User Level Techniques for Improvement of Disk I/O in WWW Caching

Ken-ichi Chinen

k-chinen@is.aist-nara.ac.jp

Graduate School of Information Science, Nara Institute of Science and Technology

Overview

- Goal, target and challenge
- Bottle necks
- Our techniques at user land
- Evaluation

Background

Current trends of WWW Cache

- Network appliance
- Kernel side techniques

However, these means...

- No or poor portability
- Vendor depend
- Expensive

Goal

- Fast WWW caching system upon **GENERIC** system
- For work-group and/or department cache service
  - ◇ up to 100Mbps

We want
More portable
Vendor independent
and cheap...

What is Generic System ?

Don’t use special something

- super computer (e.g, GS-xxx, Origin-xxxx)
- physical devices (e.g., silicon disk)
- logical devices (e.g., /dev/poll)
- filesystem architectures (e.g., raw I/O)
- system-calls (e.g., sendfile())

Targets & Challenges

Targets

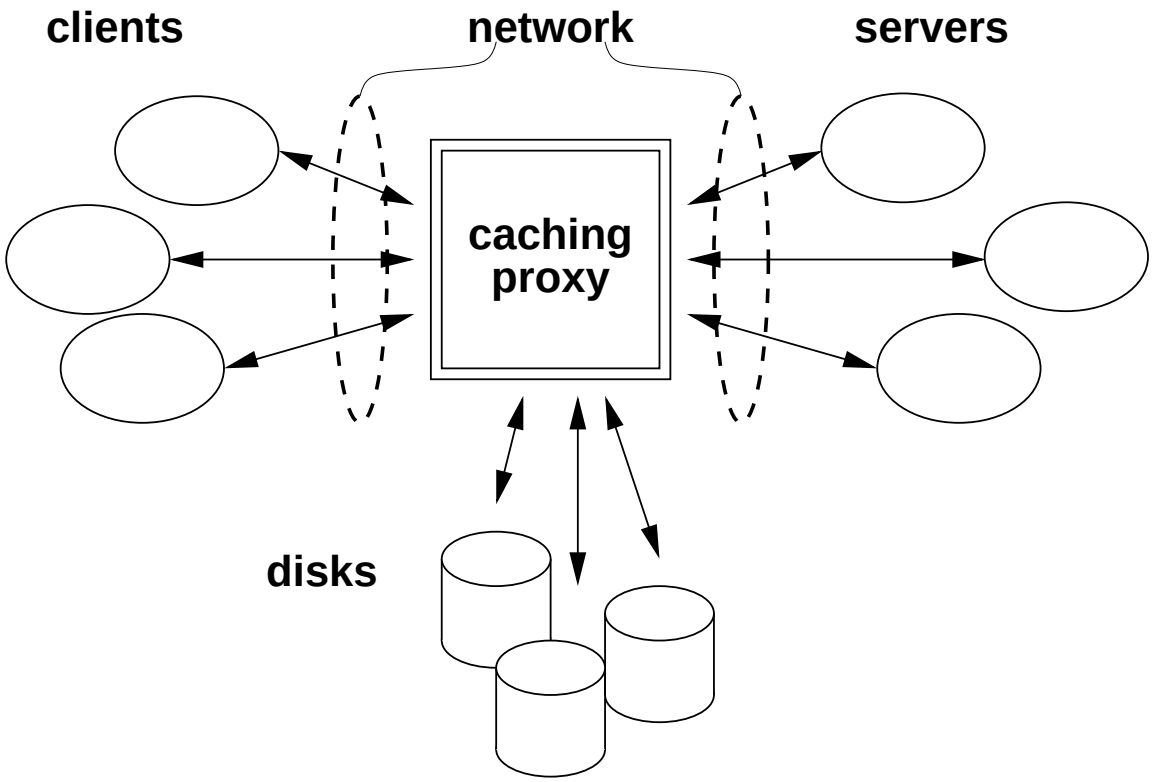
- UNIX
- BSD sockets and related
- UFS interfaces
- POSIX threads

