censorship/Privacy
  - In search of copyright violators, RIAA challenging rights to privacy
- Computing: The Grid
  - Scavenge numerous free cycles of the world to do work
  - Seti@Home most visible version of this
- Management: Businesses
  - Businesses have discovered extreme distributed computing
  - Does P2P mean "self-configuring" from equivalent resources?
  - Bound up in "Autonomic Computing Initiative"?

OceanStore

Utility-based Infrastructure



- Data service provided by storage federation
- Cross-administrative domain
- Contractual Quality of Service ("someone to sue")

OceanStore: Everyone's Data, One Big Utility

"The data is just out there"

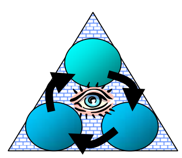
- How many files in the OceanStore?
  - Assume 10^{10} people in world
  - Say 10,000 files/person (very conservative?)
  - So 10^{14} files in OceanStore!
  - If 1 gig files (ok, a stretch), get 1 mole of bytes! (or a Yotta-Byte if you are a computer person)

Truly impressive number of elements…
… but small relative to physical constants

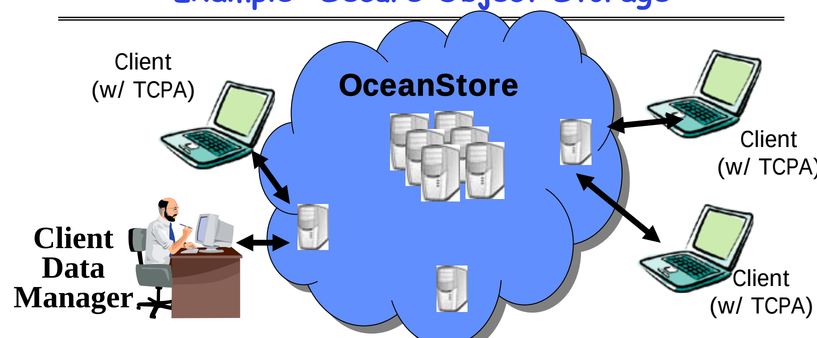
Aside: SIMS school: 1.5 Exabytes/year (1.5×10^{18}) back in 2001….!

Key Observation: Want Automatic Maintenance

- Can't possibly manage billions of servers by hand!
- System should automatically:
  - Adapt to failure
  - Exclude malicious elements
  - Repair itself
  - Incorporate new elements
- System should be secure and private
  - Encryption, authentication
- System should preserve data over the long term (*accessible* for 1000 years):
  - Geographic distribution of information
  - New servers added from time to time
  - Old servers removed from time to time
  - Everything just works



Example: Secure Object Storage



- Security: Access and Content controlled by client
  - Privacy through data encryption
  - Optional use of cryptographic hardware for revocation
  - Authenticity through hashing and active integrity checking
- Flexible self-management and optimization:
  - Performance and durability
  - Efficient sharing

