INF5071 – Performance in Distributed Systems

Server Resources:

Server Examples, Resources and CPU Scheduling

7. September 2007

Motivation

- In a distributed system, the performance of every single machine is important
  - poor performance of one single node might be sufficient to "kill" the system (not better than the weakest)
- Managing the server side machines are challenging
  - a large number of concurrent clients
  - shared, limited amount of resources

- We will see examples where simple, small changes improve performance
  - decreasing the required number of machines
  - increase the number of concurrent clients
  - improve resource utilization
  - enable timely delivery of data

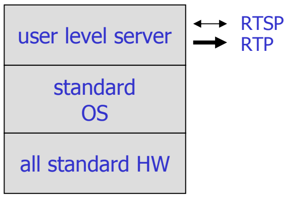
Overview

- Server examples
- Resources, real-time, "continuous" media streams, …
- (CPU) Scheduling
- Next time, memory and storage

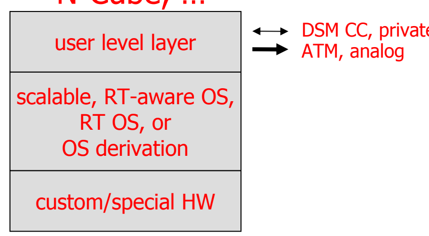
Server Examples

(Video) Server Product Examples

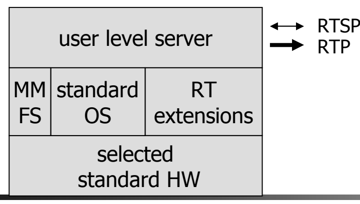
1) Real server, VXtreme, Starlight, VDO, Netscape Media Server, MS Media Server, Apple Darwin, …



2) IBM Mediastreamer, Oracle Video Cartridge, N-Cube, …

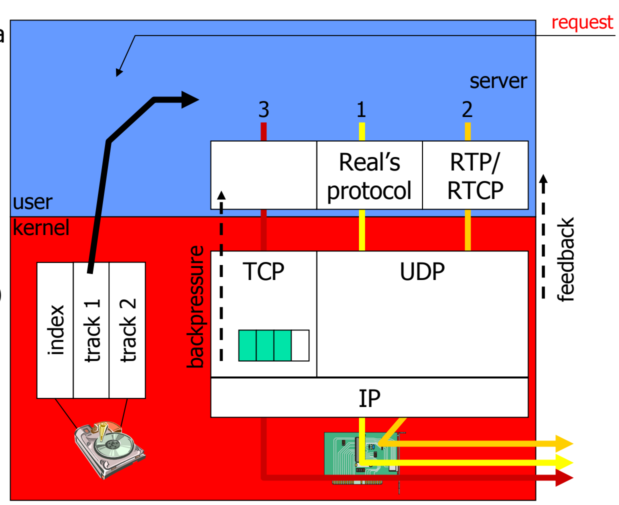


3) SGI/Kassena Media Base, SUN Media Center, IBM Video Charger, …



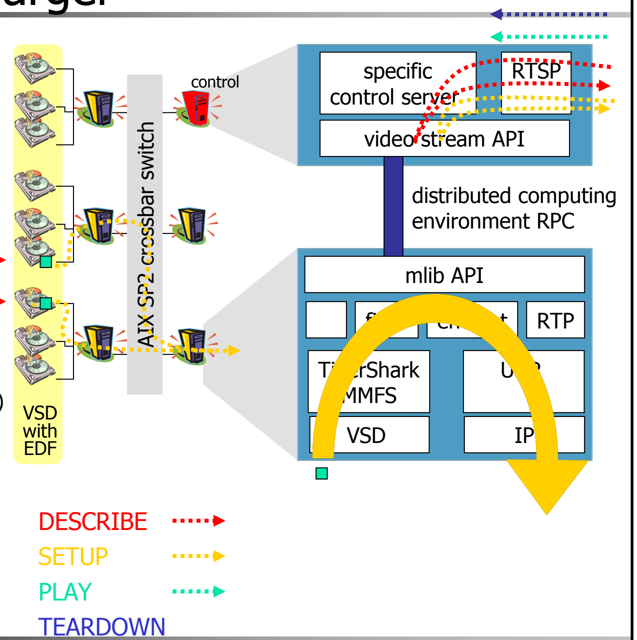
Real Server

- User space implementation
  - one control server
  - several protocols
  - several versions of data in same file
  - adapts to resources
- Several formats, e.g.,
  - Real's own
  - MPEG-2 version with "stream thinning" (dropped with REAL ➀)
- Does not support
  - Quality-of-Service
  - load leveling