Challenges

- Fast disk I/O w/o kernel change
- User level storage mechanism like filesystem

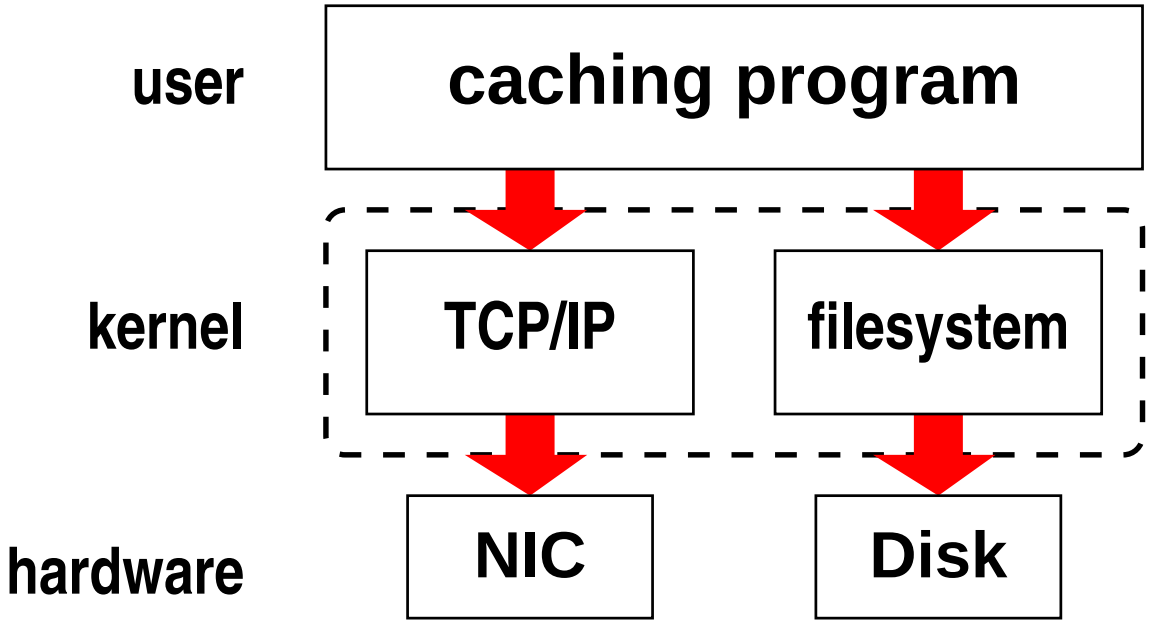
Where is Bottle-Neck(s) ?

- Caching Proxy is I/O multiplexing system
- Most of time is I/O waiting



Network v.s. Disk

- Network is controlled via TCP/IP stack.
- Disk is controlled via filesystem.



Network v.s. Disk (*cont.*)

Network

Typical: FastEthernet, Gigabit Ethernet

Max: Gigabit Ethernet

Boundary: PCI bus; thru-put is cutted under few 100Mbps

Disk

Typical: Ultra 2 Wide SCSI, SCSI 160m, ATA 100

Max: SCSI 160m, FC 2Gbps

Boundary: filesystem; few MB/s (several 10Mbps)

Filesystem is bottle-neck

- Network is enough to satisfy our requirement (under 100Mbps)
- Filesystem is not satisfy our requirement

Remarks:

- RAID is not perfect solution
  - ◇ RAID improves raw disk performance
  - ◇ But bottle-neck is file system...

Why filesystem takes long time ?

- Designed for generic purpose
- Resource management
  - ◇ i-node
  - ◇ directories
  - ◇ free block bitmaps
- System calls
  - ◇ open/creat
  - ◇ read/write

Characteristics of WWW Caching

- Small size
  - ◇ most several KBs, max few MBs
- Poor locality
  - ◇ typical hit ratio is 20% - 40%
  - ◇ smaller than memory or disk caching
- Long term stress
  - ◇ writing/reading several hours

Typical Approach

Many vendor/organization employ **special solution**

- Special filesystems
- Special scheduling for I/O
- Special systemcall for I/O
- and others

