

## Application Performance in the QLinux Multimedia Operating System

Sundaram, A. Chandra, P. Goyal,  
P. Shenoy, J. Sahni and H. Vin

Umass Amherst, U of Texas Austin

ACM Multimedia, 2000



## Introduction

- General purpose operating systems handle diverse set of tasks
  - Conventional best-effort with low response time
    - + Ex: word processor
  - Throughput intensive applications
    - + Ex: compilation
  - Soft real-time applications
    - + Ex: streaming media
- Many studies show can do one at a time, but when do two or more grossly inadequate
  - MPEG-2 when compiling has a lot of jitter

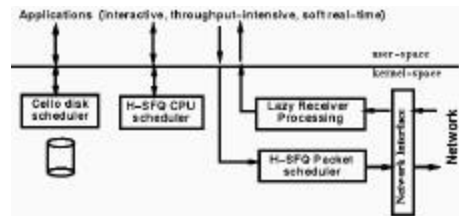


## Introduction

- Reason? Lack of service differentiation
  - Provide 'best-effort' to all
- Special-purpose operating systems are similarly inadequate for other mixes
- Need OS that:
  - Multiplexes resources in a predictable manner
  - Service differentiation to meet individual application requirements



## Solution: QLinux



- Solution: QLinux (the Q is for Quality)
  - Enhance standard Linux
  - Hierarchical schedulers
    - + classes of applications or individual applications
  - CPU, Network, Disk



## Outline

- QLinux philosophy
- CPU Scheduler
  - Evaluation
- Packet Scheduler
  - Evaluation
- Disk Scheduler
  - Evaluation
- Lazy Receiver Processing
  - Evaluation
- Conclusion



## QLinux Design Principles

- Support for Multiple Service Classes
  - Interactive, Throughput-Intensive, Soft Real-time
  - Low average response times, high aggregate throughput, performance guarantees
- Predictable Resource Allocation
  - Priority not enough (starvation of others)
  - Ex: mpeg\_decoder at highest can starve kernel
  - QLinux uses rate-based rather than priority based
    - + Weight based on rate for each:  $w_i / \sum w_j$
  - Not static partitioning since unused can be used by others



## QLinux Design Principles

- Service Differentiation
  - Within a class, applications treated differently
  - Uses hierarchical schedulers
  - Top level gives resources to class
  - In each class, can allocate resources appropriately among all applications
- Support for Legacy Applications
  - Support binaries of all existing applications (no special system calls required)
  - No worse performance (but may be better)

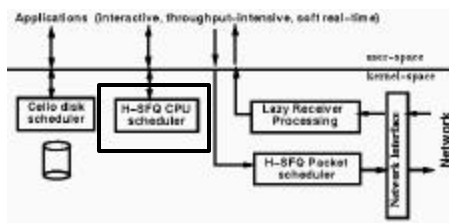


## QLinux Design Principles

- Proper Accounting of Resource Usage
  - Application level CPU easy
  - Kernel resources hard
    - + Load from interrupts difficult to charge to process
    - + Many kernel tasks are system-wide
  - *Lazy receiver processing*
    - + Defer packet processing when receiver asks
  - CPU scheduler allocation holds even when kernel uses up various amounts of CPU



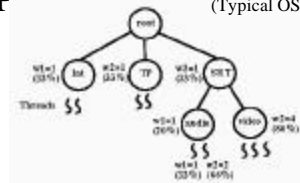
## QLinux Components



## Hierarchical Start-time Fair Queuing

(H-SFQ) CPU Scheduler (Typical OS?)

- Uses a tree
- Each thread belongs to 1 leaf
- Each leaf is an application class
- Weights are of parent class



$$B_i = \left( \frac{w_i}{\sum_j w_j} \right) * B$$

- Each node has own scheduler
- Uses Start-Time Fair Queuing at top for time for each



## H-SFQ CPU Scheduler

- Nodes can be created on the fly
- Threads can move from node to node

System call	Purpose
hsfq_mknod	create a new node in the scheduling hierarchy
hsfq_rmknod	delete an existing node from the hierarchy
hsfq_joinnod	attach the current process to a leaf node
hsfq_move	move a process to a specified child node
hsfq_parse	parse a pathname in the scheduling hierarchy
hsfq_admin	administer a node (e.g., change weights)

- Defaults to top-level fair scheduler if not specified
- Utilities to do external from application
  - Allow support of legacy apps without modifying source




## Experimental Setup (for all)

- Cluster of PCs
  - P2-350 MHz
  - 64 MB RAM
  - RedHat 6.1
  - QLinux based on Linux 2.2.0
- Network
  - 100 Mb/s 3-Com Ethernet
  - 3Com Superstack II switch (100 Mb/s)
- "Assume" machines and net lightly loaded




