Tessellation OS: an OS for the Swarm

John Kubiatowicz
kubitron@cs.berkeley.edu

Requirements for Swarm OS?
- What system structure required to support Swarm?
  - Integrate sensors, portable devices, cloud components
  - Guarantee responsiveness, real-time behavior, throughput
  - Services with guaranteed behavior, self-adapting to adjust for failure and performance predictability
  - Uniformly secure, durable, available data

Swarm app store
Middleware and services
Swarm-OS
Distributed Sense-Control-Actuate Platforms
Innovative Devices and Materials

Today's Software Reality
- Resources not well managed: QoS hard to achieve
  - 20th-century notions of utilization and resource virtualization
  - Despite a cornucopia of resources - we still cannot get the ones we need when we need them!
- Services not easily interconnected
  - Every service has a unique API
  - Highly-specialized “stovepipes” often do not provide exactly what users are looking for ⇒ they end up integrating “by hand”
  - Tradeoffs between client and cloud not easy to achieve
- Too many things explicitly depend on location:
  - Where: is my data stored? (oops – it was there!)
  - Where: can I execute this piece of functionality?
  - Where: can I display this information?
  - Where: did I start this job (because I have to finish it there)
- And others don’t properly depend on location:
  - Here I am: do something about it!

Changing the Structure of Operating Systems (and the Application that run on them)
Guaranteed 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
    - A portion of physical memory
    - A portion of shared cache
    - A fraction of memory bandwidth

- 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 (SEJITS)
  - 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
  - Solution of version constraint problem for running application
Two Level Scheduling in Tessellation OS

- 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

Modeling and Adaptation

- Modeling of Applications
  - Static Profiling: may be useful with Cell guarantees
  - Multi-variable model building: Get performance as function of resources
- Adaptation according to User and System Policies
  - Convex optimization
    - Relative importance of different Cells expressed via scaling functions
  - Walk through Configuration space
    - Meet minimum QoS properties first, enhancement with excess resources

Architecture of Tessellation OS

- Resource Allocation and Distribution
- Application Specific Scheduling

- Monolithic CPU and Resource Scheduling

- Two-Level Scheduling

- Resource Allocation and Adaptation Mechanism

- Online Performance Monitoring, Model Building, and Prediction

- Policy Service

- STRG Validator

- Resource Planner

- Partition Mapping and Multiplexing Layer

- Partition Mechanism Layer

- Partition Implementation

- QoS Enforcement

- Channel Authenticator

- Performance Counters

- Partitionable (and Trusted) Hardware

Tessellation on Multicore

- Cloud Storage BW QoS

- QoS Guarantees

- QoS Guarantees

- QoS Guarantees

- Compute-Bound Application

- Network QoS Monitor and Adapt

- Persistent Storage & Parallel File System

- Disk I/O Drivers

- Other Devices

- Monitor and Adapt

- Network QoS

- Compute-Bound Application

- I/O-Bound Application

- Persistent Storage

- Parallel File System

- Network QoS

- Other Devices

- Disk I/O Drivers

- Compute-Bound Application

- I/O-Bound Application

- Persistent Storage

- Parallel File System
Swarm Data

- Information as a First Class Citizen:
  - Current Viewpoint: Data is byproduct of computation
  - Much Better: Data independent of computation, outlasts computation, transformed by computation
  - Computation should be the ephemeral thing!
- Fallacy: Data Resides in a Particular Location
  - A breach of the system results in loss of privacy
  - Incorrect security configuration results in loss of integrity
  - A crash results in loss of updates or new information
  - Transient routing failure results in inaccessibility
  ⇒ Integrated, Secure, Deep Archival Storage
  - Data available from anywhere, anytime
  - Data encrypted all the time (except in authorized cells)
  - Data durable by default (coding, widespread replication)

Start with Truly 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!
- Integrated protection
  - Data is signed, has attached policy, Optionally encrypted
  - Unwritable only in correct trusted environment
  - Legacy support: data without secure metadata

Location Independent Data and Services

- Level of indirection in network
  - "Decentralized Object Location and Routing" (DOLR)
  - All data and services explicitly named by secure hash (Sha256?)
- Deep Archival Storage in Cloud
  - Integrated use of coding for maximum durability
The Terrestrial Petri Dish

Cloud Storage

Service A (Cloud)

Locality-Aware Routing Network

Service B (Cloud)

Applications in 2020

• Hierarchical Resource Management
  – Multi-level resource reservation and adaptation
  – Resource guarantees and QoS at all levels
• Synthesis of Tasks from Preexisting Services
  – Transparent and adaptive choice of service providers
• Permanent Secure Archival Storage

Conclusion

• Essential ideas:
  – Resource guarantees negotiated hierarchically
  – Continual adaptation and optimization
  – Deep Archival Storage available from anywhere, anytime
  – Mobility of secure data, computation (is there a difference?)
• 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