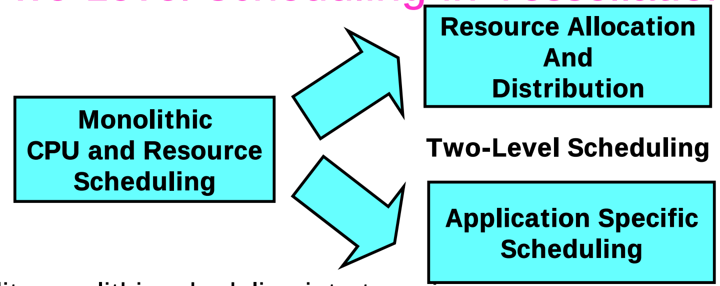
Tessellation OS

Interfaces and Mechanisms for Two-Level Scheduling

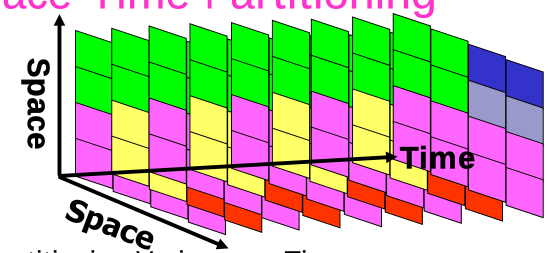
John Kubiatowicz
UC Berkeley
kubitron@cs.berkeley.edu

Two Level Scheduling in Tessellation



- Split monolithic scheduling into two pieces:
  - Course-Grained Resource Allocation and Distribution
    - Chunks of resources (CPUs, Memory Bandwidth, QoS to Services) distributed to application (system) components
    - Option to simply turn off unused resources (Important for Power)
  - Fine-Grained Application-Specific Scheduling
    - Applications are allowed to utilize their resources in any way they see fit
    - Other components cannot interfere with their use of resources
- Do all tasks fit into this model?
  - What about all the best-effort demons/processes in system?
  - Package them up as a unit and give resources to them as group

Space-Time Partitioning

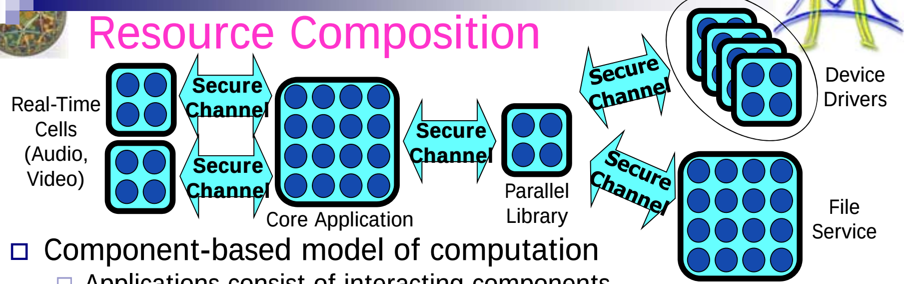


- Spatial Partitioning Varies over Time
  - Partitioning adapts to needs of the system
  - Some partitions persist, others change with time
  - Further, Partititions can be Time Multiplexed
    - Services (i.e. file system), device drivers, hard realtime partitions
- Controlled Multiplexing, *not* uncontrolled virtualization
  - Multiplexing at coarser grain (100ms?)
  - Schedule planned several slices in advance
  - Resources gang-scheduled, use of affinity or hardware partitioning to avoid cross-partition interference
- Scheduling of resources done *proactively* as possible
  - Even for unpredictable events, set up what will happen

