Speculative Use of Idle Resources

Lars Eggert
larse@isi.edu

November 5, 2001

Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781
USA
Topic

▶ Use idle OS capacity speculatively
▶ Current ad-hoc mechanisms interfere
  ▶ Caching + prefetching
  ▶ Process + data migration
▶ Goal: provide unified + integrated idle-time facility
  ▶ Replace ad-hoc → simplify + improve
  ▶ Enable new techniques
Outline

▶ Introduction
  ▶ Plenty of idle capacity → utilize it

▶ Resource model
  ▶ Prioritization, preemptability, isolation

▶ Open issues
  ▶ Model extensions
  ▶ Costs of idle-time use and speculation
  ▶ Impact of speculation on caches
Outline (Cont.)

► Proof-of-concept implementation
  ► Idle-time networking

► Related work
  ► Real-time systems
  ► Speculative techniques
  ► Non-speculative uses for idle-time

► Proposed research
  ► Application: idle-time NFS
Introduction

► Motivation
  ► Much idle capacity…

► Key Idea
  ► …utilize it
  ► Goal: all resources always busy

► Analogy
  ► CPU with speculative branch execution
Motivation

- Idle resources = lost opportunity
- Idle capacity available, even when system is “fully loaded”
  - 50-70% memory available
  - 70% of CPUs idle (Condor)
- Idea: use idle OS resource capacity speculatively
Key Idea

► Use idle OS capacity speculatively
► Can interfere with regular use, causing...
  ► …delay → minimize
  ► …incorrect processing → prevent
► Both must be addressed
► Similar: CPU w/speculative execution
Analogy: CPUs

- Predict control flow (speculate)
- Execute likely future instructions
- Use idle resources
  - Bus bandwidth → pre-fetching
  - Execution units → pre-execution
- Speculation cost < performance gain
- Reduce speculation cost with hardware
Model

- Speculative idle-time use requires
  1. Access to idle-time capacity
  2. Mechanism to speculative use it
- Define simple models
  - Resource use
  - OS processing
- Identify required properties for (1) and (2)

**Task:** validate model
Simple Resource Model

- OS sees resource request stream
  - “Send packet”
  - “Allocate page”
  - “Run me”
- Distinguish requests
  - FG (regular)
  - BG (idle-time)
- Minimize FG delays
  - Examples follow…

**Diagram**

- Time
- Resource 1
  - Request/Response Stream
- Resource 2
- Requests from Process A
- Requests from Process B
Principles

1. Prioritization
2. Preemptability
3. Isolation

► All 3 required…
  ▶ (1) + (2) → min: prevent FG starvation
               opt: minimize FG delay
  ▶ (3) → maintain correctness

► …for all resources
  ▶ Bottleneck resource controls behavior
Prioritization

► Never process idle-time requests while regular requests are waiting

► Priority queue
Preemptability

- Preempt active idle-time use for incoming regular requests
- Never preempt regular requests for idle-time use
- Preemptive servicing
Isolation

- Side effects of speculations remain hidden until committed or discarded
- Virtualize OS state

Before

After
OS State Virtualization

- Speculations modify **virtual state**
- Successful speculation $\rightarrow$ **atomic commit**
- Regular state update $\rightarrow$ **propagate** to virtual states
Resource Categories

- Spatially-shared
  - Capacity divided → serve >1 request
  - Allocation units leased indefinitely

- Temporally-shared
  - Not subdivided → serve 1 request
  - Full capacity leased for 1 request

- Physical devices may combine both
  - Disk: I/O temporal, storage spatial
Open Issues

- Preemption cost
- System-wide support for idle-time
- Inter-resource interference
- Capacity reallocation
- Effects on caches
- Speculative workload generation
- Integrated scheduling of speculations
Preemption Cost

- **Minimize cost**
  - Fixed overhead
  - ∑cost < ∑gain → win

- Devices often not interruptible
  - Some evidence
    - Workload bursty
    - Worst-case rare

![Diagram showing time, active, idle, and requests with X marking preemption cost.](image)
Need Model Extension

► Current model simplistic
► Does not describe
  ► Preemption cost
  ► Speculation gain
  ► Virtual resources
    ► Are themselves users of other resources
  ► Dynamic costs (stateful resource)

