CS162
Operating Systems and Systems Programming
Lecture 27

Peer-to-peer Systems and Other Topics

December 6th, 2006

Prof. John Kubiatowicz

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

Requests for Final topics

- Hidden Software Attacks
  - Some amusing final material from last time
- Some topics people requested:
  - Dragons: too big of a topic
    - » Here is a Chinese dragon from Wikipedia
  - Google OS
  - Parallel OSs
  - Quantum Computing
- Some Other Topics
  - Windows vs. Linux
  - Peer-to-Peer Systems (OceanStore)

Shrink Wrap Software Woes

- Can I trust software installed by the computer manufacturer?
  - Not really, most major computer manufacturers have shipped computers with viruses
  - How?
    - » Forget to update virus scanner on "gold" master machine
- Software companies, PR firms, and others routinely release software that contains viruses
- Linux hackers say "Start with the source"
  - Does that work?

Ken Thompson's self-replicating program

- Bury Trojan horse in binaries, so no evidence in source
  - Replicates itself to every UNIX system in the world and even to new UNIX's on new platforms. No visible sign.
  - Gave Ken Thompson ability to log into any UNIX system
- Two steps: Make it possible (easy); Hide it (tricky)
- Step 1: Modify login.c

```
A: if (name == "ken")
       don't check password
       log in as root
```

  - Easy to do but pretty blatant! Anyone looking will see.
- Step 2: Modify C compiler
  - Instead of putting code in login.c, put in compiler:

```
B: if see trigger1
   insert A into input stream
```

  - Whenever compiler sees trigger1 (say /*gobbledygook*/), puts A into input stream of compiler
  - Now, don't need A in login.c, just need trigger1

Self Replicating Program Continued

- Step 3: Modify compiler source code:

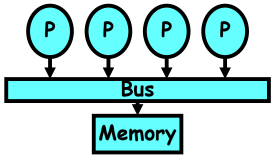
```
C: if see trigger2
     insert B+C into input stream
```

  - Now compile this new C compiler to produce binary
- Step 4: Self-replicating code!
  - Simply remove statement C in compiler source code and place "trigger2" into source instead
    - » As long as existing C compiler is used to recompile the C compiler, the code will stay into the C compiler and will compile back door into login.c
    - » But no one can see this from source code!
- When porting to new machine/architecture, use existing C compiler to generate cross-compiler
  - Code will migrate to new architecture!
- Lesson: never underestimate the cleverness of computer hackers for hiding things!

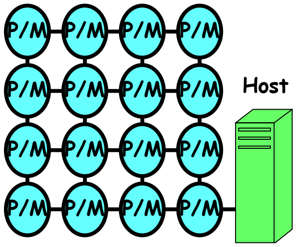
Google OS

- Is it *real* or Memorex?
  - Pure speculation! Googling Google…
  - Very thin local client (web)
    - » Google purchased writely, a web-based word processing system
    - » Gmail: web-based email
  - Storage at Google
    - » GDrive, GDS and Lighthouse?
    - » Mysterious powerpoint presentation about future products that disappeared quickly. Lots of speculation.
  - Computing at Google
  - Truly distributed system, access anywhere, anytime?
    - » What about privacy????
- Goobuntu: Google's distribution of Linux
  - A version of the Ubuntu desktop Linux distribution, based on Debian and the Gnome desktop
- Google pack (Announced at CES in January 2006)
  - A collection of desktop software bundled together

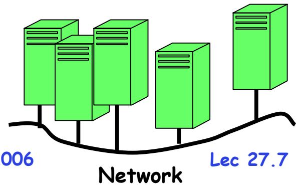
Types of Parallel Machines



- 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

Parallel OS Difference – the Kernel

- 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, …

Windows vs Linux

- Windows came from personal computer domain
  - Add-on to IBM PC providing a windowing user interface
    - » Became “good at” doing graphical interfaces
  - Didn’t have protection until Windows NT
    - » Multiple users supported (starting with Window NT), but can’t necessarily have multiple GUIs running at same time
  - Product differentiation model:
    - » Purchase separate products to get email, web servers, file servers, compilers, debuggers…
- Linux came from long line of UNIX Mainframe OSs
  - Targeted at high-performance computation and I/O
    - » High performance servers
    - » GUI historically lacking compared to Windows
  - Protection model from beginning
    - » Multiple users supported at core of OS
  - Full function Mainframe OS: email, web servers, file servers, ftp servers, compilers, debuggers, etc.

