Supporting Mobility in File Systems

- With the advent of mobile computing and the need to conveniently access data from servers it became clear that traditional file systems were inadequate
- This lead to the design of the CODA filesystem which provides direct support for replication and disconnected operation
  - Replication to enhance availability
  - Disconnected operation to support mobility

CODA basics

- Coda is tailored to access patterns typical of academic and research environments
  - Relatively little concurrent file sharing
- Not intended for applications exhibiting highly concurrent fine granularity data access
- Clients view Coda as a single location-transparent shared Unix file system
- Coda namespace is mapped to individual file servers at the granularity of subtrees
- Each client has a **cache manager (VICE)**
  - Clients run on the mobile devices

CODA Benefits

- High availability achieved using:
  - **Server replication**
    - Set of replicas of a volume is **a VSG** (Volume Storage Group)
    - At any time, client can access files in the **AVSG** (Available Volume Storage Group)
  - **Disconnected Operation**
    - When there are no available VSGs

CODA Benefits

- Must handle two kinds of failures
  - ***Server failures***
    - Data servers are ***replicated***
  - ***Communication failures*** or ***voluntary disconnections***
    - Due to mobility
    - Coda uses ***optimistic replication*** and ***file hoarding***

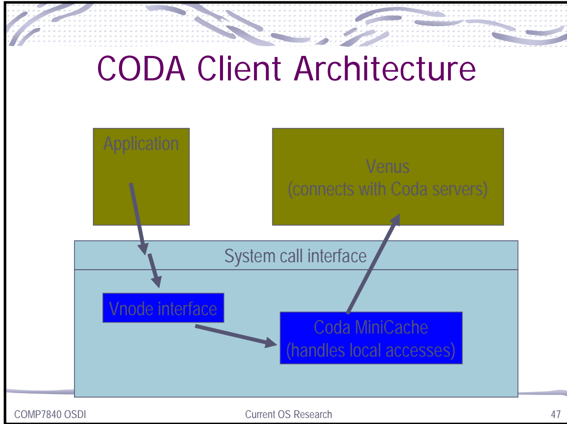
CODA challenges

- Normally consistency among replicas is provided using majority voting
  - This fails with possible disconnections
- ***Optimistic replica control*** allows access in disconnected mode
  - Tolerates temporary inconsistencies
  - Promises to detect them later
  - Provides ***much higher data availability***

CODA challenges (cont'd)

- Define an ***accessible universe***
  - set of replicas that the user can access
- The accessible universe varies over time
- At any time, the user
  - reads from the latest replica(s) in the accessible universe
  - updates all replicas in the accessible universe

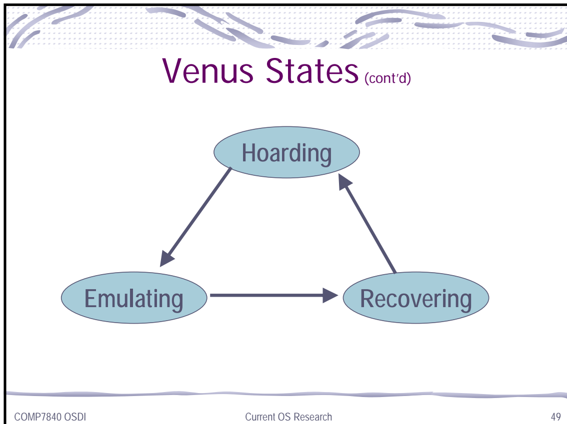
CODA Client Architecture



Venus States

1. ***Hoarding:***
   Normal operation mode
2. ***Emulating:***
   Disconnected operation mode
3. ***Reintegrating:***
   Propagates changes and detects inconsistencies

Venus States (cont'd)



CODA Emulation Mode

- In emulation mode:
  - Attempts to access files that are not in the client caches appear as failures to application
  - All changes are written in a ***client modification log*** (CML)

CODA Persistence

- Venus keeps its cache and related data structures in non-volatile storage
- All Venus metadata are updated through ***atomic transactions***
  - Using a lightweight ***recoverable virtual memory*** (RVM) developed for Coda