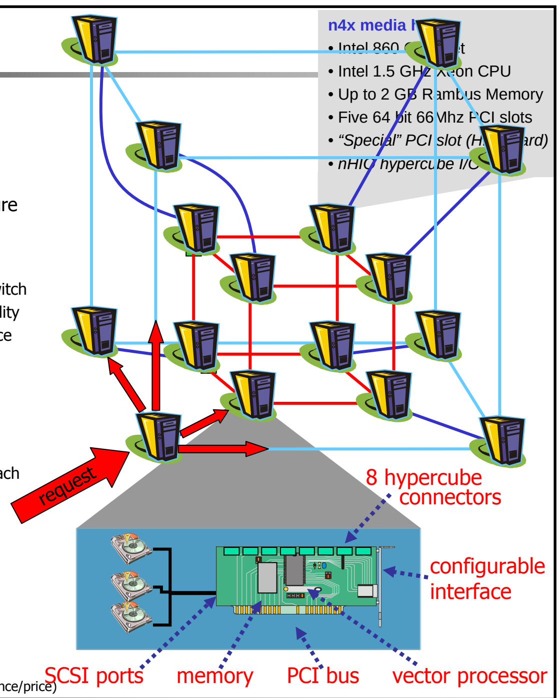
IBM Video Charger

- May consist of one machine only, or …
- … several IBM's Advanced Interactive eXecutive (AIX) machines
- Servers
  - control
  - data
- Lightly modified existing components
  - OS AIX4/5L
  - virtual shared disks (VSD) (guaranteed disk I/Os)
- Special components
  - TigerShark MMFS (buffers, data rate, prefetching, codec, ...)
  - stream filters, control server, APIs, ...



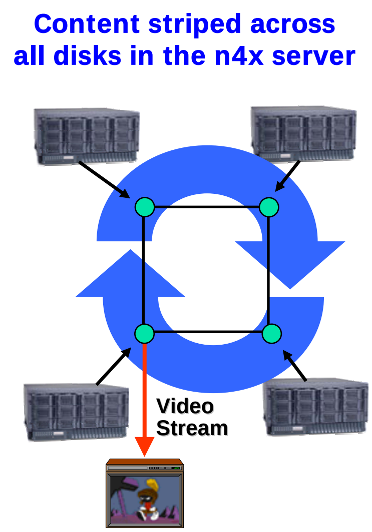
nCUBE

- Original research from Cal Tech/Intel ('83)
- Bought by C-COR in Jan. 05 (~90M$)
- One server scales from 1 to 256 machines, 2^n, n ∈ [0, 8], using a *hypercube* architecture
- Why a hypercube?
  - video streaming is a switching problem
  - hypercube is a high performance scalable switch
  - no content replication and true linear scalability
  - integrated adaptive routing provides resilience
- Highlights
  - one copy of a data element
  - scales from 5,000 to 500,000 clients
  - exceeds 60,000 simultaneous streams
  - 6,600 simultaneous streams at 2 - 4 Mbps each (26 streams per machine if n = 8)
- Special components
  - boards with integrated components
  - TRANSIT operating system
  - n4 HAVOC (1999)
    - Hypercube And Vector Operations Controller
    - ASIC-based hypercube technology
  - n4x nHIO (2002)
    - nCUBE Hypercube I/O controller (8X performance/price)



nCUBE: Naturally load-balanced

- Disks connected to All MediaHubs
  - Each title striped across all MediaHUBs
  - Streaming Hub reads content from all disks in the video server
- Automatic load balancing
  - Immune to content usage pattern
  - Same load if same or different title
  - Each stream's load spread over all nodes
- RAID Sets distributed across MediaHubs
  - Immune to a MediaHUB failure
  - Increasing reliability
- Only 1 copy of each title ever needed
  - Lots of room for expanded content, network-based PVR, or HDTV content



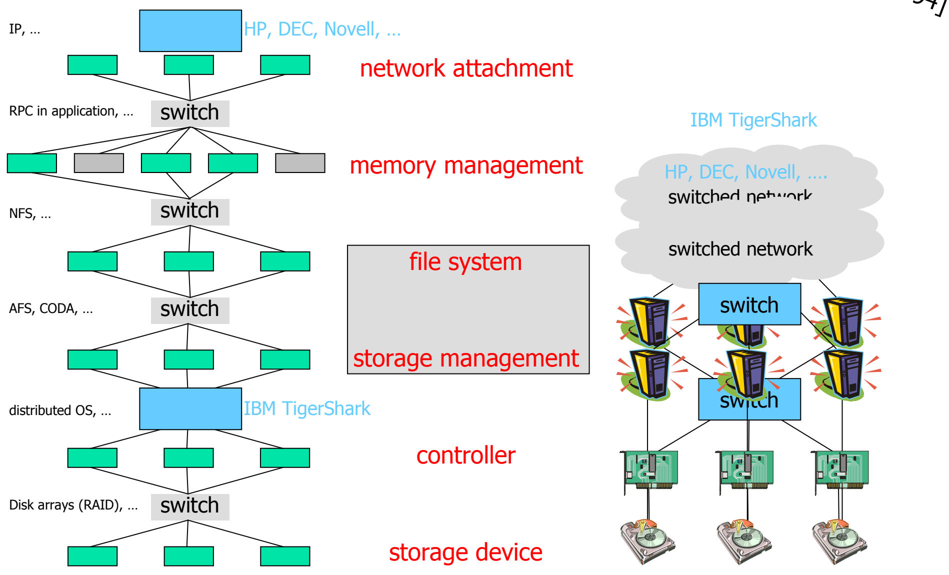
Small Comparison

Real	Video Charger	nCUBE
standard HW	selected HW	special HW
each machine its own storage, or NFS	shared disk access, no replication (except for load leveling and fault tolerance)	shared disk access, no replication
single OS image	cluster machines using switch	cluster machines using wired cube
user space server	user space server and loadable kernel modules	server in both kernel and user space