OceanStore Assumptions

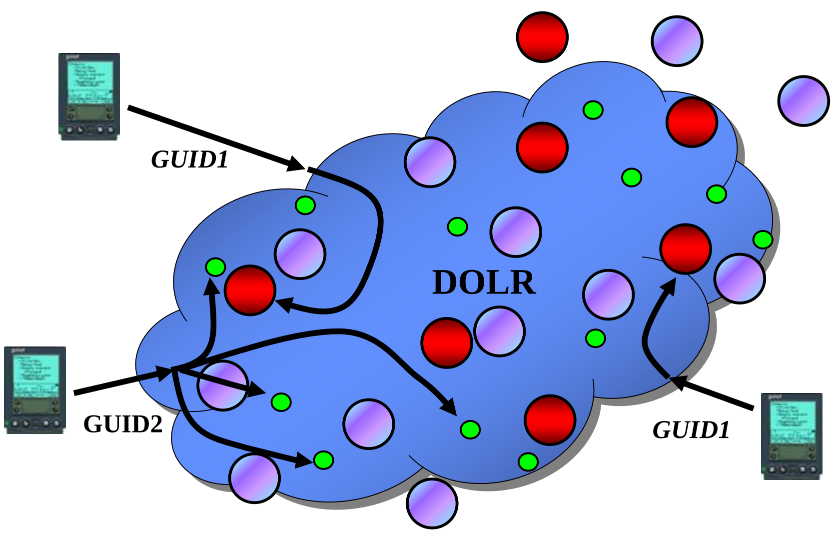
Peer-to-peer

- Untrusted Infrastructure:
  - The OceanStore is comprised of untrusted components
  - Individual hardware has finite lifetimes
  - All data encrypted within the infrastructure
- Mostly Well-Connected:
  - Data producers and consumers are connected to a high-bandwidth network most of the time
  - Exploit multicast for quicker consistency when possible
- Promiscuous Caching:
  - Data may be cached anywhere, anytime

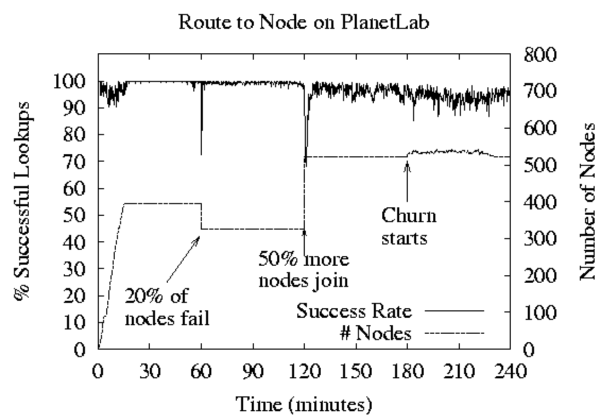
Quality-of-Service

- Responsible Party:
  - Some organization (*i.e. service provider)* guarantees that your data is consistent and durable
  - Not trusted with *content* of data, merely its *integrity*

Peer-to-Peer in OceanStore: DOLR
(Decentralized Object Location and Routing)



Stability under extreme circumstances

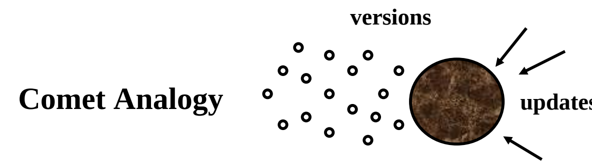


(May 2003: 1.5 TB over 4 hours)

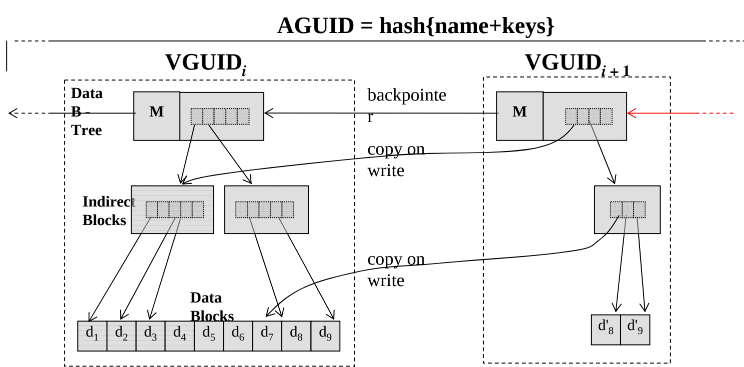
DOLR Model generalizes to many simultaneous apps

OceanStore Data Model

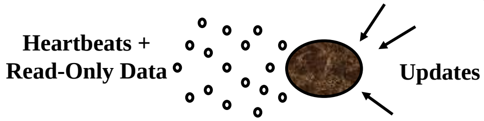
- Versioned Objects
  - Every update generates a new version
  - Can always go back in time (Time Travel)
- Each Version is Read-Only
  - Can have permanent name
  - Much easier to repair
- An Object is a signed mapping between permanent name and latest version
  - Write access control/integrity involves managing these mappings