CODA Reintegration

- When a mobile unit is reconnected, Coda initiates a ***reintegration process***
  - Performed one volume at a time
  - Venus ships replay log to all volumes
  - Each volume performs a log replay algorithm

Supporting Massive File Systems

- As the size of file systems increase, it becomes increasingly difficult to locate files
  - Especially when considering historical, distributed files
- This is a naming issue
  - Normally addressed with hierarchical structures
- The design goals of a naming system that enables dynamic resource discovery and service location:

The Intentional Naming System

- "The Design and Implementation of an Intentional Naming System"
  - Adjie-Winoto, Schwartz, Balakrishnan and Lilley
- The Intentional Naming System (INS) allows applications to describe what they are looking for, not where to find it.
- Intentional Naming Resolvers (INR), which can be provided by any devices in distributed system, self-configure into a spanning-tree based on metrics that reflect INR-to-INR round-trip latency.
- These resolvers then provide a lookup service

Architecture of INS

Architecture of INS

- Name-specifiers
- Discovering names
- Name lookup and extraction
- Resolver network
- Load balancing and scaling

Name-specifiers

Wild card (*) is supported

```
[city = washington [building = whitehouse
                    [wing = west
                    [room = oval-office]]]]
[service = camera   [data-type = picture
                      [format = jpg]]
                    [resolution = 640x480]]
[accessibility = public]
```


Discovering Names

- Services periodically advertise their intentional names to the system to describe what they provide.
- INRs update contained name information when newer information becomes available or discard as no refresh announcement is received or lifetime is out.
- Clients get all matched names from an INR by sending an intentional name with the name discovering message.
- INRs disseminate name information between each other using periodic updates and triggered updates.

Name Lookup and Extraction

- Name-trees

A data structure storing the correspondence between name-specifiers and name-records.

Resolver Network

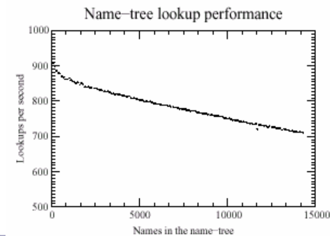
- INRs build up a spanning-tree network.
- A well-known entity in the network, called the Domain Space Resolver (DSR), is used to administrate domain constructed by a set of INRs.
- A new INR sends INR-pings messages to all active INRs and picks the INR with minimal value of round-trip latency to establish a neighbor relation with.

Load balancing & scaling

- An INR can obtain a candidate list from DSR and spawn new instances on other candidates to handle some of its current load as the INR is loaded heavily because of name lookups.
- When an INR is loaded due to update processing, the namespace is partitioned into several virtual spaces for scaling, and the names of different virtual space is stored in separate name trees.

Performance Evaluation

Name lookup performance



INS – Conclusions

- The INS integrates resolution and routing, allowing applications seamlessly handle the mobility of services and nodes.
- The resolvers can self-configure into a network and incorporate load-balancing algorithm to perform well.
- Using intentional anycast and intentional multicast provides a useful, flexible way of discovering resources in dynamic, mobile networks, and simplifies the implementation of applications.

Some Recent Issues in Scheduling

It's a Changing World

- Assumption about bi-modal workload no longer holds
  - Interactive continuous media applications
    - E.g. Graphics viewers are definitely processor-bound but needs good response time
- New computing model requires more flexibility
  - How to match priorities of cooperative jobs?
    - E.g. client/server
  - How to balance execution between multiple threads of a single process?