Windows vs Linux

- Internal Structure is different
  - Windows XP evolved from NT which was a microkernel
    - » Core “executive” runs in protected mode
    - » Many services run in user mode (Although Windowing runs inside kernel for performance)
    - » Object-oriented design: communication by passing objects
    - » Event registration model: many subsystems can ask for callbacks when events happen
    - » Loadable modules for device drivers and system extension
  - Linux Evolved from monolithic kernel
    - » Many portions of kernel operate in same address space
    - » Loadable modules for device drivers and system extension
    - » Fewer layers ⇒ higher performance
- Source Code development model
  - Windows: closed code development
    - » Must sign non-disclosure to get access to source code
    - » “Cathedral” model of development: only Microsoft’s developers produce code for Windows
  - Linux: open development model
    - » All distributions make source code available to analyze
    - » “Bazaar” model of development: many on the net contribute to making Linux distribution

Windows vs Linux

- Perceptions:
  - Windows has more bugs/is more vulnerable to viruses?
    - » True? Hard to say for sure
    - » More Windows systems ⇒ more interesting for hackers
  - Linux simpler to manage?
    - » True? Well, Windows has hidden info (e.g. registry)
    - » Linux has all configuration available in clear text
  - Microsoft is untrustworthy? Many distrust “the man”
    - » Quick to adopt things like Digital Rights Management (DRM)
    - » Quick to embrace new models of income such as software rental which counter traditional understanding of software
  - Windows is slow?
    - » This definitely seemed to be true with earlier versions
    - » Less true now, but complexity may still get in way
- Why choose one over other?
  - Which has greater diversity of graphical programs?
    - » Probably Windows
  - Which cheaper? Well, versions of Linux are “free
  - Which better for developing code and managing servers?
    - » Probably Linux, although this is changing
    - » OS API (e.g. system calls) definitely seem simpler

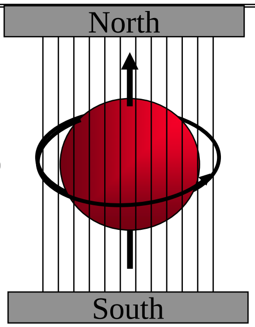
What *IS* Quantum Computing?

Can we Use Quantum Mechanics to Compute?

- Weird properties of quantum mechanics?
  - You've already seen one: tunneling of electrons through insulators to make TMJ RAM
  - Quantization: Only certain values or orbits are good
    - » Remember orbitals from chemistry???
  - Superposition: Schizophrenic physical elements don't quite know whether they are one thing or another
- All existing digital abstractions try to eliminate QM
  - Transistors/Gates designed with classical behavior
  - Binary abstraction: a "1" is a "1" and a "0" is a "0"
- Quantum Computing:
  Use of Quantization and Superposition to compute.
- Interesting results:
  - Shor's algorithm: factors in polynomial time!
  - Grover's algorithm: Finds items in unsorted database in time proportional to square-root of n.

Quantization: Use of "Spin"

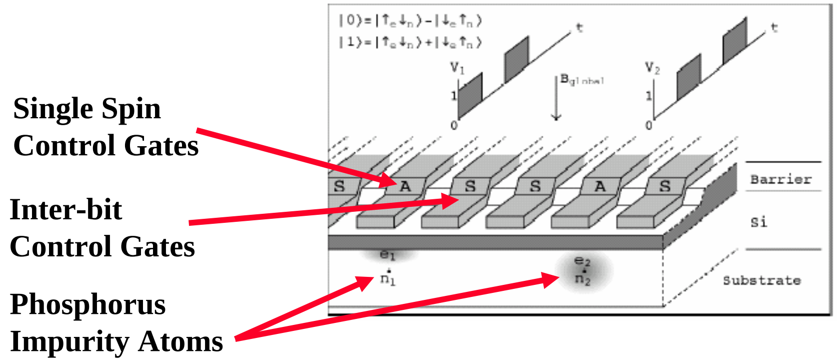
Spin ½ particle:
(Proton/Electron)