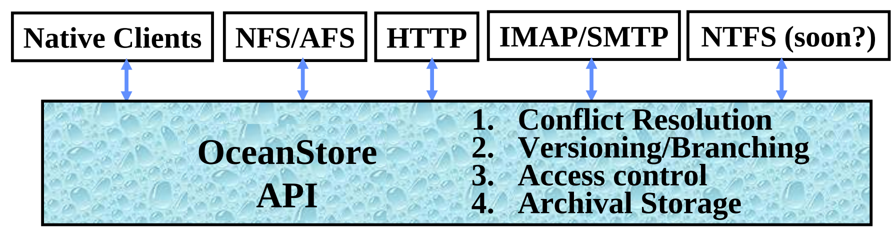
Self-Verifying Objects



♥Heartbeat: $\{AGUID, VGUID, Timestamp\}_{signed}$



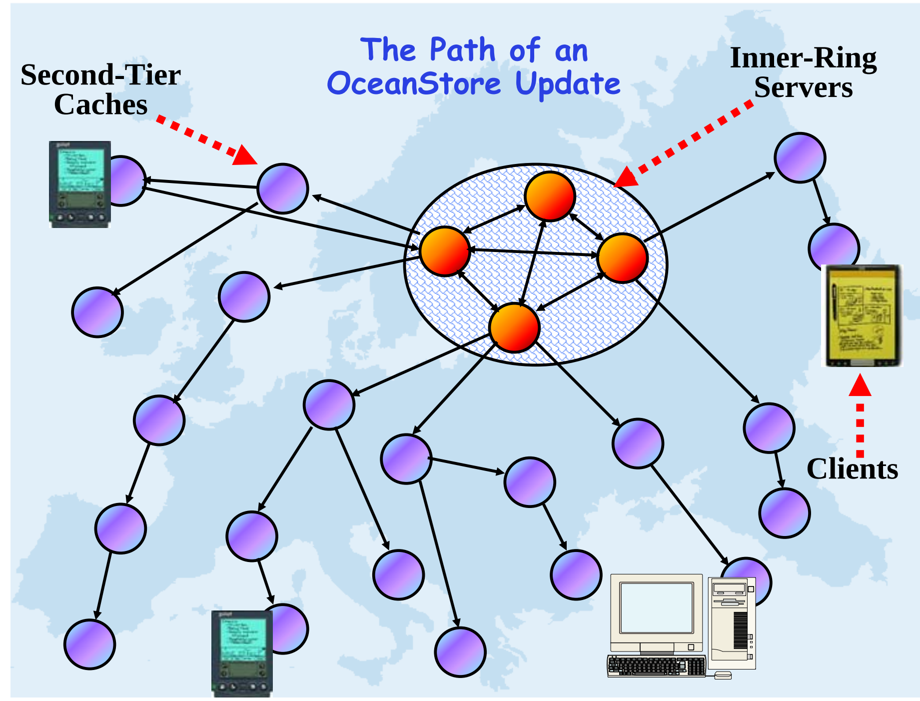
OceanStore API: Universal Conflict Resolution



- Consistency is form of optimistic concurrency
  - Updates contain predicate-action pairs
  - Each predicate tried in turn:
    » If none match, the update is aborted
    » Otherwise, action of first true predicate is applied
- Role of Responsible Party (RP):
  - Updates submitted to RP which chooses total order
- This is powerful enough to synthesize:
  - ACID database semantics
  - release consistency (build and use MCS-style locks)
  - Extremely loose (weak) consistency