## Experimental Workloads

- *Inf*: executes infinite loop
  - Compute-intensive, Best effort
- *Mpeg\_play*: Berkeley MPEG-1 decoder
  - Compute-intensive, Soft real-time
- *Apache Web Server and Client*
  - I/O intensive, Best effort
- *Streaming media server*
  - I/O intensive, Soft real-time
- *Net\_Inf*: send UDP as fast as possible
  - I/O intensive, Best effort
- *Dhrystone*: measure CPU performance
  - Compute-intensive, Best effort
- *Lmbench*: measure I/O, cache, memory ... perf



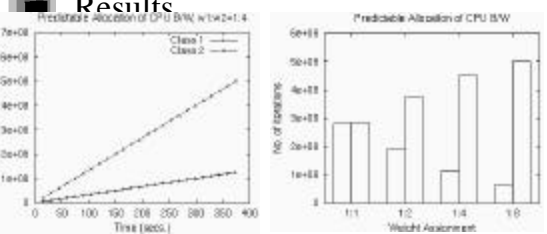
## CPU Scheduler Evaluation-1

- Two classes, run *Inf* for each
- Assign weights to each (ex: 1:1, 1:2, 1:4)
- Count the number of loops




## CPU Scheduler Evaluation-1

### Results




“count” is proportional to CPU bandwidth allocated



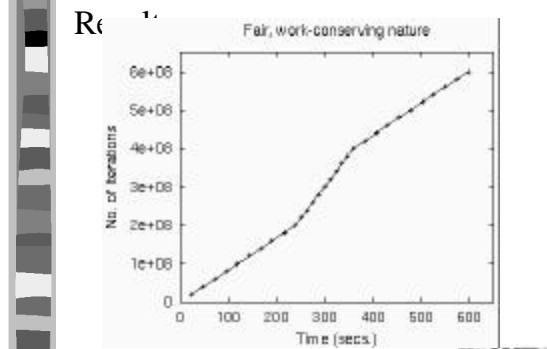
## CPU Scheduler Evaluation-2

- Two classes, equal weights (1:1)
- Run two *Inf*
- Suspend one at t=250 seconds
- Restart at t=330 seconds
- Note count




## CPU Scheduler Evaluation-2

### Results




(Counts twice as fast when other suspended)

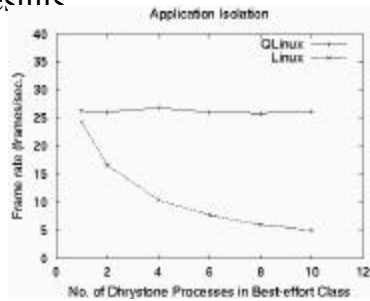


## CPU Scheduler Evaluation-3

- Two classes: soft real-time & best effort (1:1)
- Run:
  - *MPEG\_PLAY* in real-time (1.49 Mbps)
  - *Dhrystone* in best effort
- Increase Dhrystone's from 1 to 2 to 3 ...
  - Note MPEG bandwidth
- Re-run experiment with Vanilla Linux



### CPU Scheduler Evaluation-3 Results

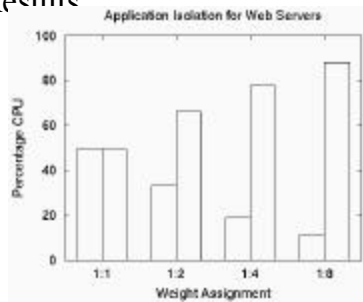


### CPU Scheduler Evaluation-4

- Explore another best-effort case
- Run two *Web servers* (representing, say 2 different domains)
- Have clients generate many requests
- See if CPU bandwidth allocation is proportional



### CPU Scheduler Evaluation-4 Results

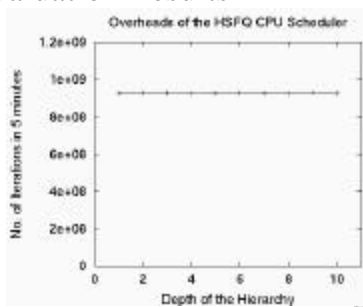


### CPU Scheduler Overhead Evaluation

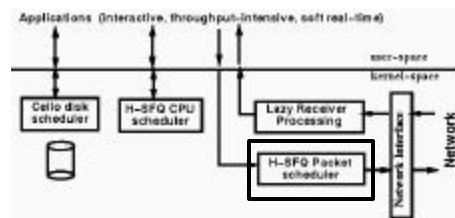
- Scheduler takes some overhead since recursively called
- Run *Inf* at increasing depth in scheduler hierarchy tree
- Record count for 300 seconds