Representation:
|0> or |1>

- Particles like Protons have an intrinsic "Spin" when defined with respect to an external magnetic field
- Quantum effect gives "1" and "0":
  - Either spin is "UP" or "DOWN" nothing between

Kane Proposal II (First one didn't quite work)


$|0\rangle = |\uparrow_e \downarrow_n\rangle - |\downarrow_e \uparrow_n\rangle$
$|1\rangle = |\uparrow_e \downarrow_n\rangle + |\downarrow_e \uparrow_n\rangle$
V_1
1
0
t
B_{global}
V_2
1
0
t
S
A
S
S
A
S
Barrier
Si
Substrate
e_1
n_1
e_2
n_2

Single Spin Control Gates

Inter-bit Control Gates

Phosphorus Impurity Atoms

- Bits Represented by combination of proton/electron spin
- Operations performed by manipulating control gates
  - Complex sequences of pulses perform NMR-like operations
- Temperature < 1° Kelvin!

Now add Superposition!

- The bit can be in a combination of "1" and "0":
  - Written as: $\Psi = C_0|0> + C_1|1>$
  - The C's are *complex numbers!*
  - Important Constraint: $|C_0|^2 + |C_1|^2 = 1$
- If *measure* bit to see what looks like,
  - With probability $|C_0|^2$ we will find $|0>$ (say "UP")
  - With probability $|C_1|^2$ we will find $|1>$ (say "DOWN")
- Is this a real effect? Options:
  - This is just statistical – given a large number of protons, a fraction of them ($|C_0|^2$) are "UP" and the rest are down.
  - This is a real effect, and the proton is really both things until you try to look at it
- Reality: second choice!
  - There are experiments to prove it!

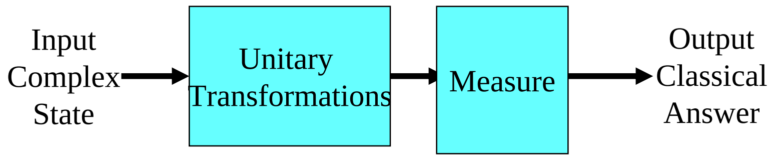
Implications: A register can have many values

- Implications of superposition:
  - An *n*-bit register can have 2^n values simultaneously!
  - 3-bit example:

$$\Psi = C_{000}|000> + C_{001}|001> + C_{010}|010> + C_{011}|011> + C_{100}|100> + C_{101}|101> + C_{110}|110> + C_{111}|111>$$

- Probabilities of measuring all bits are set by coefficients:
  - So, prob of getting $|000>$ is $|C_{000}|^2$, etc.
  - Suppose we measure only one bit (first):
    - » We get "0" with probability: $P_0 = |C_{000}|^2 + |C_{001}|^2 + |C_{010}|^2 + |C_{011}|^2$
      Result: $\Psi = (C_{000}|000> + C_{001}|001> + C_{010}|010> + C_{011}|011>)$
    - » We get "1" with probability: $P_1 = |C_{100}|^2 + |C_{101}|^2 + |C_{110}|^2 + |C_{111}|^2$
      Result: $\Psi = (C_{100}|100> + C_{101}|101> + C_{110}|110> + C_{111}|111>)$
- Problem: Don't want environment to *measure* before ready!
  - Solution: Quantum Error Correction Codes!

Model? Operations on coefficients + measurements



- Basic Computing Paradigm:
  - Input is a register with superposition of many values
    - » Possibly all 2^n inputs equally probable!
  - Unitary transformations compute on coefficients
    - » Must maintain probability property (sum of squares = 1)
    - » Looks like doing computation on all 2^n inputs simultaneously!
  - Output is one result attained by measurement
- If do this poorly, just like probabilistic computation:
  - If 2^n inputs equally probable, may be 2^n outputs equally probable.
  - After measure, like picked random input to classical function!
  - All interesting results have some form of "fourier transform" computation being done in unitary transformation

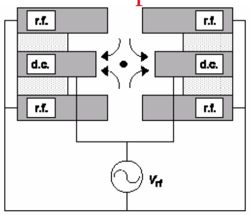
Some Issues in building quantum computer