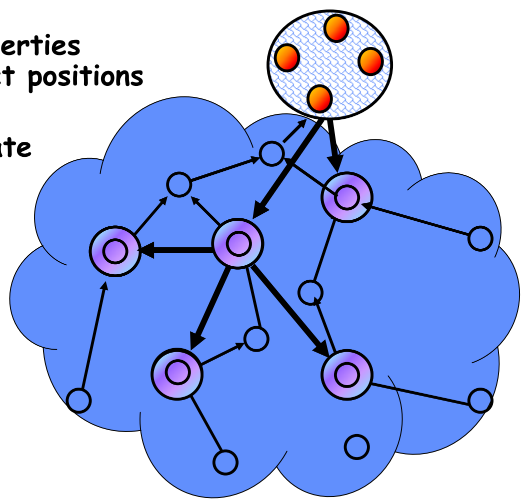
Two Types of OceanStore Data

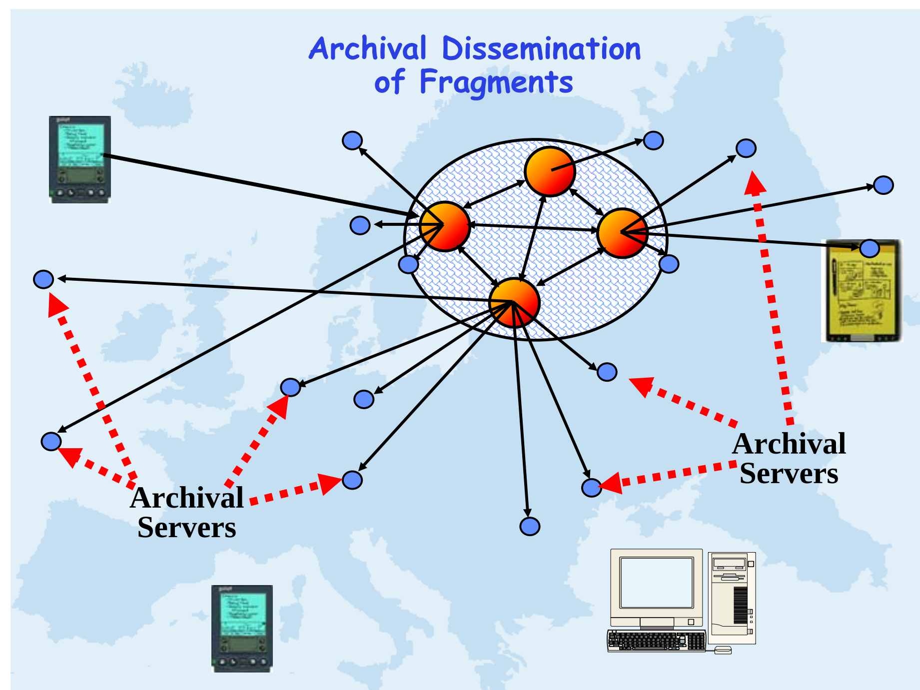
- *Active Data:* "Floating Replicas"
  - Per object virtual server
  - Interaction with other replicas for consistency
  - May appear and disappear like bubbles
- *Archival Data:* OceanStore's Stable Store
  - m-of-n coding: Like hologram
    - » Data coded into *n* fragments, any *m* of which are sufficient to reconstruct (e.g m=16, n=64)
    - » Coding overhead is proportional to n÷m (e.g 4)
    - » Other parameter, *rate*, is 1/overhead
  - Fragments are cryptographically self-verifying
- **Most data in the OceanStore is archival!**



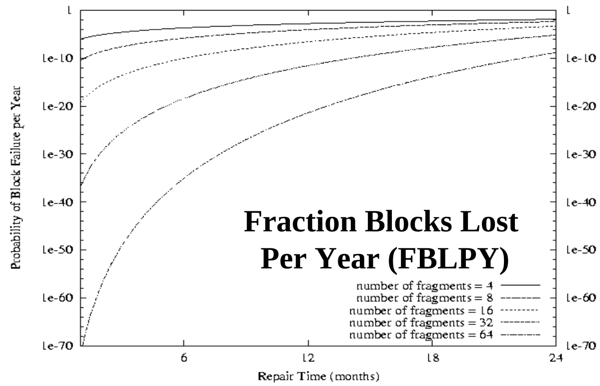
Self-Organizing Soft-State Replication

- Simple algorithms for placing replicas on nodes in the interior
  - Intuition: locality properties of Tapestry help select positions for replicas
  - Tapestry helps associate parents and children to build multicast tree
- Preliminary results encouraging
- Current Investigations:
  - Game Theory
  - Thermodynamics



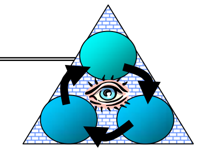


Aside: Why erasure coding? High Durability/overhead ratio!



- Exploit law of large numbers for durability!
- 6 month repair, FBLPY:
  - Replication: 0.03
  - Fragmentation: 10-35

Extreme Durability?



