Embedded Optimization

(1) Fast Booting

(2) Starvation-Free Realtime Scheduler

Masahiko Takahashi

Akira Tsukamoto

System Platforms Research Laboratories,

NEC Corporation

2006/01/20

Embedded Optimization

(1) Fast Booting

--- An approach from the experience of implementing fast network boot to embedded systems ---

Masahiko Takahashi

System Platform Research Laboratories,
NEC Corporation

2006/01/20

Outline

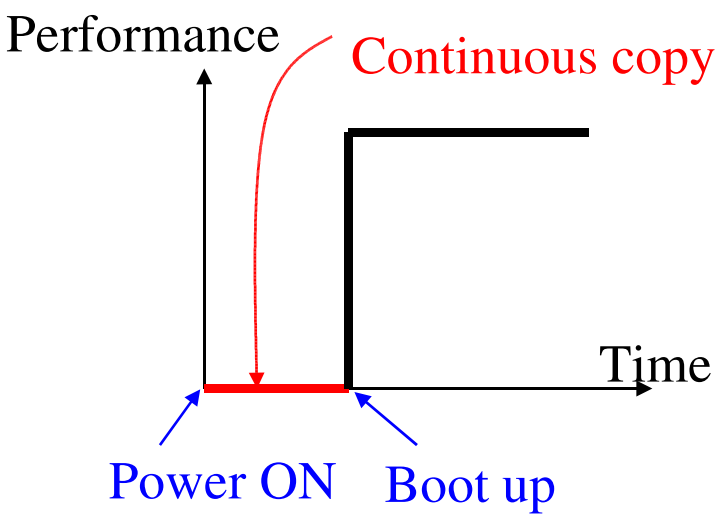
- BACKGROUND: It is hard to update all PCs' OS or applications when they should be updated due to security update, or bug fix
  - Central management of OS and application programs for all PCs with network boot technique is useful
- Normal network boot is too slow ! Therefore, we started to research “fast booting technique”
  - Using memory-image which is a image of applications and daemons after they have started
  - On booting, minimum portion of the memory-image for initial booting is retrieved
- We have implemented our technique for x86 (fall, 2004)
- This technique is fundamentally adaptable for embedded system ?
  - Just start porting to an embedded system
- Through the implementation of fast network booting technique, we have found issues of memory-image usage

What is fast network booting?

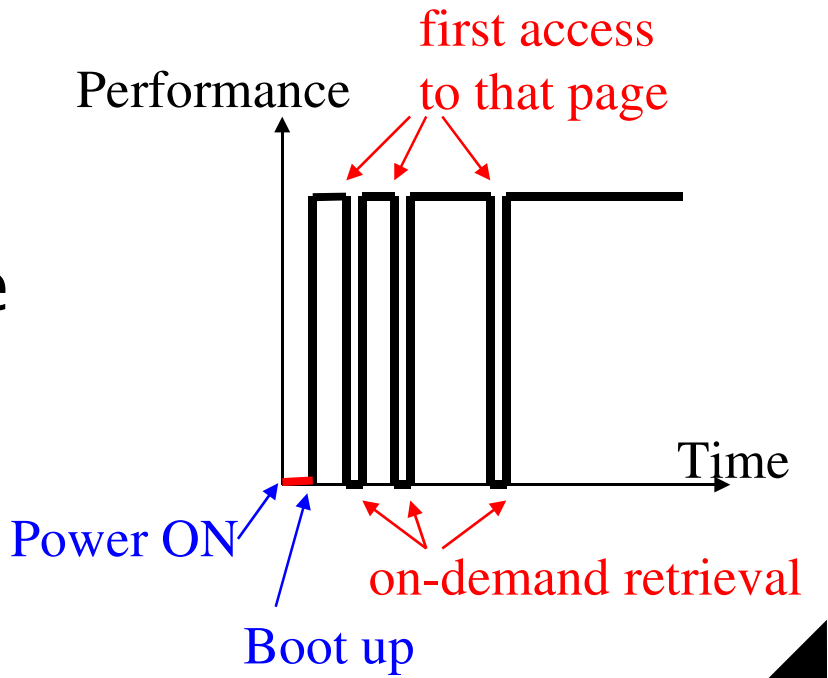
- A client system boots up by retrieving the memory-image through the network
  - Suitable for a fixed system, such as a server or embedded system
- To achieve faster boot up time, it first retrieves minimum portion of the memory-image, which is required for initial booting
- The rest of the memory-image is loaded by on-demand fashion; it is retrieved when the first access occurred (page aligned)
  - To incur a page fault to get that page, we change “present bit” of its page table entry to "0"

Comparison with Hibernation

- Hibernation
  - On booting, it incurs continuous copy from ROM/HDD to RAM
  - Boot up time depends on the RAM size (i.e., memory-image size)



- Proposed Method
  - Minimum retrieval of the memory-image brings minimum boot up time
  - Rest of the memory-image is on-demand retrieval
    - The performance degradation incurred is only in *short period*



Assumed Environment

Server

- TFTP server
- Memory-image

(initial retrieval portion)

(on-demand retrieval)