- What are the bits and how do we manipulate them?
  - NMR computation: use "cup of liquid".
    - » Use nuclear spins (special protons on complex molecules).
    - » Manipulate with radio-frequencies
    - » IBM Has produced a 7-bit computer
  - Silicon options (more scalable)
    - » Impurity Phosphorus in silicon
    - » Manipulate through electrons (including measurement)
    - » Still serious noise/fabrication issues
  - Other options:
    - » Optical (Phases of photons represent bits)
    - » Single ions trapped in magnetic fields
- How do we prevent the environment from "Measuring"?
  - Make spins as insulated from environment as possible
  - Quantum Error Correction!
- Where get "clean" bits (I.e. unsuperposed $|0>$ or $|1>$)?
  - Entropy exchange unit:
    - » Radiates heat to environment (entropy)
    - » Produces clean bits (COLD) to enter into device

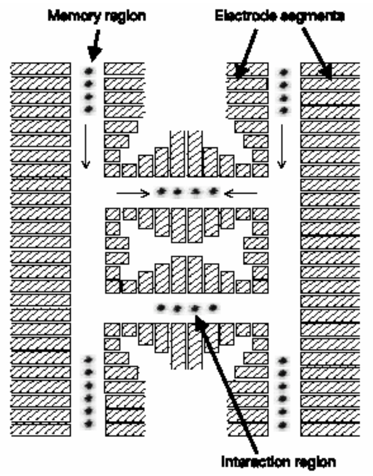
ION Trap Quantum Computer: Promising technology

Top

Cross-Sectional View



- IONS of Be+ trapped in oscillating quadrature field
  - Internal electronic modes of IONS used for quantum bits
  - MEMs technology
  - Target? 50,000 ions
  - ROOM Temperature!
- Ions moved to interaction regions
  - Ions interactions with one another moderated by lasers



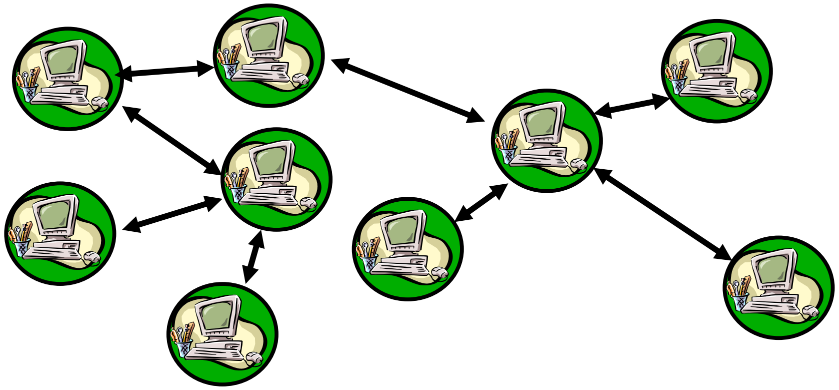
Top View

Proposal: NIST Group

Administrivia

- Midterm II
  - Still Grading!
  - I put up solutions already
- Status of Project 3 grading – hopefully soon.
- Tomorrow's section
  - Discussion of Midterm II/review for Final
  - Questions about Project 4
- Project 4
  - Due this Friday, 12/8
- Final Exam
  - 8:00-11:00, December 16th
  - Bechtel Auditorium
  - 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

Is Peer-to-peer new?

- Certainly doesn't seem like it
  - What about Usenet? News groups first truly decentralized system
  - DNS? Handles huge number of clients
  - Basic IP? Vastly decentralized, many equivalent routers
- One view: P2P is a reverting to the old internet
  - Remember? (Perhaps you don't)
  - Once upon a time, all members on the internet were trusted.
    - » Every machine had an IP address.
    - » Every machine was a client and server.
    - » Many machines were routers and/or were equivalent
- But: peer-to-peer seems to mean something else
  - More about the *scale* (total number) of directly interacting components
  - Also, has a "bad reputation" (stealing music)