Fair Share Scheduling

- Goal: Each user gets their "fair share" of the resources (e.g. CPU)
- Process scheduling priority determined by allocated "share" and recent use
  - If process has used more than its share, its priority is lowered
  - If process has received less than its share, its priority is increased
  - Priorities can get low and need adjustment but every process gets a "kick at the can"

Lottery Scheduling

- "Lottery Scheduling:Flexible Proportional-Share Resource Management" by Waldspurger and Weihl from OSDI'94
- Randomized resource allocation mechanism
- Resource rights are represented by lottery tickets assigned to processes
- Each round of a lottery the winning ticket (i.e. the scheduled process) is chosen at random
- The chances of you winning directly depends on the number of tickets that you have
  - P[winning] = t/T, t = your number of tickets, T = total number of tickets

Lottery Scheduling

- After n rounds, your expected number of wins is
  - E[win] = nP[winning]
- The expected number of lotteries that a client must wait before its first win
  - E[wait] = 1/P[winning]
- Lottery scheduling implements *proportional-share* resource management
- Like fair-share scheduling but cheaper and fast to respond to changes in priorities
- OK, so how do we actually schedule the processor using lottery scheduling?

Selecting a winner

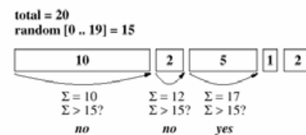


Figure 1: **Example Lottery.** Five clients compete in a list-based lottery with a total of 20 tickets. The fifteenth ticket is randomly selected, and the client list is searched for the winner. A running ticket sum is accumulated until the winning ticket value is reached. In this example, the third client is the winner.

Performance

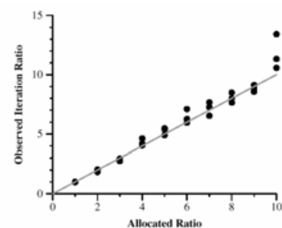


Figure 4: **Relative Rate Accuracy.** For each allocated ratio, the observed ratio is plotted for each of three 60 second runs. The gray line indicates the ideal where the two ratios are identical.

Conclusion: lottery scheduling comes very close to providing the shares requested

Scheduling in Parallel Systems

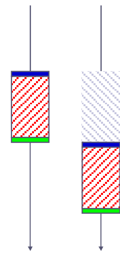
- "Scheduling Techniques for Concurrent Systems" by John Ousterhout, ICDCS'82
  - An "oldie but a goodie"
- One of the classic papers on parallel processor scheduling
- Two sets of issues:
  - Synchronization
  - Co-scheduling

Parallel Synchronization

- Recall: Mutual exclusion ≡ want to be only thread modifying a shared data
- Have three steps:
  - Acquire, Release, Waiting
- Acquire/release operations often termed Lock/Unlock
- Example: transferring $10 from A to B

```
Lock(A)
Lock(B)
A ← A + 10
B ← B - 10
Unlock(B)
Unlock(A)
```

```
Function Transfer (Amount, A, B)
    Lock(Transfer_Lock)
    A ← A + 10
    B ← B - 10
    Unlock(Transfer_Lock)
```



What To Do While Waiting?

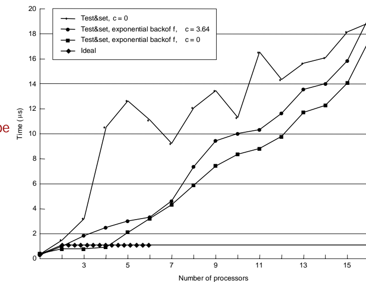
- Blocking
  - OS or RT system de-schedules waiting threads
  - Allows processor to do other things but high overhead
- Spinning
  - Waiting threads keep testing location until it changes
  - Lower overhead but keeps processor busy and can cause bus traffic
- Spinning better when
  - Scheduling overhead is larger than expected wait time
  - Processor not needed for other tasks
- Hybrid methods: Spin a while, then block

Lock Contention

Culler et al., 1999

Measured on an SGI Challenge

Hmm, looks like this is going to be important!