These approachs depend paticular OSes and/or vendors

Our Approach — KOTETU

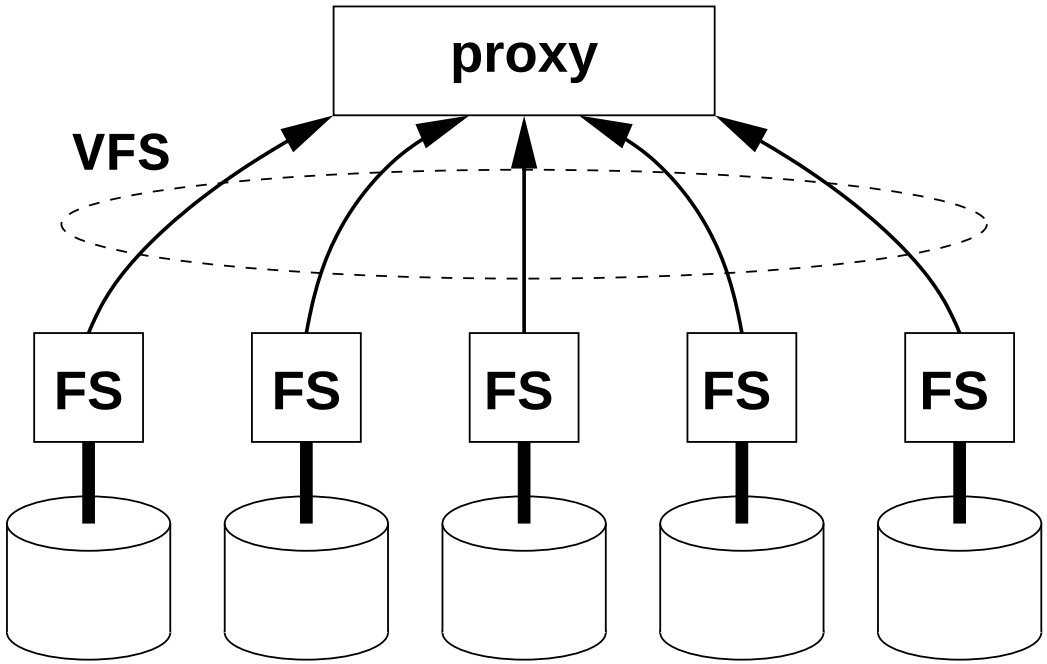
Reduce filesystem overheads w/ user level techniques

- Disk Splicing
- Object Packing
- Signature/Bitmaps

...Following pages show you these techniques

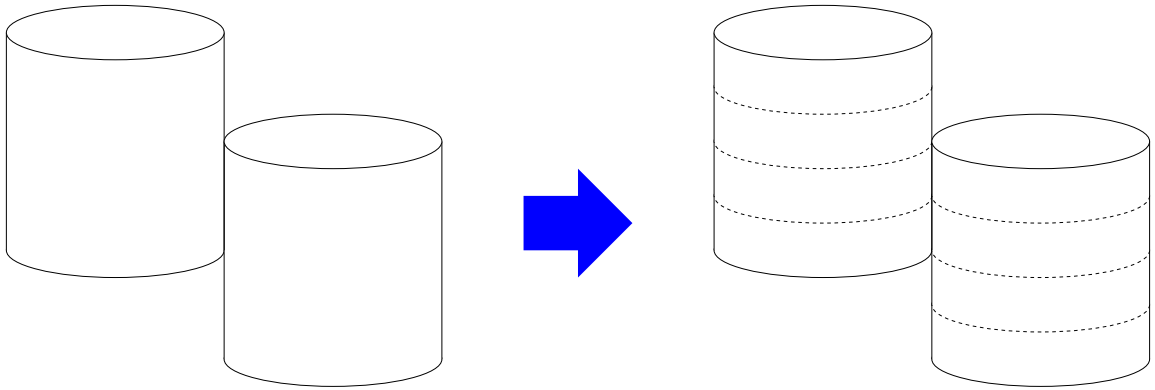
Disk Splicing

- Gathering multi disks (filesystem)
- Transparency access w/o symlink or other tricks



Disk Splicing — Splitting

- The performance of filesystem is depend on its size
- Small filesystem is faster than large one
- Split disk(s) into a set of small filesystems