Research Community View of Peer-to-Peer

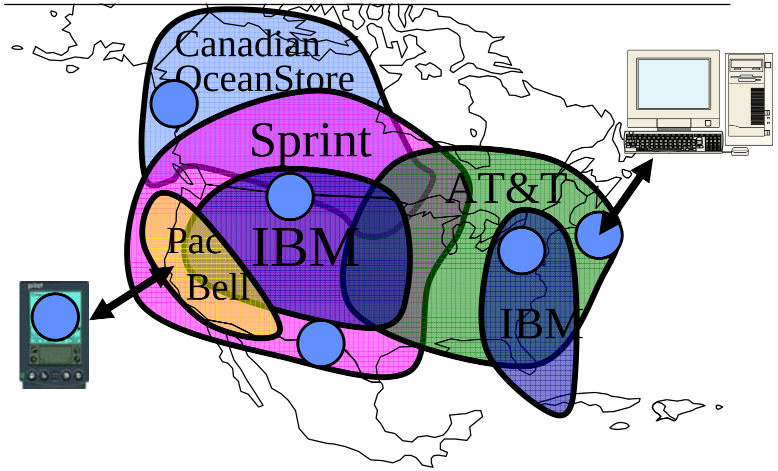
- Old View:
  - A bunch of flakey high-school students stealing music
- New View:
  - A philosophy of systems design at extreme scale
  - Probabilistic design when it is appropriate
  - New techniques aimed at unreliable components
  - A rethinking (and recasting) of distributed algorithms
  - Use of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

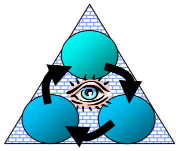
- How many files in the OceanStore?
  - Assume 10^{10} people in world
  - Say 10,000 files/person (very conservative?)
  - So 10^{14} files in OceanStore!
  - If 1 gig files (ok, a stretch), get 1 mole of bytes! (or a Yotta-Byte if you are a computer person)

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

Aside: SIMS school: 1.5 Exabytes/year (1.5×10^{18})

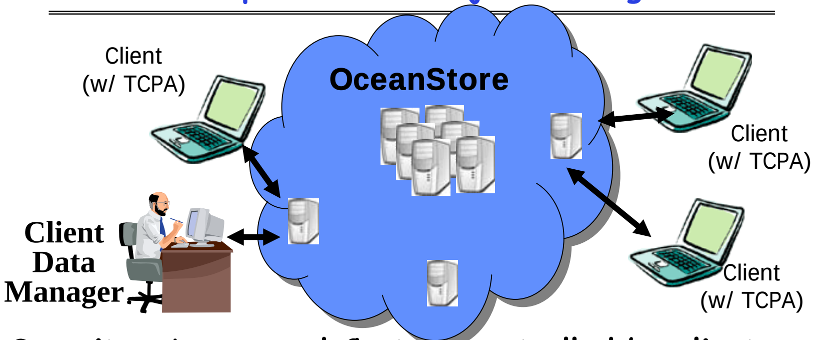
Key Observation: Want Automatic Maintenance

- 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

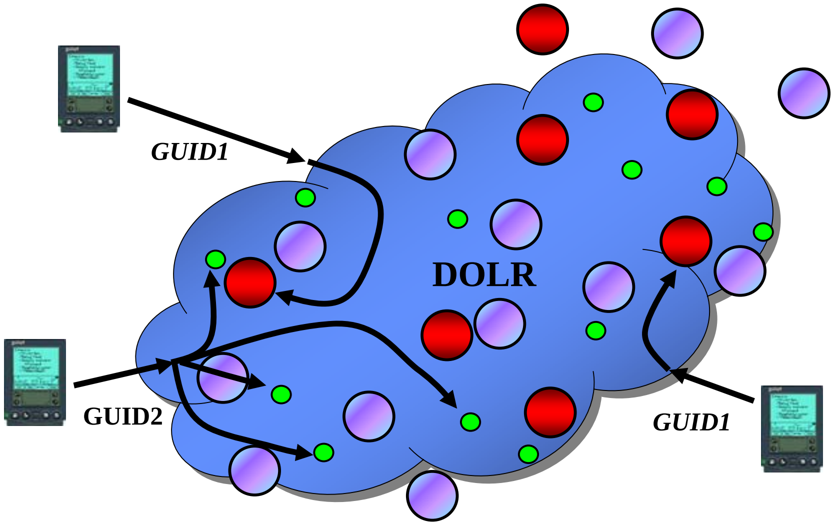
- Untrusted Infrastructure:
  - The OceanStore is comprised of untrusted components
  - Individual hardware has finite lifetimes
  - All data encrypted within the infrastructure
- Mostly Well-Connected:
  - Data producers and consumers are connected to a high-bandwidth network most of the time
  - Exploit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