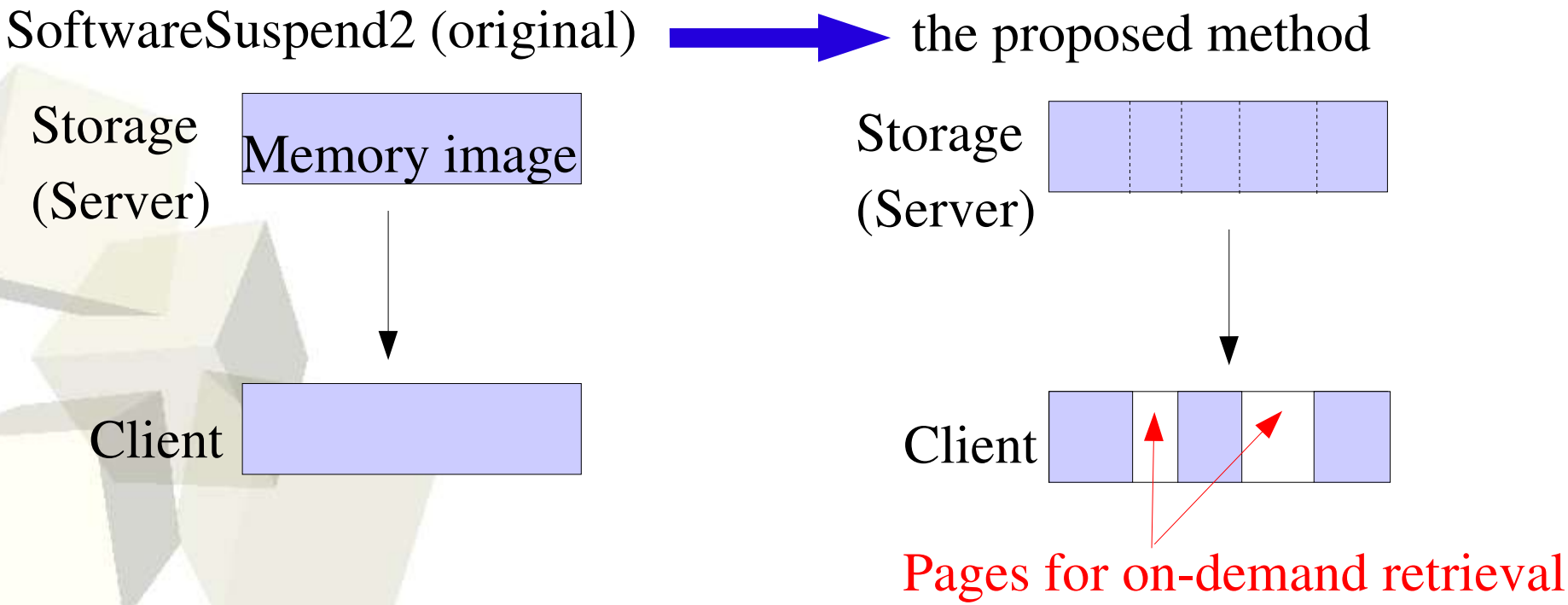
Diskless client

- Linux (2.6.6)
  TFTP client
- SoftwareSuspend2
  (2.0.0.72)

Note: To achieve higher network throughput, block size of TFTP should be 65464 bytes (RFC 2348).

Modifying SoftwareSuspend2

- The memory-image is logically divided into two part: initial and on-demand retrieval portion
  - Current implementation:
    - pageset1 is initial portion
    - pageset2 is on-demand portion



Modifying Linux

- Overwrite Page table entry
  - Present bits in page table entries for the pages belonging to pageset2 should be "0" to incur page fault
- Modification of page fault handler
  - Page fault hander must retrieve a page of rest of the memory-image at the first time it is accessed

  Note: it must distinguish on-demand retrieval and normal page fault
- Adding TFTP client functionality
  - To retrieve pages (both initial and on-demand portion) from TFTP server

Performance Evaluation

- Environment
  - Server: Pentium4 2GHz, 1024MB(PC133), eepro100
  - Client: Pentium4 2GHz, 512MB(PC133), eepro100
  - PXE/DHCP/TFTP/NFS root filesystem
- Memory-image: “X Window + xterm + window manager"
- Performance Result
  - the time from Power ON to enable the keyboard input
    - Proposed method (fast network booting): 25 sec.
    - Normal network booting: 60 sec.

Try to apply to the embedded systems

- Fast booting by using memory-image has already proposed in the past CELF Jamboree
- Hibernation type has a issue of “longer copy latency” as described

→ So, our technique that retrieves initial portion of the memory-image so as to achieve faster boot up time may be adaptable for embedded systems ?
  - Just start to implement

Demerit of fast boot with memory-image

- Known Issues (example)
  - Inconsistency of filesystem may occur if the system has writable partition after reboot, because the memory-image assumes old state of the filesystem
  - Hard to support individual customization
  - Maybe fail to initialize a device which are initialized by application program
  - The memory-image strongly depends on the hardware environment

→ It is required to make a guideline to build a system including application programming