► Task: extend model
System-Wide Idle-Time

► All resources must support idle-time
  ► Bottleneck resource unknown
  ► Its scheduler controls overall system
► Unmodified scheduler → interference
  ► Example next slide

► Task: support other resources
  ► For idle-time NFS: disk I/O + storage
Example: CPU vs. Network

- POSIX real-time CPU scheduler extensions
  - Prioritization
  - Preemptability
- Experiment: bottleneck...
  - ...= CPU → good
  - ...≠ CPU → bad
- POSIX only effective when bottleneck = CPU
Inter-Resource Interference

- Idle-time use of one resource delays/preempts regular use on another
- Problem: kernel interrupt handling
  - Priority: Lower-layer interrupts
  - Work-conserving interrupt handlers
- Approach: vertically-structured OS

Task? evaluate “vertical” OS
Capacity Reallocation

- Reclaim (enough) idle-time for FG
  - Only for **spatially-shared** resources, e.g. disk space
- Idle-time tasks must cope
- **Task:** investigate transparent reclaim mechanisms

**Before**

- Allocation
- Requests

**After**

- Allocation
- Requests

Time:
- Before: \( t_1, t_2, t_3 \)
- After: \( t_1, t_2, t_3 \)
-effects on Caches

- Speculation → create/flush entries
  - Cache pollution → loss
  - Pre-load effect → gain
- Both observed in studies
- Safe: disable caches for speculations
  - Loose benefit, issue: hardware cache
- Task? investigate “cache = resource”
  - Partition + speculate with capacity
Speculative Task Creation

► Who generates speculative tasks?

► Ideal: automatic generation (OS)
  ► Hard: cannot look ahead
  ► CPUs can, instruction stream static
    ▶ Branch ambiguities are resolvable
  ► Resource request stream is dynamic
  ► Possible in some cases (Chang)

► Outside the scope of this proposal
Integrated Scheduling

- Multi-resource speculation
- Finish unlikely if not all available
- Availability controls speculation scheduling
- Need resource requirements
- Task? investigate integrated scheduling

- Task A uses CPU + net
- Task B uses CPU + disk
- Assume net busy
- Task B more likely to finish
Idle-Time Networking

► Proof-of-concept for model
► 3 principles applied to Internet
  ► Prioritization → two traffic classes
  ► Preemptability → abort send/receive
  ► Isolation → IP routers are stateless
► ITN for IP is simple extension
  ► FG = best-effort, BG = “effortless”
  ► Similar to diff-serv, etc.
    ► But lower-than-default priority
ITN End System Support

► ITN simple for routers (layer 3 only)
  ► Use diff-serv “expedited forwarding”
  ► Similar to some link layer mechanisms
    ► Frame Relay “discard eligible”
    ► UNI ATM red/green marking
  ► Out of scope (assume present)

► End systems process layers 3-5
  ► Use router-like layer 3 mechanisms
  ► Need new mechanisms at layers 4+5
UNIX Network Processing

- **TCP**
  - Process: snd buffer
  - Socket Layer: socket snd buffer
  - Network Layer: device snd queue
  - Hardware: NIC TX queue

- **UDP**
  - Process: send buffer
  - Socket Layer: socket snd buffer
  - Network Layer: device snd queue
  - Hardware: NIC TX queue

**CPU**
- **Transport Protocol**
- **Device IRQ**
- **Media Access**
ITN Mechanism

Drop BG when FG active
- “Active” = data queued at socket
- Simplistic + sender-side only
- TCP drop $\rightarrow$ congestion control
- UDP drop $\rightarrow$ rate reduction (maybe)

Features
- Prioritization: Yes
- Preemptability: Yes (packet-level)
- Isolation: No
ITN Implementation

▶ Proof-of-concept implementation
  ▶ FreeBSD 4.X + modified KAME/ALTQ
  ▶ Linux (CS558 students)
▶ FreeBSD experiments
  ▶ Private, switched 100Mbps Ethernet
  ▶ Parallel FG + BG sending processes
  ▶ 2 protocols: TCP + UDP
  ▶ 2 FG sender loads: 10% + 100%
ITN Results: FG 100% Load