Parallel Synchronization

- Spin for a while then block
- How much to spin?
  - Too little then you will pay the cost of both spinning and context switching
  - Too much leads to wasting processor cycles and contention showed in last slide
  - Research shows that spinning for t = context switch time is competitive – it's at worst 2 times as bad as the optimal algorithm
  - Adaptive spinning can do even better

The Need for Co-scheduling

- Cooperating processes may interact frequently
  - What's the problem with this?
- Fine-grain parallel applications have a *process* working set
- Two things needed
  - Identify the process working set of a job
  - Co-schedule them
- Ousterhout
  - Identifying process working set is hard – let's punt for now
  - Just co-schedule parallel programs that have an explicit process working set

Co-scheduling Algorithms

- First issue is where should processes execute and can the OS control this?
  - e.g. Do they have "affinity" for specific processors
    - Based on available cache data or ...
- Ousterhout described three "algorithms"
  - Matrix
  - Continuous Algorithm
  - Undivided Algorithm

Ousterhout's Matrix Algorithm

- Construct a matrix of rows where each row has 'p' elements (p=# of processors)
- A new process working set ("gang") is either assigned to a new row or, if possible, is fit into empty spaces in an existing gang
- Schedule a "row" (i.e. gang) at a time
  - All processes from a working set so IPC wait times are minimized
  - It is also safe to use spin locks – only wasting the same process' time

Ousterhout's other Algorithms

- The matrix algorithm suffers from "process fragmentation"
  - Process working sets must fit in one row and this is unnatural
    - Often some "slots" (i.e. processors) left unfilled (unused)
- Ousterhout's continuous algorithm views the process space as a sequence and slides a window of size 'P' over the sequence until the leftmost slot in the window corresponds to the first process to be scheduled in a working set yet to be executed
- The undivided algorithm is an extension of this

Performance of Scheduling Algos

- A. Gupta, A. Tucker,and S. Urushibara "*The Impact of Operating System Scheduling Policies and Synchronization Methods on the Performance of Parallel Applications*" SIGMETRICS '91 continue this sort of work.
- They consider the performance of a set of applications using a variety of scheduling schemes:
  - Feedback priority scheduling with:
    - Spinning vs. Blocking
    - Spin-and-block
    - Block-and-hand-off (to a specified process)
    - Block-and-Affinity (what's in the cache)
  - Gang Scheduling (e.g. Matrix) – Time sharing
  - Process Control – Space sharing
    - Application varies number of processes based on processors available

Applications

Table 1: General statistics for the benchmarks.

Application	Total Cycles ($\times 10^6$)	Total Refs ($\times 10^6$)	Shared Refs ($\times 10^6$)	Cache Misses ($\times 10^6$)	Speedup
MP3D	160	38	30 (80%)	2.7 (7%)	11.1
LU	93	17	17 (99%)	0.7 (4%)	18.1
Maxflow	194	48	29 (60%)	3.1 (6%)	10.0
Matmul	600	107	54 (50%)	4.0 (4%)	12.0

Batch vs. Regular Priority

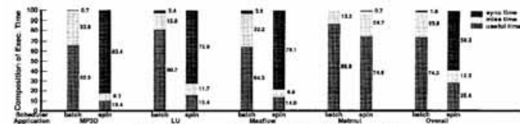


Figure 2: Comparison of batch mode and regular priority scheduling.

Blocking Synchronization

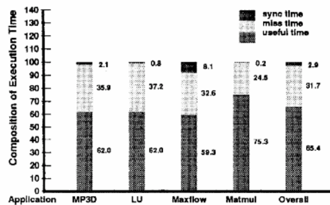


Figure 3: Performance with blocking locks.

Gang Scheduling

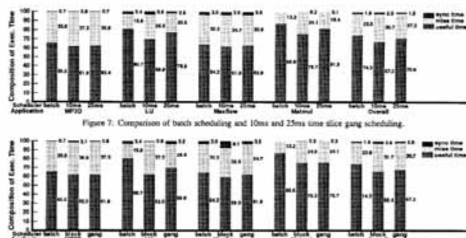


Figure 7: Comparison of batch scheduling and 10ms and 25ms time slice gang scheduling.

Figure 8: Comparison of batch, blocking, and gang scheduling.

Scheduling with *Implicit* Info.

