Reinventing Operating Systems for the Manycore Ubiquitous Swarm

John Kubiatowicz
kubitron@cs.berkeley.edu
The Swarm of Resources

Cloud Services

The Local Swarm

Enterprise Services

- What system structure required to support Swarm?
  - Discover and Manage resource
  - Integrate sensors, portable devices, cloud components
  - Guarantee responsiveness, real-time behavior, throughput
  - Self-adapting to adjust for failure and performance predictability
  - Uniformly secure, durable, available data
Support for Applications

• What support do we need for new Swarm applications?
  – Should we just port Linux, Android, or Windows 7?
  – A lot of functionality, hard to experiment with, possibly fragile, ...

• Clearly, these applications will contain:
  – Direct interaction with Swarm and Cloud services
    • Potentially extensive use of remote services
    • Serious security/data vulnerability concerns
  – Real Time requirements
    • Sophisticated multimedia interactions
    • Control of/interaction with health-related devices
  – Responsiveness Requirements
    • Provide a good interactive experience to users
  – Explicitly parallel components
    • However, parallelism may be “hard won” (not embarrassingly parallel)
    • Must not interfere with this parallelism

• No existing OS handles all of these well....
Changing the Structure of Operating Systems (and the Application that run on them)
Guaranteeing Resources

• What might we want to guarantee?
  – Examples:
    • Guarantees of BW (say data committed to Cloud Storage)
    • Guarantees of Requests/Unit time (DB service)
    • Guarantees of Latency to Response (Deadline scheduling)
    • Guarantees of maximum time to Durability in cloud
    • Guarantees of total energy/battery power available to Cell

• What level of guarantee?
  – Firm Guarantee (Better than existing systems)
    • With high confidence (specified), Maximum deviation, etc.

• What does it mean to have guaranteed resources?
  – A Service Level Agreement (SLA)?
  – Something else?

• “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
Space-Time Partitioning

- **Spatial Partition:** Performance isolation
  - Each partition receives a vector of basic resources
    - A number HW threads
    - Chunk of physical memory
    - A portion of shared cache
    - A fraction of memory BW
    - Shared fractions of services

- **Partitioning varies over time**
  - Fine-grained multiplexing and guarantee of resources
    - Resources are gang-scheduled
  - Controlled multiplexing, not uncontrolled virtualization
  - Partitioning adapted to the system’s needs
New OS Primitive: the Cell

- **Cell Properties:**
  - A user-level software component, with guaranteed resources
  - Explicit security context which allows access to protected data
  - Knowledge of how to adapt itself to new environments
  - Checkpoint/restart to provide fault tolerance, mobility and adaptation

- **Execution Environment:**
  - Explicitly parallel computation
  - Resource Guarantees
  - Trusted computing base
  - Secure channels (intra/interchip) with ability to suspend and restart during migration
Applications Composed of Interconnected Cells

- Component-based model of computation
  - Applications consist of interacting components
  - Components may be local or remote
- Communication defines Security Model
  - Channels are points at which data may be compromised
  - Channels define points for QoS constraints
  - Question: Can we provide proofs of correctness on inter-cell protocols?
- Naming process for initiating endpoints
  - Need to find consistent version of library code (within cell)
  - Need to find compatible remote services
  - Continuous adaptation: links changing over time!
Impact on the Programmer

- Connected graph of Cells ↔ Object-Oriented Programming
  - Lowest-Impact: Wrap a functional interface around channel
    - Cells hold “Objects”, Secure channels carry RPCs for “method calls”
    - Example: POSIX shim library calling shared service Cells
  - Greater Parallelism: Event triggered programming

- Shared services complicate resource isolation:
  - How to guarantee that each client gets guaranteed fraction of service?
  - Distributed resource attribution (application as distributed graph)

- Communication defines Security Model
  - SecureCell: Keys as resource – Outside entity handles privacy concerns
  - Mandatory Access Control Tagging (levels of information confidentiality)
Allocation of Resources
Discovery, Distribution, and Adaptation
Two Level Scheduling: Control vs Data Plane

- Split monolithic scheduling into two pieces:
  - Course-Grained Resource Allocation and Distribution to Cells
    - Chunks of resources (CPUs, Memory Bandwidth, QoS to Services)
    - Ultimately a hierarchical process negotiated with service providers
  - Fine-Grained (User-Level) Application-Specific Scheduling
    - Applications allowed to utilize their resources in any way they see fit
    - Performance Isolation: Other components of the system cannot interfere with Cells use of resources
Adaptive Resource-Centric Computing (ARCC)