Quality-of-Service

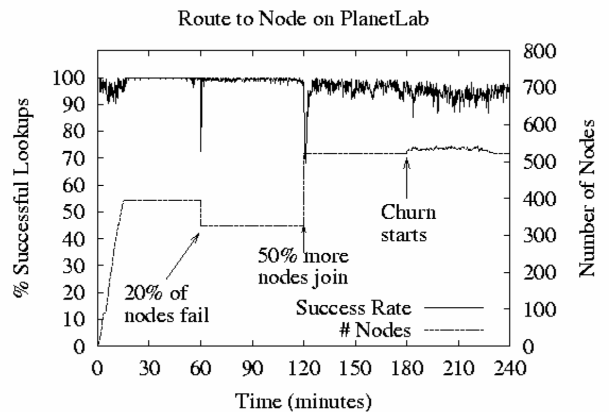
- Responsible Party:
  - Some organization (*i.e. service provider)* guarantees that your data is consistent and durable
  - Not trusted with *content* of data, merely its *integrity*

Peer-to-Peer for Data Location

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



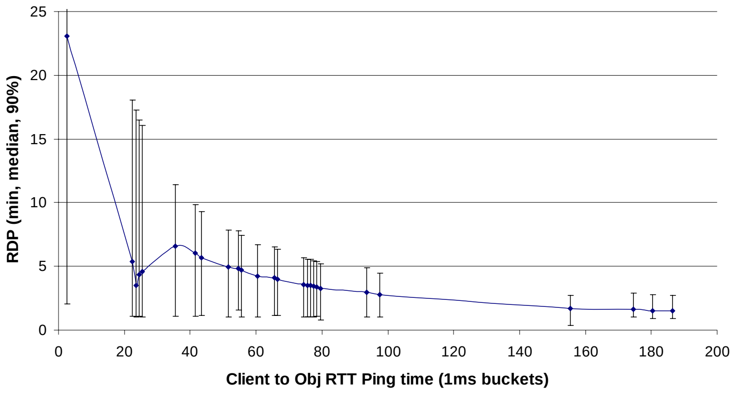
Stability under extreme circumstances



(May 2003: 1.5 TB over 4 hours)

DOLR Model generalizes to many simultaneous apps

Object Location with Tapestry DOLR

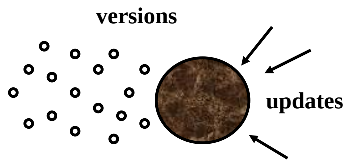


Peek at OceanStore Mechanisms

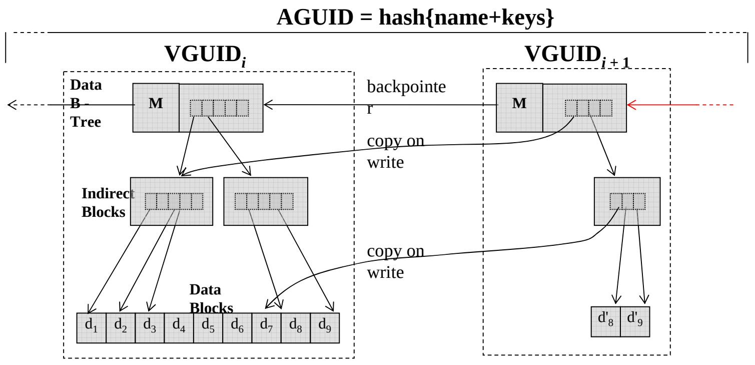
OceanStore Data Model

- 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

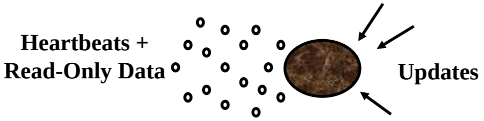
Comet Analogy



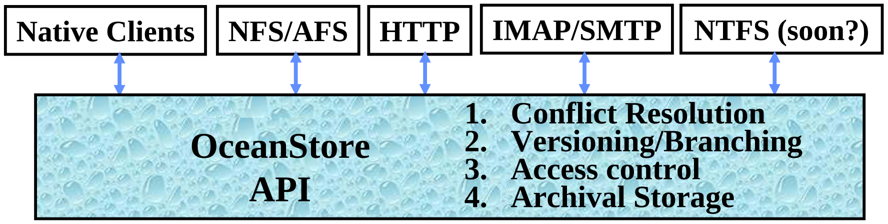
Self-Verifying Objects



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



OceanStore API: Universal Conflict Resolution



Native Clients | NFS/AFS | HTTP | IMAP/SMTP | NTFS (soon?)