- Lots of research results show that space-sharing outperforms time-sharing for parallel processor scheduling if applications are structured to deal with space-sharing
- Without appropriate system support, it's difficult to write programs for space-sharing
- Also, hard to space-share in a distributed environment (client/server)
- Want some form of Gang Scheduling
  - Hard to implement on clusters of independent machines
  - May not have explicit working set information

Implicit Co-scheduling

- Two-level scheduler
  - Kernel-level scheduler is ignorant of parallel nature of jobs
  - User-level scheduler uses *local* events to achieve co-scheduling
- Basic idea is for a thread to
  - Relinquish processor if peers are not scheduled
  - Hold on to the processor if peers are scheduled
- Use two-phase spin-block and priority as the two basic mechanisms to implement implicit co-scheduling
- Use message response time, message arrival, and scheduling progress as meaningful local events

Implicit Information

Information	Type	Local Observation:	Remote Implication	Local Action	Mechanism
Response Time	Inherent	Fast:	Remote scheduled	Stay scheduled	Two-phase spin-block
		Slow:	Remote not scheduled	Relinquish processor	Two-phase spin-block
Message Arrival	Inherent	Request:	Sender scheduled	Increase Spin-Time	Two-phase spin-block
		Response:	Receiver scheduled	Wake original sender	Two-phase spin-block
Scheduling Progress	Derived	Starving locally:	Starving globally	Schedule job more	Raise priority
		Acceptable locally	Acceptable globally	(No action)	(None)

Some Recent Issues in Protection and Sharing

Single Address Space OSs

- Historically, processes have operated in separate addresses spaces and the OS has been isolated from user processes
- This provides protection but makes sharing inconvenient and costly
  - Must cross address space boundaries via OS calls
- With 64 bit addressing, the need to separate address spaces is disappearing
- If a single address space is to be shared, some other form of protection is needed

The 'Opal' SAS OS

- The Opal Single Address Space (SAS) OS developed at U. Washington introduced this idea
- Proposed mapping primary and secondary storage persistently into the single virtual address space
  - Also across the network
- Separate protection via "protection domains"
  - Sort of like capabilities with HW support

The 'Opal' SAS OS (cont'd)

- Using a single address space offers several advantages:
  - Improved sharing
    - Easy and efficient
  - Better efficiency
    - HW supported protection is cheaper than address space creation, etc.
  - Support for persistence
    - No more files

Protection Domains

- A protection domain is an abstraction of a set of access rights and multiple threads may be assigned to the same protection domain
- Virtual memory segments are explicitly attached to protection domains using "capabilities"
- Capabilities can be passed via shared memory
- Since there is a single address space, pointers maintain significance across processes
  - No pointer swizzling

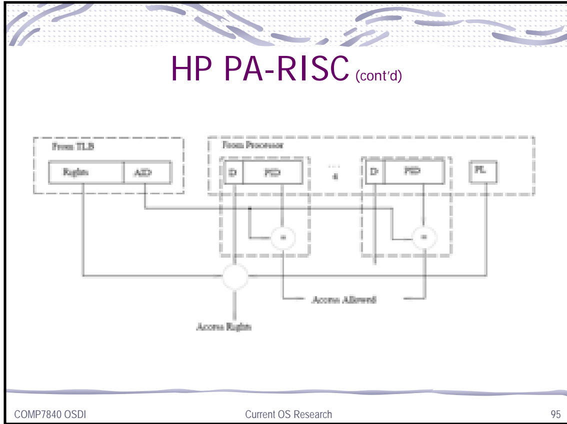
Protection Lookaside Buffer

- Conceptually, the enforcement of protection could be done entirely in software but, as with conventional virtual address translation, it is more efficient with hardware support
- Koldinger, et al propose a PLB which caches protection information on a per <domain,page> basis
  - Each thread belongs to a domain and has access to pages in segments it has attached
  - Access checks are done in HW using the PLB

HP PA-RISC

- The PLB is only a theoretical architectural component
- It is possible to implement protection domains without a PLB
- For example, the HP PA-RISC architecture supports a TLB that contains an AID (Access IDentifier) for each entry
- The executing thread can load AIDs for virtual segments into page group registers
  - These are checked against the AID in HW