Defining the Partitioned Environment

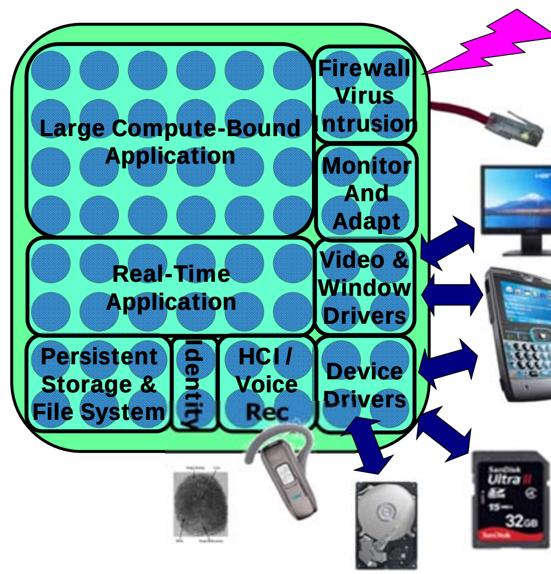
- Our new abstraction: Cell
  - A user-level software component, with guaranteed resources
  - Is it a process? Is it a Virtual Private Machine? Neither, Both
  - Different from Typical Virtual Machine Environment which duplicates many Systems components in each VM
- Properties of a Cell
  - Has full control over resources it owns ("Bare Metal")
  - Contains at least one address space (memory protection domain), but could contain more than one
  - Contains a set of secured channel endpoints to other Cells
  - Contains a security context which may protect and decrypt information
  - Interacts with trusted layers of Tessellation (e.g. the "NanoVisor") via a heavily Paravirtualized Interface
    - E.g. Manipulate address mappings without knowing format of page tables
- When mapped to the hardware, a cell gets:
  - Gang-schedule hardware thread resources ("Harts")
  - Guaranteed fractions of other physical resources
    - Physical Pages (DRAM), Cache partitions, memory bandwidth, power
  - Guaranteed fractions of system services

Resource Composition



- Component-based model of computation
  - Applications consist of interacting components
  - Produces composable: Performance, Interfaces, Security
- CoResident Cells ⇒ fast inter-domain communication
  - Could use hardware acceleration for fast secure messaging
  - Applications could be split into mutually distrusting partitions w/ controlled communication (echoes of μKernels)
- Fast Parallel Computation within Cells
  - Protection of computing resources not required within partition
    - High walls between partitions ⇒ anything goes within partition
    - Shared Memory/Message Passing/whatever within partition
- Natural Extension to Cloud
  - Services can either be local or remote

Tessellation: The Exploded OS



- Normal Components split into pieces
  - Device drivers (Security/Reliability)
  - Network Services (Performance)
    - TCP/IP stack
    - Firewall
    - Virus Checking
    - Intrusion Detection
  - Persistent Storage (Performance, Security, Reliability)
  - Monitoring services
    - Performance counters
    - Introspection
  - Identity/Environment services (Security)
    - Biometric, GPS, Possession Tracking
- Applications Given Larger Partitions
  - Freedom to use resources arbitrarily

Guaranteed Resources

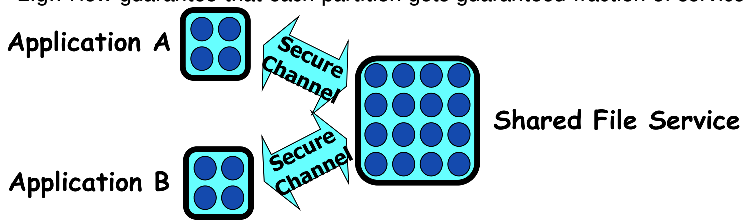
- So – what sort of things might we want to guarantee?
  - Number of processors/fraction of processor time
  - Memory BW, Cache, Network BW (needs HW)
  - Access to accelerator resources (GPU, Crypto, etc)
  - QoS to services (Shared Lib, FileSystem, DB server, Cloud Services)
- What might we put into our Service Level Agreements (SLAs)?
  - Can we use Internet services as a model?
  - Examples:
    - Guarantees of BW (say data committed to Cloud Storage)
    - Guarantees of Requests/Unit time (DB service)
    - Guarantees of Latency to Response (Deadline scheduling)
  - What level of guarantee?
    - Hard Guarantee? (Hard to do)
    - Soft Guarantee? (Better than existing systems)
      - With high confidence (specified), Maximum deviation, etc.
- Impedance-mismatch problem
  - The SLA guarantees properties that programmer/user wants
  - The *resources* required to satisfy SLA are not things that programmer/user really understands

How to Adhere to SLAs for Services?