### CPU Scheduler Overhead Evaluation Results

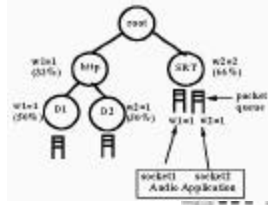


### QLinux Components



## H-SFQ Packet Scheduler

- Typical OS uses FIFO scheduler for outgoing packets
- Use H-SFQ (Fair Queue) to schedule
- Each leaf is one or more queues of packets
- Weights for queues
- Unused bandwidth to others



## H-SFQ Packet Scheduler

- Operations on the fly
- Associate with queue via setsockopt()

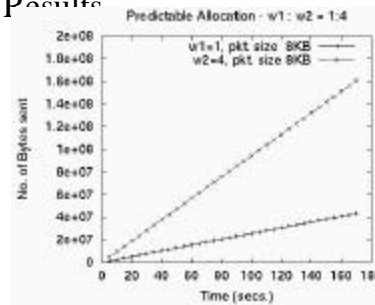
System call	Purpose
hsfqdiscinstall	install HSFQ queuing discipline of a network interface
hsfqlinklnknod	create a node in the scheduling hierarchy
hsfqlinkcreateq	create a packet queue
hsfqlinkattachq	attach a queue to a leaf node
hsfqlinkmoveq	move a queue between schedulers
hsfqlinkrmnode	delete the specified node
hsfqlinkrmq	delete the specified queue
hsfqlinkmodify	change the weight of a node/queue
hsfqlinkparamnode	parse a pathname in the scheduling hierarchy
hsfqlinkgetroot	get the ID of the root node at a particular network interface
hsfqlinkstatus	display the scheduling tree
setsockopt	attach a socket to a queue

## Packet Scheduler Evaluation-1

- Two classes using *Net\_inf*
- Run two receivers to count received packets
- 8KB packets



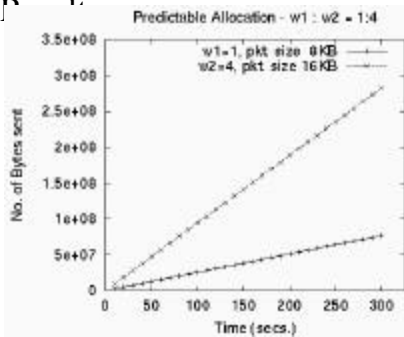
## Packet Scheduler Evaluation-1 Results



(Different packets sizes?)



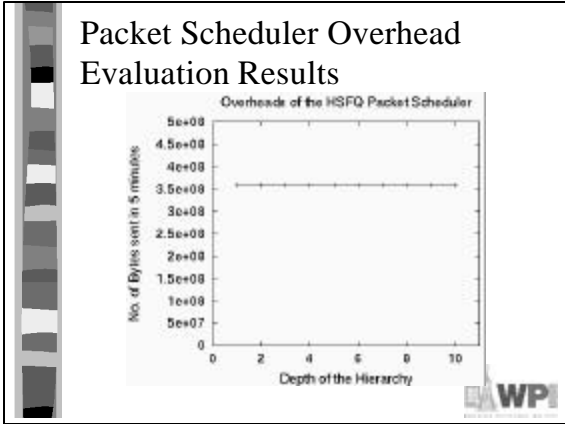
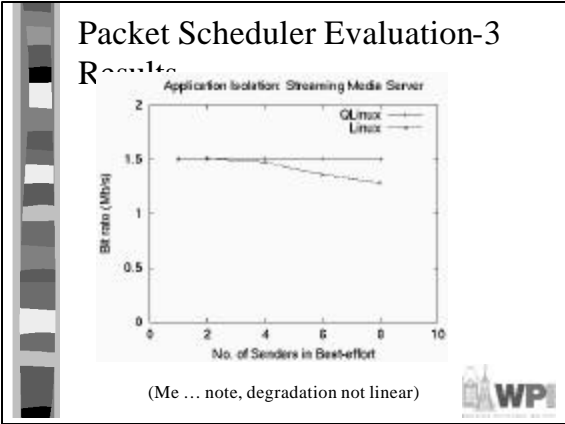
## Packet Scheduler Evaluation-2 Results