HP PA-RISC (cont'd)



COMP7840 OSDI Current OS Research 95

Resource Containers

- "Resource Containers: A new facility for resource management in server systems"
  - Banga, Druschel and Mogul from OSDI'99
- The paper is concerned with *server* systems
  - File servers, web servers, compute servers, etc.
- We want to be able to accurately track and control access to OS resources by clients
  - To enable fair sharing of, and charging for, resource consumption
- Current OS abstractions are inadequate to do this
  - Clients/users are not always well identified at the servers so who should be charged, etc.?

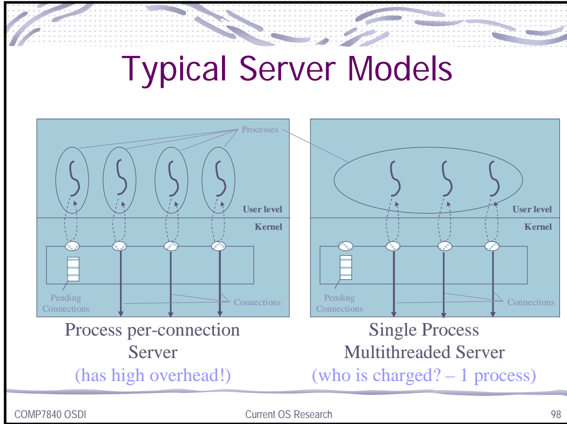
COMP7840 OSDI Current OS Research 96

The Problem

- Existing OS resource management mechanisms are most commonly tied to processes running on a given machine
  - Processes (protection domains) are the unit of resource management – the "resource principal"
    - See Single Address Space OSs
- This is inadequate to permit servers to effectively manage their services
  - Commonly, a single server process performs functions on behalf of different entities (i.e. clients) which are attributed to the single process

COMP7840 OSDI Current OS Research 97

Typical Server Models



Process per-connection Server
(has high overhead!)

Single Process Multithreaded Server
(who is charged? – 1 process)

COMP7840 OSDI Current OS Research 98

The Proposed Solution

- How can this problem be fixed?
- Must separate the notion of resource principal from that of process
  - or any other type of protection domain
- "Resource Containers" are a new OS abstraction that does exactly this
  - OS manages resource containers and tracks usage for them but they are not associated with any particular protection domain

COMP7840 OSDI Current OS Research 99

The Proposed Solution (cont'd)

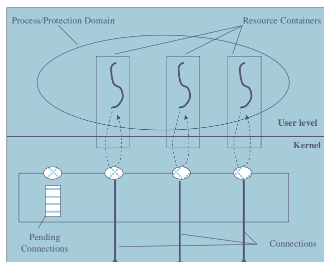
- Since resource containers are separate from processes, they may be associated with arbitrary computations
  - Like network connections to a server
- With resource containers, the multi-threaded server can be used (for efficiency) and *client* resource use can be managed
  - via their connections (one container per client)

COMP7840 OSDI Current OS Research 100

Resource Container Operations

- Create a Container: processes create containers as needed and may manage several at once. (`fork` → new container)
- Set Container's Parent: containers may be nested (e.g resources assigned to 1 container shared by its children)
- Sharing Containers: containers may be passed and shared between processes at will
  - Hence, an application may be executed in >1 protection domain
- Container Release: containers are reclaimed once no one is using them
- Container Attributes: processes can exert control over containers (control resources "in" them) by setting attributes
- Container Usage Information: processes can collect usage information from containers (e.g. for charging purposes)

Server with Resource Containers



Single Process Multithreaded Server *with* *Resource Containers* (efficient and can manage/charge individually for each client)

Resource Container Conclusions

- Resource containers provide an effective mechanism for addressing the problem
- Separation of resource management from protection domain makes sense
- Resource containers may be efficiently implemented
- Resource containers also provide side benefits
  - E.g. protections against SYN-flooding