- First question: what is 100%?
  - Available network BW depends on communication pattern
    - e.g. transpose pattern vs nearest neighbor in mesh topology
  - Available DB bandwidth depends on number of processors and I/O devices assigned to service.
  - Available disk BW depends on ratio of seek/sequential
  - Need static models or training period to discover how service properties vary with resources
- Second question: How to enforce SLA?
  - Need way to restrict users of service to prevent DOS
    - e.g. Consumer X receives designated fraction of service because we prevent consumers Y and Z from overusing service
  - May need to grow resources quickly if cannot meet SLA
    - However, this provides challenge because it may take resources away from others

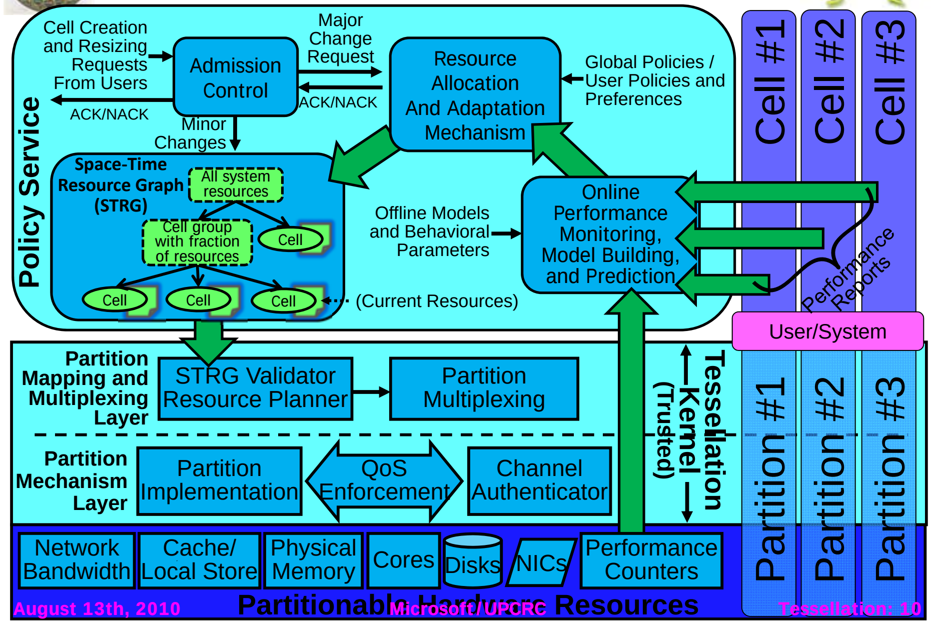
It's all about the communication

- We are interested in communication for many reasons:
  - Communication crosses resource and security boundaries
  - Efficiency of communication impacts (de)composability
- Shared components complicate resource isolation:
  - Need distributed mechanism for tracking and accounting of resources
    - E.g.: How guarantee that each partition gets guaranteed fraction of service?



- How does presence of a message impact Cell activation?
  - Not at all (regular activation) or immediate change (interrupt-like)
- Communication defines Security Model
  - Mandatory Access Control Tagging (levels of information confidentiality)
  - Ring-based security (enforce call-gate structure with channels)

Resource Allocation Architecture



What is in the Cell ABI?