(Video) Server Structures

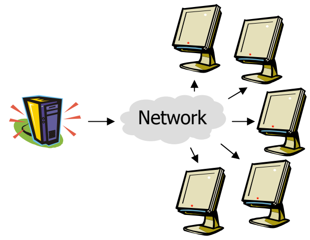
Server Components & Switches

[Tetzlaff & Flynn 94]

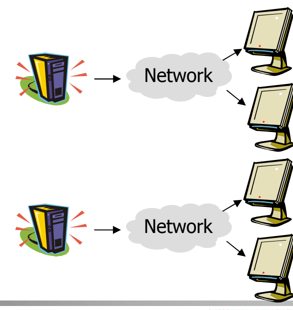


Server Topology – I

- Single server
  - easy to implement
  - scales poorly

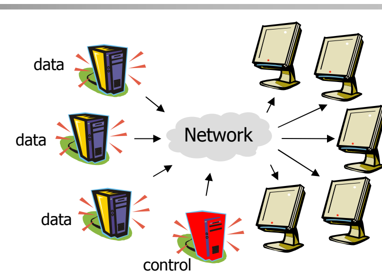


- Partitioned server
  - users divided into groups
  - *content* : assumes equal groups
  - *location* : store all data on all servers
  - load imbalance

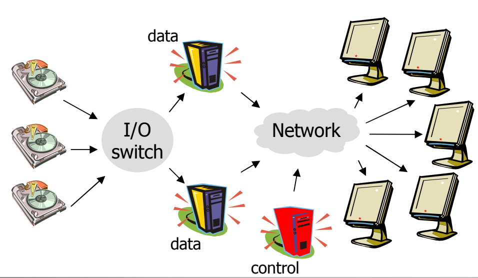


Server Topology – II

- Externally switched servers
  - use network to make server pool
  - manages load imbalance (control server directs requests)
  - still data replication problems
  - (control server doesn't need to be a physical box - distributed process)

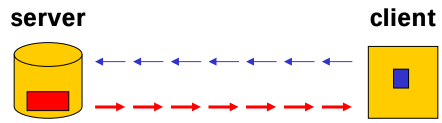


- Fully switched server
  - server pool
  - storage device pool
  - additional hardware costs
  - e.g., Oracle, Intel, IBM

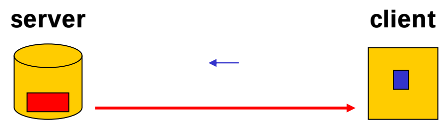


Data Retrieval

- Pull model:
  - client sends several requests
  - deliver only small part of data
  - fine-grained client control
  - favors high interactivity
  - suited for editing, searching, etc.



- Push model
  - client sends *one* request
  - streaming delivery
  - favors capacity planning
  - suited for retrieval, download, playback, etc.



Typical Trends In the Internet Today

- Push systems
  (pull in video editing/database systems)
- Traditional (specialized) file systems – not databases – for data storage
- No in-band control
  (control and data information in separate streams)
- External directory services for data location
  (control server + data pump)
- Request redirection for access control
- Single stand-alone servers → (fully) switched servers

Resources and Real-Time

Resources

- Resource:
  "A resource is a system entity required by a task for manipulating data"
  [Steimetz & Narhstedt 95]

- Characteristics:
  - active: provides a service, e.g., CPU, disk or network adapter
  - passive: system capabilities required by active resources, e.g., memory
  - exclusive: only one process at a time can use it, e.g., CPU
  - shared: can be used by several concurrent processed, e.g., memory

Deadlines and Real-Time

- Deadline:
  "A deadline represents the latest acceptable time for the presentation of the processing result"

- Hard deadlines:
  - must never be violated → system failure

- Soft deadlines:
  - in some cases, the deadline might be missed
    - not too frequently
    - not by much time
  - result still may have some (but decreasing) value

- Real-time process:
  "A process which delivers the results of the processing in a given time-span"

- Real-time system:
  "A system in which the correctness of a computation depends not only on obtaining the result, but also upon providing the result on time"

Admission and Reservation

- To prevent overload, admission may be performed:
  - schedulability test:
    - "are there enough resources available for a new stream?"
    - "can we find a schedule for the new task without disturbing the existing workload?"
    - a task is allowed if the utilization remains < 1
  - ⇨ yes – allow new task, allocate/reserve resources
  - ⇨ no – reject

- Resource reservation is analogous to booking (asking for resources)
  - pessimistic
    - avoid resource conflicts making worst-case reservations
    - potentially under-utilized resources
    - guaranteed QoS
  - optimistic
    - reserve according to average load
    - high utilization
    - overload may occur
  - "perfect"
    - must have detailed knowledge about resource requirements of all processes
    - too expensive to make/takes much time

Real-Time and Operating Systems

- The operating system manages **local** resources
  (CPU, memory, disk, network card, busses, ...)

- In a real-time scenario, support is needed for
  - real-time processing
  - high-rate, timely I/O

- This means support for proper …
  - *scheduling* –
    high priorities for time-restrictive tasks
  - *timer support* –
    clock with fine granularity and event scheduling with high accuracy
  - *kernel preemption* –
    avoid long periods where low priority processes cannot be interrupted
  - *efficient memory management* –
    prevent code for real-time programs from being paged out (replacement)
  - *fast switching* –
    both interrupts and context switching should be fast
  - ...