OceanStore API

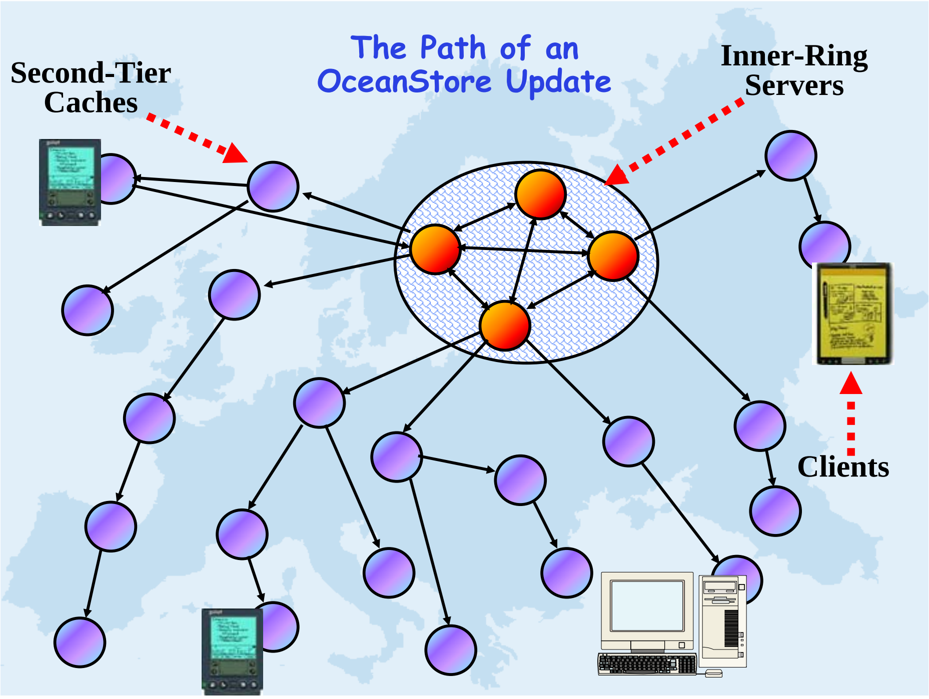
1. Conflict Resolution
2. Versioning/Branching
3. Access control
4. Archival Storage

- 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

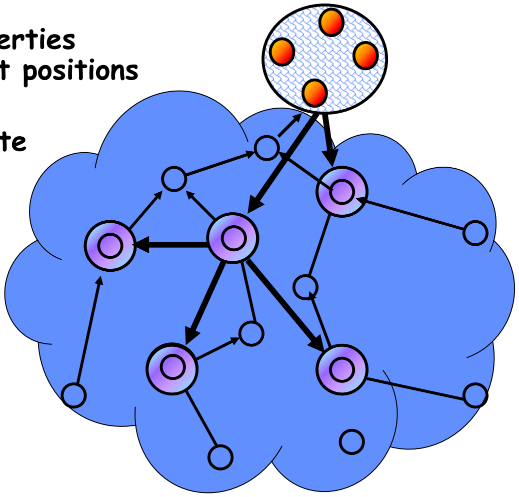
- *Active Data:* "Floating Replicas"
  - Per object virtual server
  - Interaction with other replicas for consistency
  - May appear and disappear like bubbles
- *Archival Data:* OceanStore's Stable Store
  - m-of-n coding: Like hologram
    - » Data coded into *n* fragments, any *m* of which are sufficient 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!