- Exploiting Infrastructure for Repair
  - DOLR permits efficient heartbeat mechanism to notice:
    - » Servers going away for a while
    - » Or, going away forever!
  - Continuous sweep through data also possible
  - Erasure Code provides Flexibility in Timing
- Data transferred from physical medium to physical medium
  - No "tapes decaying in basement"
  - Information becomes fully Virtualized
- Thermodynamic Analogy: Use of Energy (supplied by servers) to Suppress Entropy

Differing Degrees of Responsibility

- Inner-ring provides quality of service
  - Handles of live data and write access control
  - Focus utility resources on this vital service
  - Compromised servers must be detected quickly
- Caching service can be provided by anyone
  - Data encrypted and self-verifying
  - Pay for service "Caching Kiosks"?
- Archival Storage and Repair
  - Read-only data: easier to authenticate and repair
  - Tradeoff redundancy for responsiveness
- Could be provided by different companies!

Peer-to-peer Goal: Stable, large-scale systems

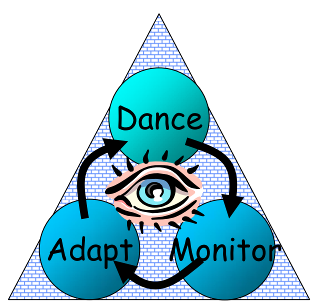
- State of the art:
  - Chips: 10^8 transistors, 8 layers of metal
  - Internet: 10^9 hosts, terabytes of bisection bandwidth
  - Societies: 10^8 to 10^9 people, 6-degrees of separation
- Complexity is a liability!
  - More components ⇒ Higher failure rate
  - Chip verification > 50% of design team
  - Large societies unstable (especially when centralized)
  - Small, simple, perfect components combine to generate complex emergent behavior!
- Can complexity be a useful thing?
  - Redundancy and interaction can yield stable behavior
  - Better figure out new ways to design things…

Exploiting Numbers: Thermodynamic Analogy

- Large Systems have a variety of *latent order*
  - Connections between elements
  - Mathematical structure (erasure coding, etc)
  - Distributions peaked about some desired behavior
- Permits "Stability through Statistics"
  - Exploit the behavior of aggregates (redundancy)
- Subject to Entropy
  - Servers fail, attacks happen, system changes
- Requires continuous repair
  - Apply energy (i.e. through servers) to reduce entropy

Exploiting Numbers: The Biological Inspiration

- Biological Systems are built from (extremely) faulty components, yet:
  - They operate with a variety of component failures ⇒ Redundancy of function and representation
  - They have stable behavior ⇒ Negative feedback
  - They are self-tuning ⇒ Optimization of common case
- Introspective (Autonomic) Computing:
  - Components for performing
  - Components for monitoring and model building
  - Components for continuous adaptation



What does this really mean?

- Redundancy, Redundancy, Redundancy:
  - Many components that are roughly equivalent
  - System stabilized by consulting multiple elements
  - Voting/signature checking to exclude bad elements
  - Averaged behavior/Median behavior/First Arriving
- Passive Stabilization
  - Elements interact to self-correct each other
  - Constant resource shuffling
- Active Stabilization
  - Reevaluate and Restore good properties on wider scale
  - System-wide property validation
  - Negative feedback/chaotic attractor
- Observation and Monitoring
  - Aggregate external information to find hidden order
  - Use to tune functional behavior and recognize dysfunctional behavior.

Problems?

- **Most people don't know how to think about this**
  - Requires new way of thinking
  - Some domains closer to thermodynamic realm than others:
    peer-to-peer networks fit well
- Stability?
  - Positive feedback/oscillation easy to get accidentally
- Cost?
  - Power, bandwidth, storage, ….
- Correctness?
  - System behavior achieved as aggregate behavior
  - Need to design around fixed point or chaotic attractor behavior (How does one think about this)?
  - Strong properties harder to guarantee
- Bad case could be quite bad!
  - Poorly designed ⇒Fragile to directed attacks
  - Redundancy below threshold ⇒ failure rate increases drastically

Conclusions

- Berkely PARLAb
  - Check out: view.eecs.berkeley.edu
    parlab.eecs.berkeley.edu
- ManyCore OS
  - Spatial partitioning
  - Thin Hypervisor
  - Explicit security tracking of information
  - Need for fine-grained synchronization
- Peer to Peer
  - A philosophy of systems design at extreme scale
  - Probabilistic design when it is appropriate
  - New techniques aimed at unreliable components
  - A rethinking (and recasting) of distributed algorithms
- Let's give a hand to the TAs!
- Good Bye!