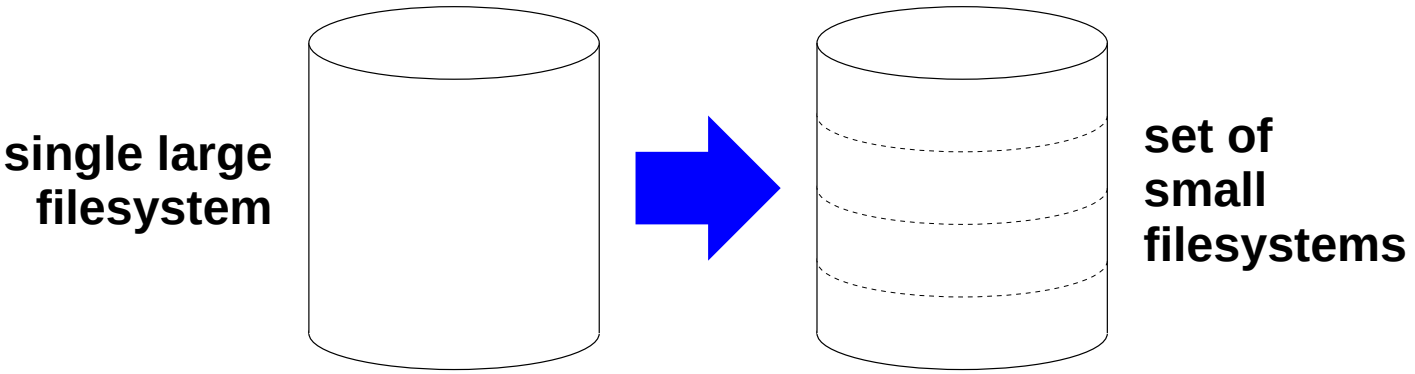
2 filesystems on 2 disks

8 filesystems on 2 disks

Disk Splicing — Splitting (*cont.*)

Striped disk array (RAID0) has same characteristic, also
Disk splitting brings more better performance
You can use following ways for splitting

- Partitions
- Logical drives, if possible



Object Packing

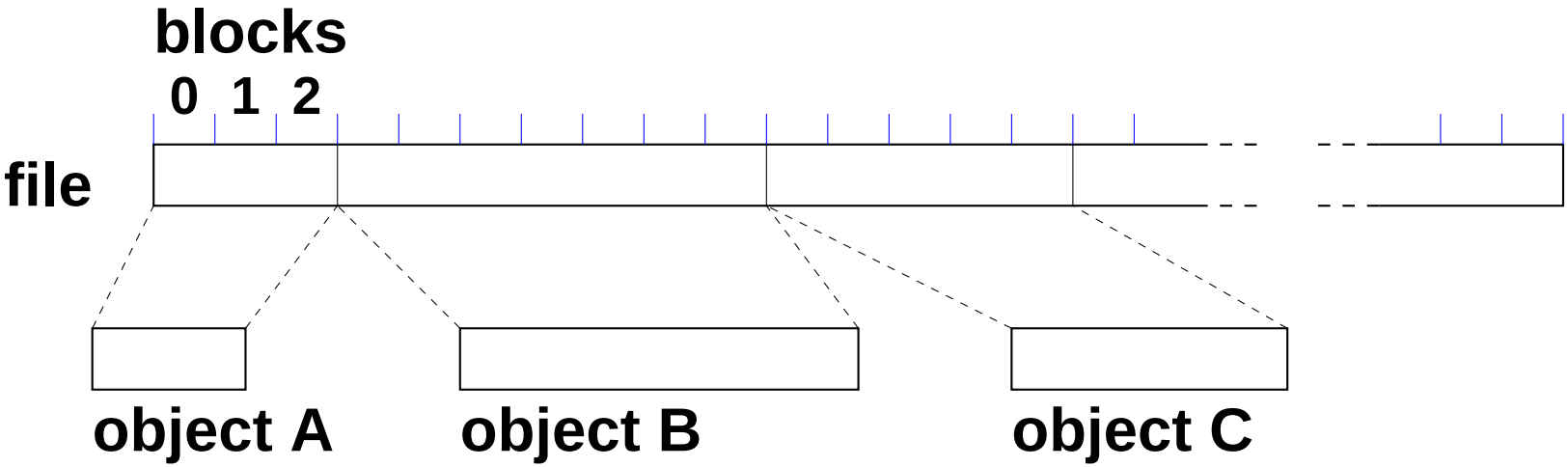
- Store many objects in single file
- Reduce the number of files
- Reduce open/close overhead (included directory traces in kernel side)

Remarks:

- Traditional systems store an object in single file
  - ◇ CERN httpd, Squid
- Don’t complain, it is historical reason

Object Packing (*cont.*)