- The set of interfaces between software component and system
  - Perhaps "Cell or Component Binary Interface" (CBI)?
- Interfaces with User-Level Runtime
  - Start partition, stop partition, resource removal request
  - Standard format/location for processor state storage
    - Allow suspension/resumption of partition by Tessellation
    - Allow resumption of partition with less processor resources than before
- User-level event delivery mechanisms (like user-level interrupt)
  - Deliver interrupts directly to User-Level runtime
    - One or more queues of events
    - Ability to perform user-level disable of event delivery
  - Message arrival, timer expiration, page faults
  - Designated receiver within partition (since channels are Cell ⇔ Cell)
- Channel interfaces
  - Connect with named service (either local or remote)
  - Message transmission, Reception options (interrupt, interrupt on Cell active, polled)
  - SLA request, Return

Cell ABI (Con't)

- Paravirtualized machine interfaces
  - Access to page mappings, construction of address spaces
  - User-level interrupt disable mechanisms
- Cell spawning/Dynamic Library Interfaces
  - "Adaptive Task"
  - Support for dynamic adaptation and autotuning
    - i.e. SEJITS
    - Ability to perform performance tests, compilation
    - Ability to access cached pre-tuned versions of code (local or cloud)
    - On-the-fly linking into running binary
- Resource specification and reporting interfaces
  - Requirements, Policies
  - Progress Reporting / Progress expectation
  - Promise (SLA) return
  - Resource Revocation/Renegotiation
  - Information about what is about to happen

Some Objections/Philosophy

- Isn't the Cell model a "Death by 1000 Knobs?"
  - Adds 1000s of knobs (timing and quantity of resource distribution to Cells)
  - Same problem as Exokernel
    - Would anyone actually write their own app-specific libOS?
- Ans: Parallel programming hard enough without unpredictability
  - Parallel projects of 1990s generated whole PhDs on tuning single parallel apps
  - Of course, UPCRC is all about fixing this problem, but Cell-model can help!
- Ans: Real-time is very hard with unpredictable resources
- Ans: Advancement in mechanisms helps policy (Knob) problem
  - By removing unpredictable multiplexing of resources, gain predictability of behavior
    - Mechanisms to provide a clean Cell model not fully available in today's OSes
    - Different policy/mechanism separation from today's systems
  - Task model associates resources with particular tasks
  - Benefit of Cell model must outweigh disadvantages
    - Clear "graceful degradation" to more standard use of resources
- Ans: Resources are central to many modern systems
  - E.g. battery life, Video BW, etc.

Responsiveness vs QoS vs Efficiency

- Is Performance-Isolation worth forcing idle resources?
  - Yes. In the manycore world (even multicore world) we have "excess" processors (and other resources).
  - Yes. We have very diverse application requirements
  - Yes. Systems are already power/heat/battery limited
    - Don't want to power on everything anyway
  - Contrarian argument: No, because it wastes resources
    - Clearly, there is a balance to be struck – remember "graceful degradation"
- Is Responsiveness contrary QoS/Performance-Isolation?
  - Not really – they are different sides of same coin
    - QoS/Performance isolation is about guaranteed resources in continuous use
    - Responsiveness is about guaranteed resources used intermittantly
  - Difference is really about efficiency
    - We need a way to pre-allocate/pre-reserve resources to guarantee responsiveness, then be able to use them for something else
    - Unfortunately, you cannot always retrieve resources quickly
  - Important Idea:
Pre-plan how resources will be redistributed when event arrives
- How to increase predictability of unpredictable apps (GUIs)
  - Divide app into part that is unpredictable but must be responsive and
  - QoS-assured piece (i.e. physics engine)

What about Virtual Memory?

- Why is this a question?
  - Demand paging is contrary to two-level scheduling philosophy
    - Causes unpredictable resource availability (namely memory)
    - Ties together I/O and memory in strange way
  - Demand paging kills performance in way that is opaque to runtime
  - Ideally: give memory as resource, I/O as resource
    - Let user-level scheduler do what it wants, with full information
- Seems like a no-brainer, but many an argument about this…!
  - Assignable Resources:
    - Chunks of physical memory
    - Chunks of address space (virtual memory)
  - Mechanism to allow runtime to assign virtual⇒physical
    - assign regions of physical address space to partitions,
give user-level runtime full control over page table,
hard memory fault if processor within partition goes outside bounds
- Advantages?
  - Runtime can now choose to "page or not to page"
  - Runtime can overlay or otherwise manage memory in app-specific way
  - When thread or other entity in partition hits page-fault, user-level scheduler can decide what to do
    - Better than Scheduler Activations?