Resource Allocation (Control Plane)
- Partitioning and Distribution

Runnin System (Data Plane)
- Application 1
  - Channel
  - GUI Service
    - QoS-aware Scheduler

- Application 2
  - Channel
  - Block Service
    - QoS-aware Scheduler

- Network Service
  - QoS-aware Scheduler

Performance Reports
- Resource Assignments
- Resource Allocation (Control Plane)
- Observation and Modeling
- Observation and Modeling
- Performance Reports
Brokering Service: The Hierarchy of Ownership

- Discover Resources in “Domain”
  - Devices, Services, Other Brokers
  - Resources self-describing?
- Allocate and Distribute Resources to Cells that need them
  - Solve Impedance-mismatch problem
  - Dynamically optimize execution
  - Hand out Service-Level Agreements (SLAs) to Cells
  - Deny admission to Cells when violates existing agreements
- Complete hierarchy
  - Throughout world graph of applications
Architecture of Tessellation OS

Policy Service

- Admission Control
- Resource Allocation and Adaptation Mechanism

Space-Time Resource Graph (STRG)

- All system resources
- Cell group with fraction of resources
- Current Resources

Partition Mapping and Multiplexing Layer

- STRG Validator
- Resource Planner

Partition Mechanism Layer

- Partition Implementation
- QoS Enforcement
- Channel Authenticator

Partitionable (and Trusted) Hardware

- TPM/Crypto
- Network Bandwidth
- Cache/Local Store
- Physical Memory
- Cores
- NICs
- Performance Counters

Major Changes

- Request ACK/NACK

Minor Changes

- Request ACK/NACK

User/System

- Cell #1
- Cell #2
- Cell #3

Performance Reports

Global Policies/User Policies and Preferences

Offline Models and Behavioral Parameters

Online Performance Monitoring, Model Building, and Prediction

Resource Allocation and Adaptation Mechanism

- Global Policies/User Policies and Preferences
- Offline Models and Behavioral Parameters

Online Performance Monitoring, Model Building, and Prediction

Partition Multiplexing

- Kernel (Trusted)

Partition #1
Partition #2
Partition #3
Example of Wrapping Resource In Service-providing Cell
The Tesselation GUI Service

- Operate on user-meaningful “actions”
  - E.g. “draw frame”, “move window”
- Service time guarantees (soft real-time)
  - Differentiated service per application
  - E.g. text editor vs video
- Performance isolation from other applications
Experimental Data

Out of 4000 frames
On Toward the Swarm
Secure Cell: Portable Secure Data

- Data divided into **globally-addressable** capsules
  - Addressable by unique GUID and/or metadata search
  - Conceptually stored in THE Storage Cloud (cyberspace?)
    → If you can name it, you can use it!
- Secure Cell: Security Context as a resource
  - Data is signed, has attached policy, Optionally encrypted
  - Unwrappable only in correct trusted environment
- Key Distribution ⇒ resource management

**Signature, Policy, Version, GUID**

**Data Jail (free use according to policy)**

**Trusted HW/SW**

**Distributed Public Key Infrastructure**

**Challenge/Response**

**Encrypt**

**Decrypt**

**Distributed Public Key Infrastructure**

**Trusted HW/SW**

**Data Jail (free use according to policy)**

**Encrypt**

**Decrypt**
Cell as Ubiquitous Primitive for the Swarm?

- Every component in system should host Cells?
  - Even sensors!?  
  - What is minimal support?
    - Security Primitives
    - Communication support
- Alternative: Bare sensors do not host Cells
  - Requires minimal computational capability
- Mobile Environment
  - Constant changes in resource availability
  - Adaptation in resource requirements
  - Must change connected graph continuously ⇒ Dynamic SLAs?
- Hierarchical Resource Broker Architecture
  - Many overlapping broker domains
  - New APIs for service providers and intra-Broker communication
  - Resource Ontologies?
- Cell is a natural way to handle heterogeneity
  - From the outside: export services to other Cells
Conclusion

• Adaptive Resource-Centric Computing
  – Use of Resources negotiated hierarchically
  – Underlying Execution environment guarantees QoS
  – New Resources constructed from Old ones:
    • Aggregate resources in combination with QoS-Aware Scheduler
    • Result is a new resource that can be negotiated for
  – Continual adaptation and optimization

• Important components of future OS environment
  – Cells as Basic Unit of Resource and Security
    • User-Level Software Component with Guaranteed Resources
    • Secure Channels to other Cells
  – Observation, Monitoring, and Adaptation layers
    • Machine learning, Convex Optimization
  – Portable Secure Data infrastructure
    • If you can name it, you can use it

• Tessellation OS: http://tessellation.cs.berkeley.edu