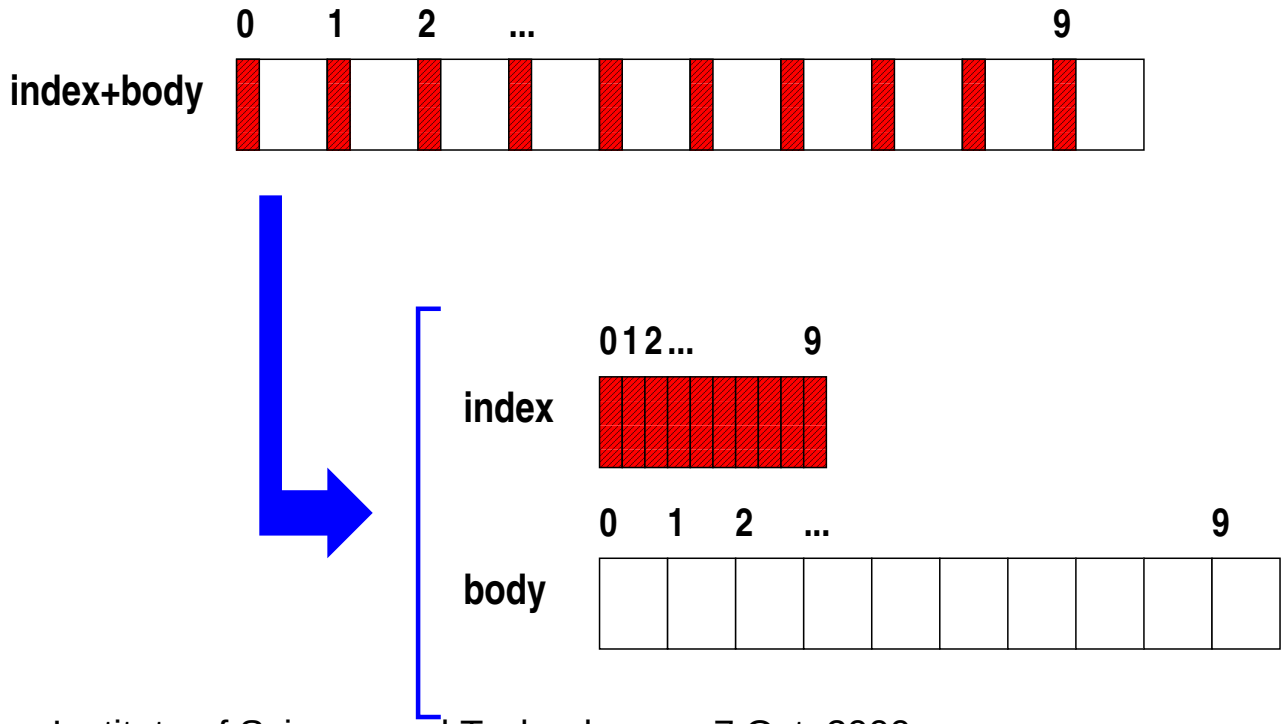
- File consists 1000-1500 blocks
  - ◇ Block size is 8-20Kbytes
- Objects are stored with several blocks



Split file into index and body

index – control block (attributes) // body – response

- Decrease distance between control blocks
- Improve seeking time



Object Packing - Example

- Cache capacity = 30GB
- Average object size = 10KB
- Block size = 8K
- The number of blocks in file = 1500

case 1: Store to single file $\Rightarrow$ 3M files

case 2: Store to packed file $\Rightarrow$ 2621 files

file size = 8K x 1500 = 12.3M

Packing reduce the number of files and overheads

Signature/Bitmaps

Hints for decision w/o disk access

Signature: Is the object stored ?

- Reduces seeking for checking object existence
- Makes fast swapin

Bitmap: where is empty block ?

- Reduces seeking to resolve object position
- Makes fast swapout

Signature/Bitmaps – Example

How much size is spend by such maps ?

1block = 8KBytes

map type	ratio	1MB map covers
bit (bitmap)	1:8192*8	64GB
integer (4byte)	4:8192	2GB

Swap

Swapping memory to disk is not eliminate

- Because main memory is very smaller than traffic
- Size of main memory is up to few GB
- WWW traffic is several tens GB/day

