Java, Peer-to-Peer, and Accountability: Building Blocks for Distributed Cycle Sharing

Ali Raza Butt
Xing Fang
Y. Charlie Hu
Samuel Midkiff
The need for sharing compute-cycles

- Scientific applications
  - Complex, large data sets

- Dedicated resources
  - Expensive

- Modern workstation
  - Powerful resource
  - Available in large numbers
  - Underutilized

→ Harness idle-cycles of network of workstations
Current cycle-sharing schemes

• Examples: SETI@Home, Distributed.net, Entropia

• Use centralized application servers
  ▪ Performance bottleneck
  ▪ Single point of failure

• Applications are explicitly trusted
  ▪ Introduce a plethora of security problems

• Users contribute compute-cycles
  ▪ Individuals cannot utilize the shared cycles
Cycles-sharing for All!

- Goal: all participants can utilize the system

Challenges:
- Resource discovery and management
- Portability
- Safety
- Security
- Fairness

Our solution:
- Exploit existing peer-to-peer networking
- Leverage Java Virtual Machine Sandboxing
- Add the ability to remotely monitor Java program progress
- Develop distributed credit based accountability
Agenda

• Background
• Discovering resources
• Ensuring fairness
• Design & Implementation
• Evaluation
• Conclusions
Background: Overlay Networks

P2P networks are self-organizing overlay networks without central control.
Background: structured p2p overlays

- Overlays with imposed structure
  - Each node has a unique random nodeId
  - Each message has a key
  - The nodeId and key reside in the same name space

- Routing: Takes a message with a key and sends it to a unique node

- Implements Distributed Hash Table (DHT) abstraction

- DHT abstraction is preserved in the presence of node failure/departure
Properties of structured p2p networks

- Scalable
- Self-organizing
- Fault-tolerant
- Locality-aware
- Simple to deploy

- Many implementations available
  - E.g. Pastry, Tapestry, Chord, CAN…
Example: Pastry

- 128-bit circular identifier space

- Routing: A message is routed reliably to a node with $\text{nodeId}$ numerically closest to the key

- Locality-aware
  - Routing table has $O(\log N)$ entries matching increasingly long prefix of local $\text{nodeId}$
Agenda

• Background
• Discovering resources
• Ensuring fairness
• Design & Implementation
• Evaluation
• Conclusions
Resource availability information

- Announcements to nearby nodes
  - Contain resource characteristics and availability information
  - Leverage locality-aware routing table
  - Soft state
    - Periodically refreshed
Resource announcements are physically close to
Execution node selection

• Utilize local resource availability information
• Query nearby nodes for job execution
  ▪ Proximity
  ▪ Credit-worthiness
• Request remote execution
Agenda

• Background
• **Discovering resources**
• Ensuring fairness
• Design & Implementation
• Evaluation
• Conclusions
Fairness in cycle-sharing

- More complex than fairness in storage sharing [Samsara: SOSP 2003]
  - Cycles are perishable resources

- Challenge
  - Mutual guarantees for submitting and contributing nodes

- Our Solution:
  - VM and compiler instrumented code for progress monitoring
  - DHT based feedback system to report unfair nodes
  - Assumption: nodes act in their own self-interest
Job progress monitoring

- System leverages existing *Instrumentation Sampling Framework*

- A thread periodically retrieves contents of Method Invocation Counters

- VM communicates progress (using *beacons*) asynchronously to the *Reporting Module*
Monitoring setup

- Reporting module
  - Provides submitter with job monitoring capability
  - Decouples design of beacons from that of query
  - Provides asynchronous job monitoring
Distributed credit feedback system

- Ensure compensation for consumed cycles

- Tradable credit-reports
  - Digitally signed
  - Un-forgable

- DHT based distributed credit tracking
  - Allows a node’s transactions to be checked by other nodes
  - Allows determination of a node’s credit-worthiness

- Credit-worthiness used to punish and reward nodes
Agenda

• Background
• Discovering resources
• **Ensuring fairness**
• Design & Implementation
• Evaluation
• Conclusions
Implementation

• Prototype implementation:
  ▪ P2p functionality using FreePastry 1.3
  ▪ DHT feedback built on PAST

• Augmented Jikes RVM
  ▪ Added new VM thread to use adaptive compiler information to monitor progress
Software modules

- P2p network
- Modified JVM with Probe support
- P2p communication
- Credit generator
- Remote job probing
- Code instrumentation engine
- Resource Manager
- P2p Storage
- Modified JVM with Probe support
Agenda

• Background
• Discovering resources
• Ensuring fairness
• Design & Implementation
• Evaluation
• Conclusions
Methodology

• Overhead measurement in a real implementation
  ▪ Overhead of beacons
  ▪ Overhead of reporting module

• Effectiveness of catching thieves in a large scale simulation
Implementation Setup

• Hardware
  ▪ Pentium 4, 2 GHz, 512MB RAM
  ▪ Linux kernel 2.4.18
  ▪ Connected via 100 Mb/s Ethernet
Overhead of beacons

![Graph showing slowdowns for different beacon durations and benchmarks.](image-url)
Overhead of the reporting module

![Bar chart showing slowdowns (percentage) for different benchmarks and latency settings. The chart includes categories for 10ms, 50ms, 200ms, 500ms, and No Beacon.]
Simulations

• 1000 Nodes setup

• Georgia Tech-Internet Topology Models (GT-ITM)
  ▪ Transit-stub model
  ▪ 100 transit domains
  ▪ 10 stub domains

• Sequence
  ▪ 100 (issue time: T, job length: L) pairs
  ▪ Interval \((T_n - T_{n-1})\), L uniform distribution \([1,17]\)
  ▪ Random overload/idle periods
Jobs issued and completed: No cheaters
Jobs issued and completed: cheaters

- Yellow line: Jobs issued
- Purple line: Jobs completed

Number of jobs vs Time graph.
Jobs issued and completed: cheaters caught
Evaluation conclusion

- The overhead of monitoring code is insignificant
- The accounting system effectively recognizes cheating nodes and restricts them
Agenda

- Background
- Discovering resources
- Ensuring fairness
- Design & Implementation
- **Evaluation**
- Conclusions
Conclusions

• Building blocks for cycle-sharing
  ▪ Peer-to-peer networks
  ▪ Java based progress monitoring and security
  ▪ Credit-based accountability mechanisms

• Ideal system for inter-organizational networks of pooled resources
Questions?