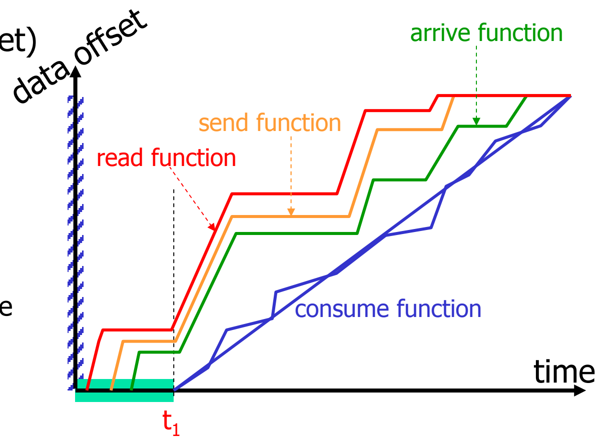
Timeliness

Timeliness

- Start presenting data (e.g., video playout) at t_1

- Consumed bytes (offset)
  - variable rate
  - constant rate

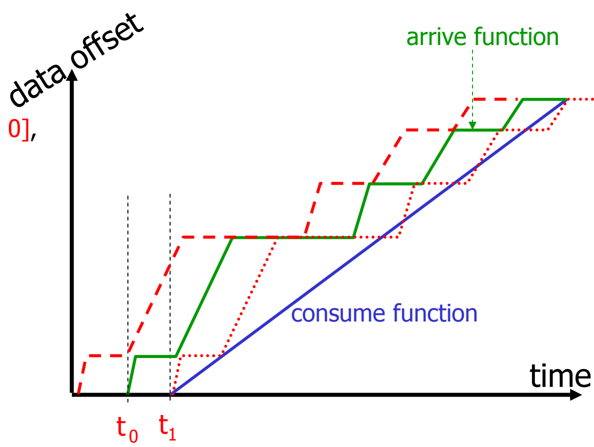
- Must start retrieving data earlier
  - Data must arrive before consumption time
  - Data must be sent before arrival time
  - Data must be read from disk before sending time



Timeliness

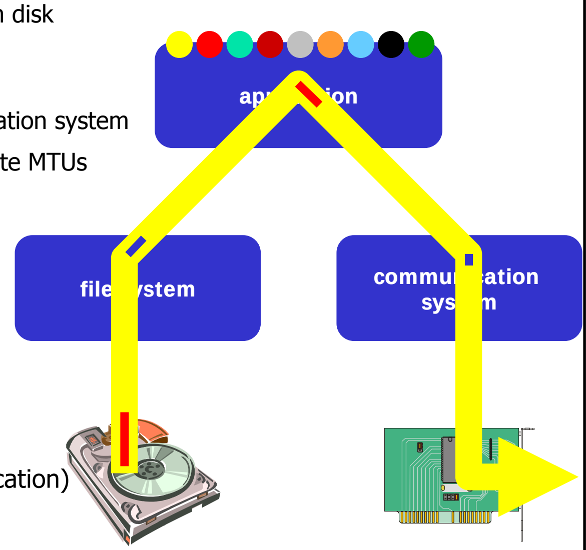
- Need buffers to hold data between the functions, e.g., client B(t) = A(t) – C(t), i.e., ∀ t : A(t) ≥ C(t)

- Latest start of data arrival is given by $\min[B(t,t_0,t_1)\ ;\ \forall\ t\ B(t,t_0,t_1) \geq 0]$, i.e., the buffer must at all times **t** have more data to consume



Timeliness: Streaming Data

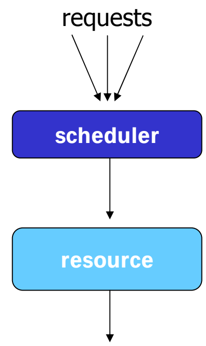
- "Continuous Media" and "continuous streams" are ILLUSIONS
  - retrieve data in blocks from disk
  - transfer blocks from file system to application
  - send packets to communication system
  - split packets into appropriate MTUs
  - ... (intermediate nodes)
  - ... (client)
  - ➔ different optimal sizes
  - pseudo-parallel processes (run in time slices)
  - ☞ need for scheduling (to have timing and appropriate resource allocation)



(CPU) Scheduling

Scheduling

- A task is a schedulable entity (a process/thread executing a job, e.g., a packet through the communication system or a disk request through the file system)
- In a multi-tasking system, several tasks may wish to use a resource simultaneously
- A scheduler decides which task that may use the resource, i.e., determines order by which requests are serviced, using a scheduling algorithm



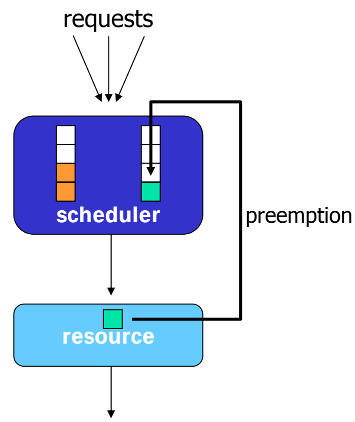
- Each active (CPU, disk, NIC) resources needs a scheduler (passive resources are also "scheduled", but in a slightly different way)