The Path of an OceanStore Update

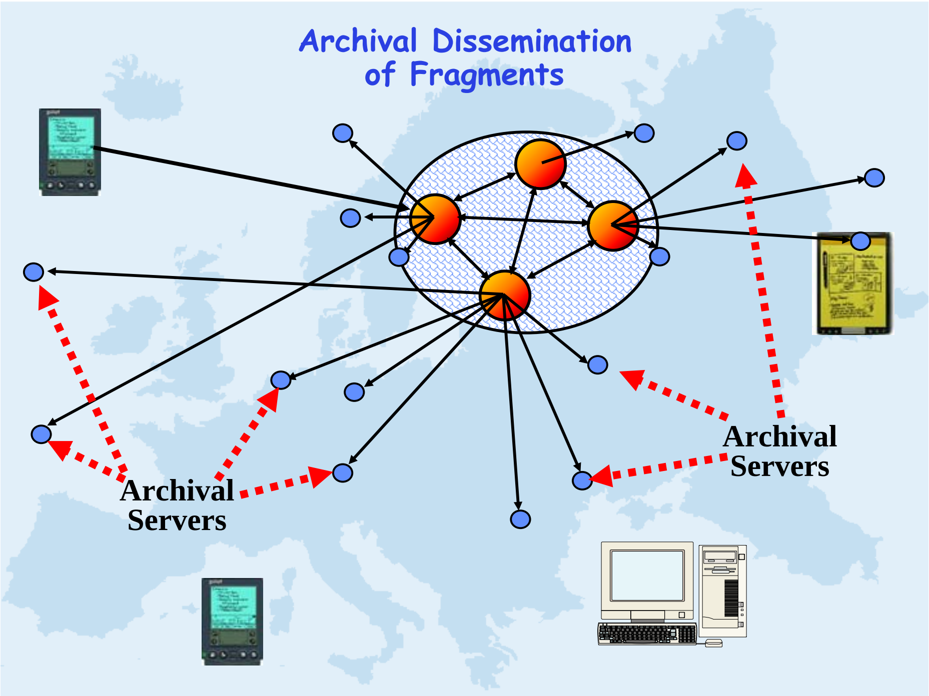


Self-Organizing Soft-State Replication

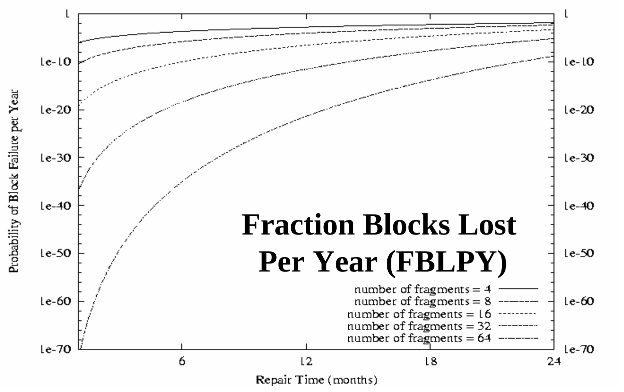
- Simple algorithms for placing replicas on nodes in the interior
  - Intuition: locality properties of Tapestry help select positions for replicas
  - Tapestry helps associate parents and children to build multicast tree
- Preliminary results encouraging
- Current Investigations:
  - Game Theory
  -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



Archival Dissemination of Fragments



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!

Closing View on Peer-to-Peer

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

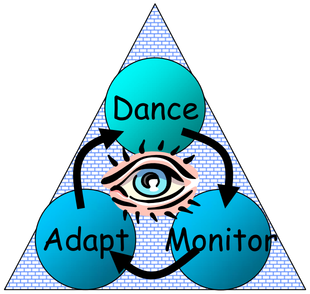
- State of the art:
  - Chips: 10^8 transistors, 8 layers of metal
  - Internet: 10^9 hosts, terabytes of bisection bandwidth
  - Societies: 10^8 to 10^9 people, 6-degrees of separation
- Complexity is a liability!
  - More components ⇒ Higher failure rate
  - Chip verification > 50% of design team
  - 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

- Google OS
  - Not so much a product as a speculation on future direction
- Parallel OSs
  - Need for fine-grained synchronization
- Windows vs Linux:
  - Graphics vs Server?
  - Cathedral vs Bazaar
  - Controlled vs Free
- Quantum Computing
  - Using interesting properties of physics to compute
- 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!