Virtual Memory (Con't)

- What interfaces to give to runtime?
  - Translation assignment interfaces:
    - Paravirtualized system call to perform assignment
    - OR: hardware to separate protection (of physical memory) from translation, give user-level full control over page tables
    - Fake previous solution using virtual-machine support of current processors
  - User-level event delivery mechanism for page-faults
    - Same for delivering timer-interrupts (for scheduler) and channel-delivery events
- But what about:
  - Fragmentation of physical memory?
    - TLB on edge of chip to translate (Cell ID + physical) to real DRAM physical at coarse granularity (1MB, 16MB?)
      - Can have normal TLB under control of user-level for paging
      - Processors could run untranslated mode or with large pages for lower power
  - Protecting of code in Cell from itself or other components within Cell
    - More than one address space/Cell
    - Only "primary" (first address space) can change translation ("3 rings of protection")
- Bottom-line:
  - Can link application-specific demand-paging runtime, if desired
    - Most resource-controlled Cells will not demand page, may swap/overlay
    - Cell devoted to "standard processes" probably will have user-level runtime devoted to demand paging

Discussion

- How to divide application into Cells?
  - Cells probably best for coarser-grained components
    - Fine-grained switching between Cells antithetical to stable resource guarantees
  - Division between Application components and shared OS services natural (obvious?)
    - Both for security reasons and for functional reasons
  - Division between types of scheduling
    - Real-time (both deadline-driven and rate-based), pre-scheduled
    - GUI components (responsiveness most important)
    - High-throughput (As many resources as can get)
    - Stream-based (Parallelism through decomposition into pipeline stages)
- What granularity is best for Policy Service?
  - Fewer Cells in system leads to simpler optimization problem
- Language-support for Cell model?
  - Task-based, not thread based
  - Cells produced by annotating Software Frameworks with QoS needs?
  - Cells produced automatically by just-in-time optimization?
    - i.e. Selective Just In Time Specialization or SEJITS

Policies "User" might want to express

- Need progress X on measurement Y
  - i.e. need 5 frames/second (where frame rate measured by application)
- When Battery below 20%, slow usage of everything but application Z
  - i.e. below 20%, only voice calls work normally
- When in location X, give higher priority to Y over Z
  - i.e. when in car, higher priority to GPS than web browser
- Tradeoffs between types of apps:
  - Video quality more important than email poll rate
  - Should always be able to make 911 calls
  - Whatever happens, I want my battery to last until midnight
- Profile managers for new Android phones very interesting
  - Allow user-visible properties (ringtones, screen brightness, volume, even whole apps) to be set based on situations
  - Possible situational information:
    - GPS location, battery power, docked/not docked, time, user profile selection, …
  - (sorry for bringing up rival platform, but always good to know what is happening out there)

Opportunities for Collaboration

- We would love to decompose applications into Cells
  - Telemersion application seems very natural here
  - Browser natural as well
- We are in the process of designing our Policy layer
  - What do the SLAs actually look like?
  - What sort of adaptive resource distribution mechanism make sense?
    - Burton Smith/Sarah Bird with convex optimization
    - Rule-based policy assignment
    - Others?
- Would like to make sure that channels with:
  - UIUC, Microsoft, Intel

Conclusion

- Space-Time Partitioning: grouping processors & resources behind hardware boundary
  - Two-level scheduling
    1) Global Distribution of resources
    2) Application-Specific scheduling of resources
  - Bare Metal Execution within partition
  - Composable performance, security, QoS
- Cells: Basic Unit of Resource and Security
  - User-Level Software Component with Guaranteed Resources
  - Secure Channels to other Cells
- Partitioning Service
  - Explicit Admission Control: Sometimes requests for resources must be denied
  - Policy-driven optimization of resources
- Tessellation OS
  - Exploded OS: spatially partitioned, interacting services
  - Exploit Hardware partitioning mechanisms when available
  - Components
    - Partitioning Mechanisms ("NanoVisor")
    - Policy Service: Resource Management
    - OS services as independent servers