Scheduling

- Scheduling algorithm classification:
  - dynamic
    - make scheduling decisions at *run-time*
    - flexible to adapt
    - considers only actual task requests and execution time parameters
    - large run-time overhead finding a schedule
  - static
    - make scheduling decisions at *off-line* (also called pre-run-time)
    - generates a dispatching table for run-time dispatcher at compile time
    - needs complete knowledge of task before compiling
    - small run-time overhead
  - preemptive
    - currently executing tasks may be interrupted (preempted) by higher priority processes
    - the preempted process continues later at the same state
    - potential frequent contexts switching
    - (almost!?) useless for disk and network cards
  - non-preemptive
    - running tasks will be allowed to finish its time-slot (higher priority processes must wait)
    - reasonable for short tasks like sending a packet (used by disk and network cards)
    - less frequent switches

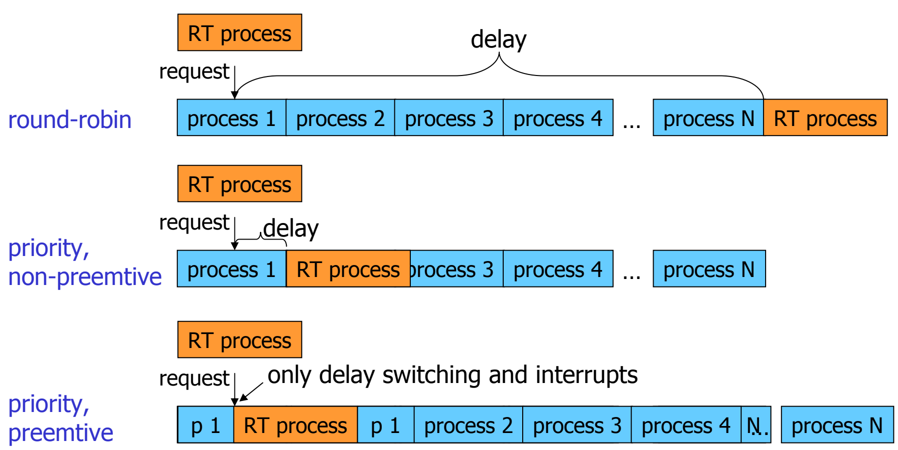
Scheduling

- Preemption:
  - tasks waits for processing
  - scheduler assigns priorities
  - task with highest priority will be scheduled first
  - preempt current execution if a higher priority (more urgent) task arrives
  - real-time and best effort priorities (real-time processes have higher priority - if exists, they will run)
  - to kinds of preemption:
    - preemption points
      - predictable overhead
      - simplified scheduler accounting
    - immediate preemption
      - needed for hard real-time systems
      - needs special timers and fast interrupt and context switch handling



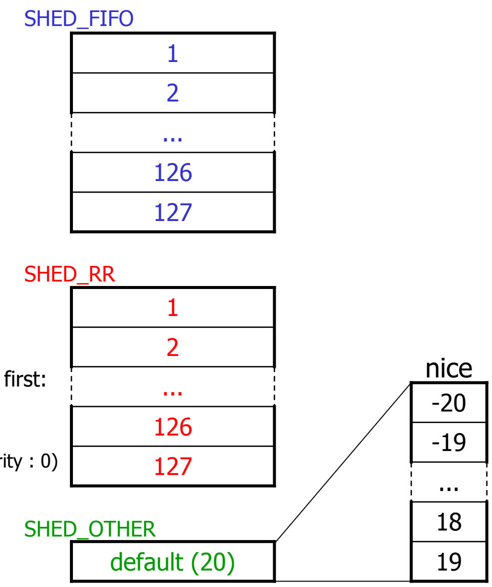
Scheduling

- Scheduling is difficult and takes time – RT vs NRT example:



Scheduling in Linux

- Preemptive kernel
- Threads and processes used to be equal, but Linux uses (in 2.6) thread scheduling
- SHED_FIFO
  - may run forever, no timeslices
  - may use it's own scheduling algorithm
- SHED_RR
  - each priority in RR
  - timeslices of 10 ms (quantums)
- SHED_OTHER
  - ordinary user processes
  - uses "nice"-values: 1≤ priority≤40
  - timeslices of 10 ms (quantums)
- Threads with highest *goodness* are selected first:
  - realtime (FIFO and RR): goodness = 1000 + priority
  - timesharing (OTHER): goodness = (quantum > 0 ? quantum + priority : 0)
- *Quantums* are reset when no ready process has quantums left (end of *epoch*): quantum = (quantum/2) + priority



Scheduling in Linux

http://kerneltrap.org/node/8059

- The 2.6.23 kernel used the new *Completely Fair Scheduler (CFS)*
  - address unfairness in desktop and server workloads
  - uses ns granularity, does not rely on jiffies or HZ details
  - uses an extensible hierarchical scheduling classes
    - SCHED_FAIR / SCHED_NORMAL – the CFS desktop scheduler – replace SCHED_OTHER
      - no run-queues, a tree-based timeline of future tasks
    - sched_rt replace SCHED_RT and SCHED_FIFO
      - uses 100 run-queues

Real-Time Scheduling

- Resource reservation
  - QoS can be guaranteed
  - relies on knowledge of tasks
  - no fairness
  - origin: time sharing operating systems
  - e.g., earliest deadline first (EDF) and rate monotonic (RM)
    (AQUA, HeiTS, RT Upcalls, ...)