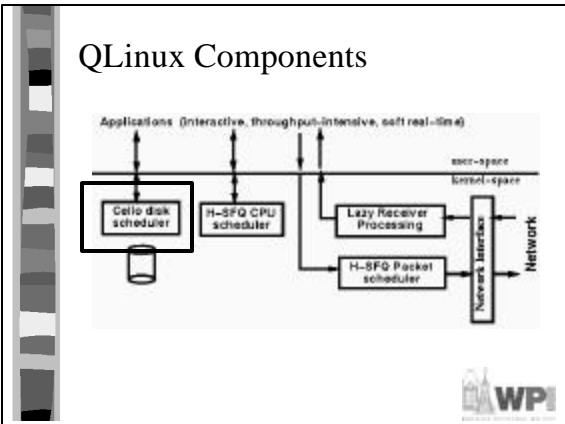
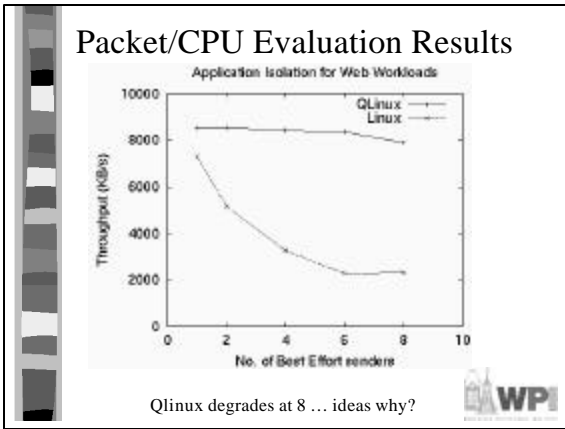
## Packet Scheduler Evaluation-3

- Real-world applicatis
- *Streaming media server* in soft real-time class
- Increasing number of *Net\_inf* apps
- Compare QLinux with Vanilla Linux





- ### Combined Packet and Scheduler Evaluation
- Web server and several I/O intensive apps
  - Two classes in CPU and Packet scheduler
    - Web server in one
    - All I/O intensive *Net\_inf* in other
  - Web server driven by trace (ClarkNet)
  - Increase number of *Net\_inf*
  - Compare to Vanilla Linux



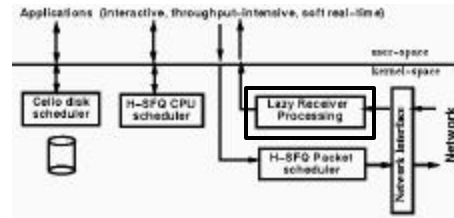
- ### Cello Disk Scheduler
- Typical OS uses SCAN for disk
  - Cello 2 levels: class independent, class specific
  - 3 classes
  - Class specific decides when and how many to move
  - Class independent puts where
  - Lastly moved FCFS
- 
- (Badri's thesis)

## Cello Disk Scheduler Evaluation

- (None in this paper)
- (Previous paper at SIGMetrics)



## QLinux Components



## Lazy Receiver Processing (LRP)

- Process *A* running
- Packet arrives for process *B*
  - Interrupt, IP, TCP, Enqueue gets charged to *A*!
- LRP postpones until process does a read
- Tricky! Some steps, e.g. TCP ack, requires it to happen right away
  - Special thread for each process for packets
- QLinux uses special queues, decodes only as far as needed
  - Special queue for ICMP, ARP ...



## LRP Evaluation and Results

- Run 2 Apache Web Servers
  - Lightly loaded, retrieve 2KB file in 51ms
- Bombard 1 server with DoS by sending 300 requests/sec
  - Other server load went to 70ms
- Re-run with Vanilla Linux
  - Other server load went to 80ms



## QLinux Total System Evaluation

- Run Imbench
  - System call overhead
  - Context switch times
  - Network I/O
  - File I/O
  - Memory performance
- QLinux vs. Vanilla Linux




## QLinux Total System Evaluation Results

Table 4: Imbench Results

Test	QLinux	Linux
syscall overhead	1 $\mu$ s	1 $\mu$ s
fork()	400 $\mu$ s	400 $\mu$ s
exec()	2 ms	2 ms
Context switch (2 proc/ 0KB)	4 $\mu$ s	1 $\mu$ s
Context switch (16 proc/ 64KB)	286 $\mu$ s	283 $\mu$ s
Local UDP latency	47 $\mu$ s	53 $\mu$ s
Local TCP latency	83 $\mu$ s	82 $\mu$ s
File create (0 KB file)	21 $\mu$ s	21 $\mu$ s
File delete (0 KB file)	2 $\mu$ s	2 $\mu$ s



- Not much overall.
- Context switch overhead, but 100 ms time slice
- QLinux untuned, so could be better







## Conclusion

- Qlinux provides
  - CPU scheduler
  - Packet scheduler
  - Disk scheduler
  - Proper I/O processing
- Provide fair and predictable allocation
- Multimedia and Web applications can benefit
- Overhead is low
- All conventional operating systems should incorporate



## Future Work

- Disk scheduler results
- Multiprocessors
- Fair allocation of other I/O interrupts
- Other devices since Cello disk specific
  - RAID, tape,



## Evaluation of Science?

- Category of Paper
- Science Evaluation (1-10)?
- Space devoted to Experiments?