- **FG: TCP**
- **BG: no, TCP, UDP**
- **FG: no ITN: ~50%**
- **FG ITN: ~99%**

![Graph showing TCP throughput with ITN and without ITN comparison]

- **FG: UDP**
- **BG: no, TCP, UDP**
- **FG no ITN: ~47%**
- **FG ITN: ~98%**

![Graph showing UDP throughput with ITN and without ITN comparison]
ITN Results: FG 10% Load

**FG: TCP sender**
- No ITN: <80%
- ITN: ~98%

**FG: UDP sender**
- No ITN: <20%
- ITN: ~97%

Baseline TCP BG Sender UDP BG Sender

FG TCP Throughput [Mbps]

Just FG No ITN ITN No ITN ITN

FG UDP Throughput [Mbps]

Just FG No ITN ITN No ITN ITN
Implementation Summary

► Simplistic proof-of-concept design
  ► Prioritization + preemptability
  ► No isolation
  ► Sender-only

► Still effective
  ► Isolate FG from BG to within 1-2%
    ► Depends on packet size

► Task? design isolation mechanism
Related Work

► Real-time systems
► Speculative use of idle capacity
  ► Prefetching + caching
  ► System optimizations
  ► Process/data migration
  ► Speculative execution in hardware/software
► Non-speculative uses of idle-time
  ► Maintenance tasks
Real-Time Systems

- Correctness of computation depends on **timeliness**
- Key components
  - (P) Predictability
  - (RRS) Resource requirements specification
  - (AC) Admission control
- All 3 not required for idle-time
Real-Time Systems (Cont.)

- RT systems prioritize resources
  - (RRS) + (P) → dynamic scheduling for periodic workloads
- RT prioritized schedulers OK for idle-time prioritization
- Strict preemptability not required
  - Any order OK that meets deadlines
- RT systems have no isolation
Prefetching + Caching

- **Goal: latency hiding**
  - Cache: replicas in fast storage
  - Prefetch: interleaved retrieve (then cache)

- **Ad-hoc mechanisms for idle-time + speculation**

- **Improved through proposed mechanisms**
  - Prefetch in idle capacity
  - Cache in idle storage
Speculative Optimizations

- Disk block replication in idle storage
- Disk arm movement in idle time
- Defragment FS
- “Prefetch the means” (warm caches)
  - Tragedy of the commons
- IKE negotiation + PMTU discovery
- Recompilation, dynamic optimization
- **All predict - none preempt, prioritize or consider costs**
Process + Data Migration

► Local work → remote idle times
  ► PM: Sprite, Condor, V-System, Butler
  ► DM: Mether, SETI, distributed.net

► Detect/predict remote idle times
  ► Coarse heuristics focus on 1 resource
  ► Idleness system-wide condition
  ► Fail to utilize short, transient idle times

► Only remote hosts benefit from local idle capacity
Speculative Execution

- **Hardware**
  - Simultaneous multi-threading
  - Speculative versioning cache
  - Mirage: treat bandwidth for latency

- **Software**
  - Speculative branch execution in LISP
  - Method-level speculation in Java
  - Information agents
Non-Speculative Uses

➤ System maintenance
➤ File system
  ➤ Reorganize LFS
  ➤ Check integrity
  ➤ Scan for viruses
➤ Cron jobs, net news, email…
  ➤ Try to run with idle resources
  ➤ At deadline, process regularly
Proposed Research

- Idle-time NFS
- Extends existing idle-time network
- Requires new mechanisms
  - Disk bandwidth + storage
- File system
  - more interactions + operations
  - FTP, HTTP mostly read-only
- Addresses several “tasks”
Tasks

► Proposed research
1. Extend model for costs + gains
2. Validate (extended) model
3. Idle-time support for disk I/O + storage
4. Reclaim mechanism for disk storage

► Tasks outside the thesis scope
5. Integrated scheduling
6. Design isolation mechanism
7. Evaluate vertically-structured OS
8. Treat caches as resources
Conclusion

► Use idle resources speculatively
► Principles
  ► Prioritization, preemptability, isolation
► Issues
► Idle-time networking
  ► Implementation + experiments
► Related work
► Proposed research
  ► Idle-time NFS