- Proportional share resource allocation
  - no guarantees
  - requirements are specified by a relative share
  - allocation in proportion to competing shares
  - size of a share depends on system state and time
  - origin: packet switched networks
  - e.g., Scheduler for Multimedia And Real-Time (SMART)
    (Lottery, Stride, Move-to-Rear List, ...)

Earliest Deadline First (EDF)

- Preemptive scheduling based on dynamic task priorities

- Task with *closest deadline has highest priority (dynamic)*
  → stream priorities vary with time

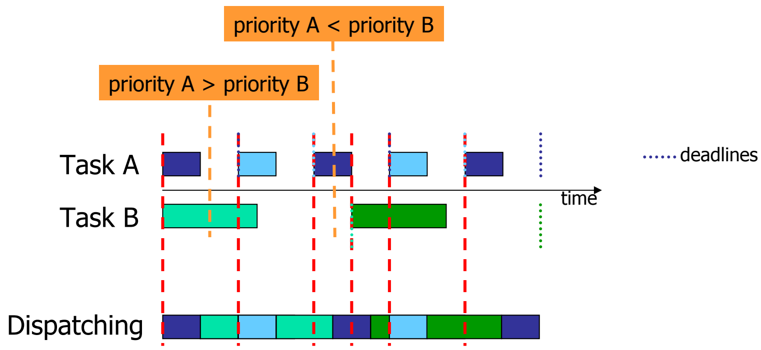
- Dispatcher selects the highest priority task

- Assumptions:
  - requests for all tasks with deadlines are periodic
  - the deadline of a task is equal to the end on its period (starting of next)
  - independent tasks (no precedence)
  - run-time for each task is known and constant
  - context switches can be ignored

University of Oslo INF5071, Autumn 2007, Carsten Griwodz & Pål Halvorsen [simula . research laboratory]

Earliest Deadline First (EDF)

- Example:

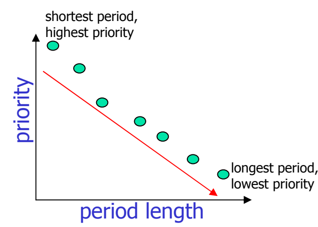


Rate Monotonic (RM) Scheduling

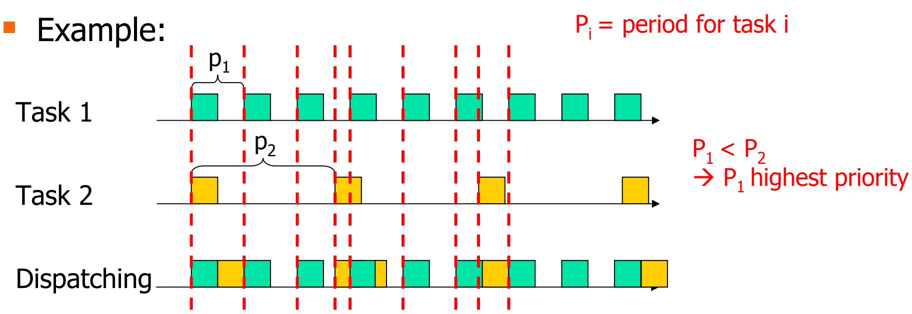
- Classic algorithm for hard real-time systems with one CPU
  [Liu & Layland '73]
- Pre-emptive scheduling based on *static task priorities*
- Optimal: no other algorithms with static task priorities can schedule tasks that cannot be scheduled by RM
- Assumptions:
  - requests for all tasks with deadlines are periodic
  - the deadline of a task is equal to the end on its period (starting of next)
  - independent tasks (no precedence)
  - run-time for each task is known and constant
  - context switches can be ignored
  - *any non-periodic task has no deadline*

Rate Monotonic (RM) Scheduling

- Process priority based on task periods
  - task with shortest period gets highest *static* priority
  - task with longest period gets lowest *static* priority
  - dispatcher always selects task requests with highest priority

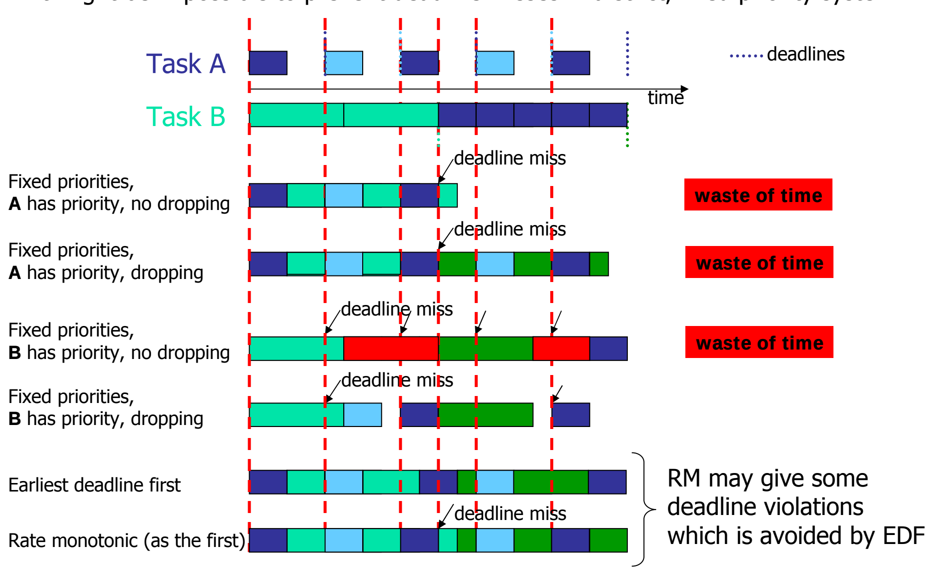


- Example:



EDF Versus RM

- It might be impossible to prevent deadline misses in a strict, fixed priority system:



SMART (**S**cheduler for **M**ultimedia **A**nd **R**eal–**T**ime applications)

- Designed for multimedia and real-time applications

- Principles
  - priority – high priority tasks should not suffer degradation due to presence of low priority tasks
  - proportional sharing – allocate resources proportionally and distribute unused resources (work conserving)
  - tradeoff immediate fairness – real-time and less competitive processes (short-lived, interactive, I/O-bound, ...) get instantaneous higher shares
  - graceful transitions – adapt smoothly to resource demand changes
  - notification – notify applications of resource changes

SMART (**S**cheduler for **M**ultimedia **A**nd **R**eal–**T**ime applications)

- Tasks have…
  - urgency – an immediate real-time constraint, short deadline (determine when a task will get resources)
  - importance – a priority measure
    - expressed by a tuple:
      `[ priority p , biased virtual finishing time bvft ]`
    - **p** is static: supplied by user or assigned a default value
    - **bvft** is dynamic:
      - virtual finishing time: virtual application time for finishing if given the requested resources
      - bias: bonus for interactive tasks

- *Best effort schedule* based on urgency and importance
  1. find most important tasks – compare tuple:
     $T_1 > T_2 \Leftrightarrow (p_1 > p_2) \vee (p_1 = p_2 \wedge bvft_1 > bvft_2)$
  2. sort each group after urgency (EDF based sorting)
  3. iteratively select task from candidate set as long as schedule is feasible

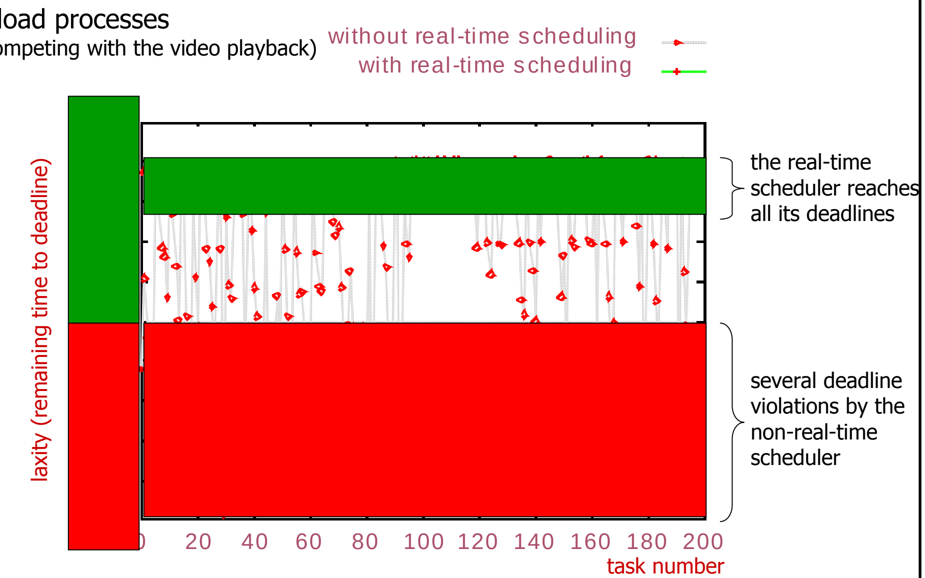
Evaluation of a Real-Time Scheduling

- Tests performed
  - by IBM (1993)
  - executing tasks with and without EDF
  - on an 57 MHz, 32 MB RAM, AIX Power 1

- Video playback program:
  - one real-time process
    - read compressed data
    - decompress data
    - present video frames via X server to user
  - process requires 15 timeslots of 28 ms each per second
    → 42 % of the CPU time

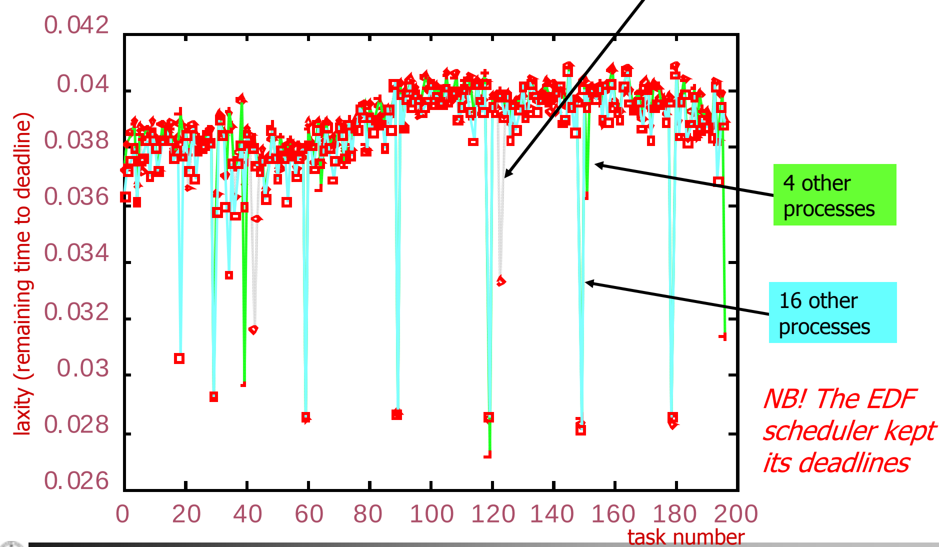
Evaluation of a Real-Time Scheduling

3 load processes
(competing with the video playback)



Evaluation of a Real-Time Scheduling

Varied the number of load processes
(competing with the video playback)

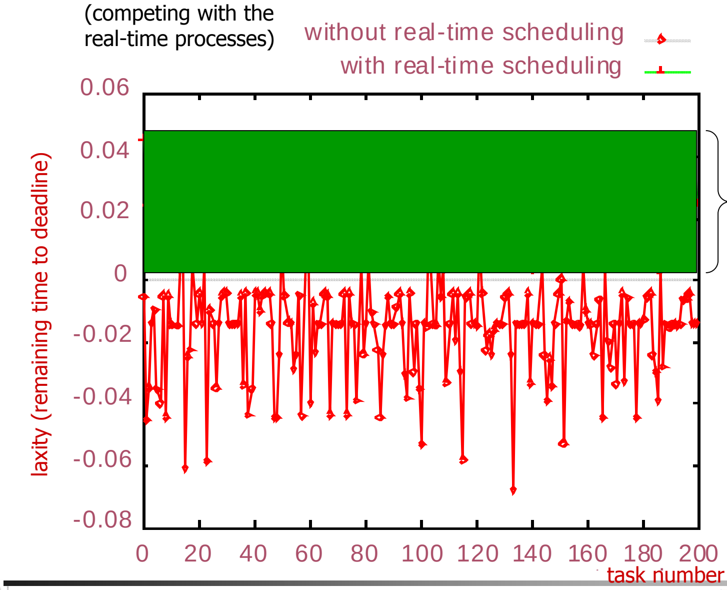


Evaluation of a Real-Time Scheduling

- Tests again performed
  - by IBM (1993)
  - on an 57 MHz, 32 MB RAM, AIX Power 1
- "Stupid" end system program:
  - 3 real-time processes only requesting CPU cycles
  - each process requires 15 timeslots of 21 ms each per second
    → 31.5 % of the CPU time each
    → 94.5 % of the CPU time required for real-time tasks

Evaluation of a Real-Time Scheduling

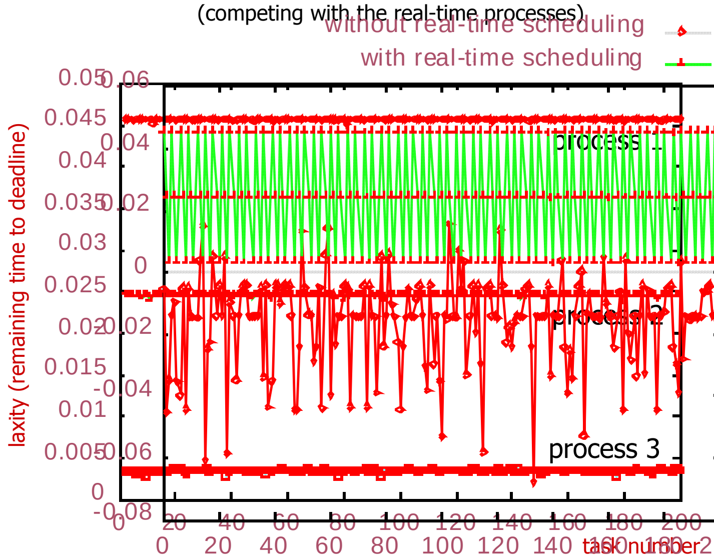
1 load process
(competing with the real-time processes)



the real-time scheduler reaches all its deadlines

Evaluation of a Real-Time Scheduling

16 load process
(competing with the real-time processes)



Regardless of other load, the EDF-scheduler reach its deadlines (laxity almost equal as in 1 load process scenario)

NOTE: Processes are scheduled in same order

Summary

- Resources need to be properly scheduled
- CPU is an important resource
- Many ways to schedule depending on workload
- Hierarchical, multi-queue priority schedulers have existed a long time already, and newer ones usually try to improvement upon of this idea
- Next week, memory and persistent storage

Some References

1. AMD, http://multicore.amd.com/en/Products
2. C-COR, http://www.c-cor.com
3. Haskin, R.L: "Tiger Shark--A scalable file system for multimedia", IBM Journal of Research and Development, Vol. 42, No. 2, 1997, p. 185
4. IBM: http://www-306.ibm.com/software/data/videocharger/
5. Intel, http://www.intel.com
6. MPEG.org, http://www.mpeg.org/MPEG/DVD
7. nCUBE, http://ncube.com (not available after Jan. 2005)
8. Sitaram, D., Dan, A.: "Multimedia Servers – Applications, Environments, and Design", Morgan Kaufmann Publishers, 2000
9. Tendler, J.M., Dodson, S., Fields, S.: "IBM e-server: POWER 4 System Microarchitecture", Technical white paper, 2001
10. Tetzlaff, W., Flynn, R.: "Elements of Scalable Video Servers", IBM Research Report 19871 (87884), 1994