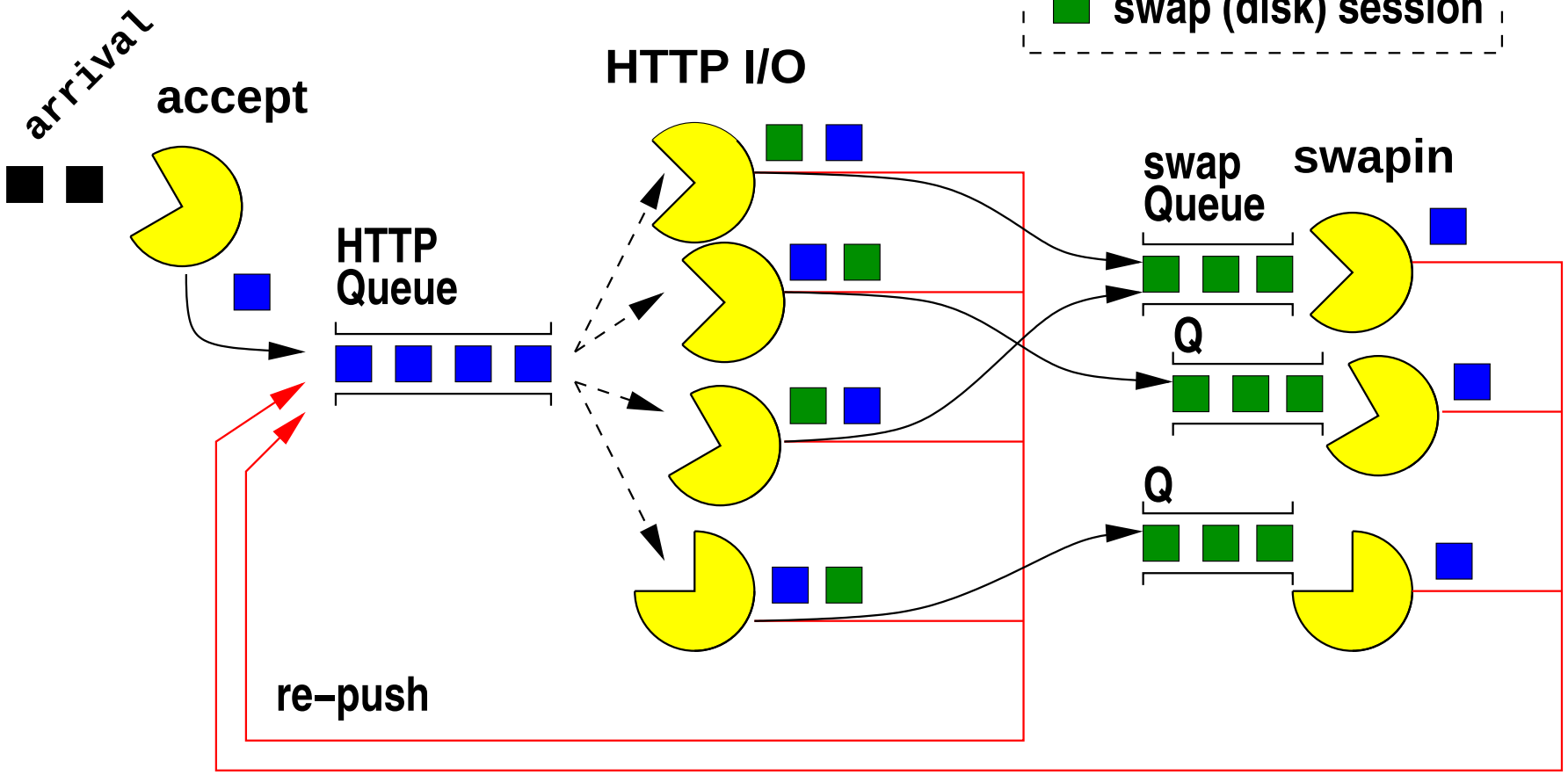
Example: fill 1GB memory with 100Mbps (12.8MB/s)

- All miss $\rightarrow$ 80 seconds
- 50% hit $\rightarrow$ upto 160 seconds

Swap (*cont.*)

- Swapin is depend on client requests
  - ◇ Swapin is proccessed in foreground
- Swapout is independent from client requests
  - ◇ Swapout is processed in background

Thread Formation



Evaluation – Facts

- Hardware

  CPU: dual Pentium 3 866MHz

  disk: six 10krpm SCSI disk w/ hardware RAID

  memory: four 256MB SDRAM w/ registered ECC

- Software

  OS: Linux 2.2.17

  FS: ReiserFS

  Proxy: KOTETU 1.5b

Using 35 logical drives on single disk array

Evaluation

- Our system achives 300 request/sec
- It means about 25Mbps
- Enough for workgroup service
- Workload // polygraph with Polymix-3
  - ◇ Cache Size = 31.3GB
  - ◇ Peak Rate = 300rps
  - ◇ Fill ratio = 75% (default)

Evaluation – Workload & Result

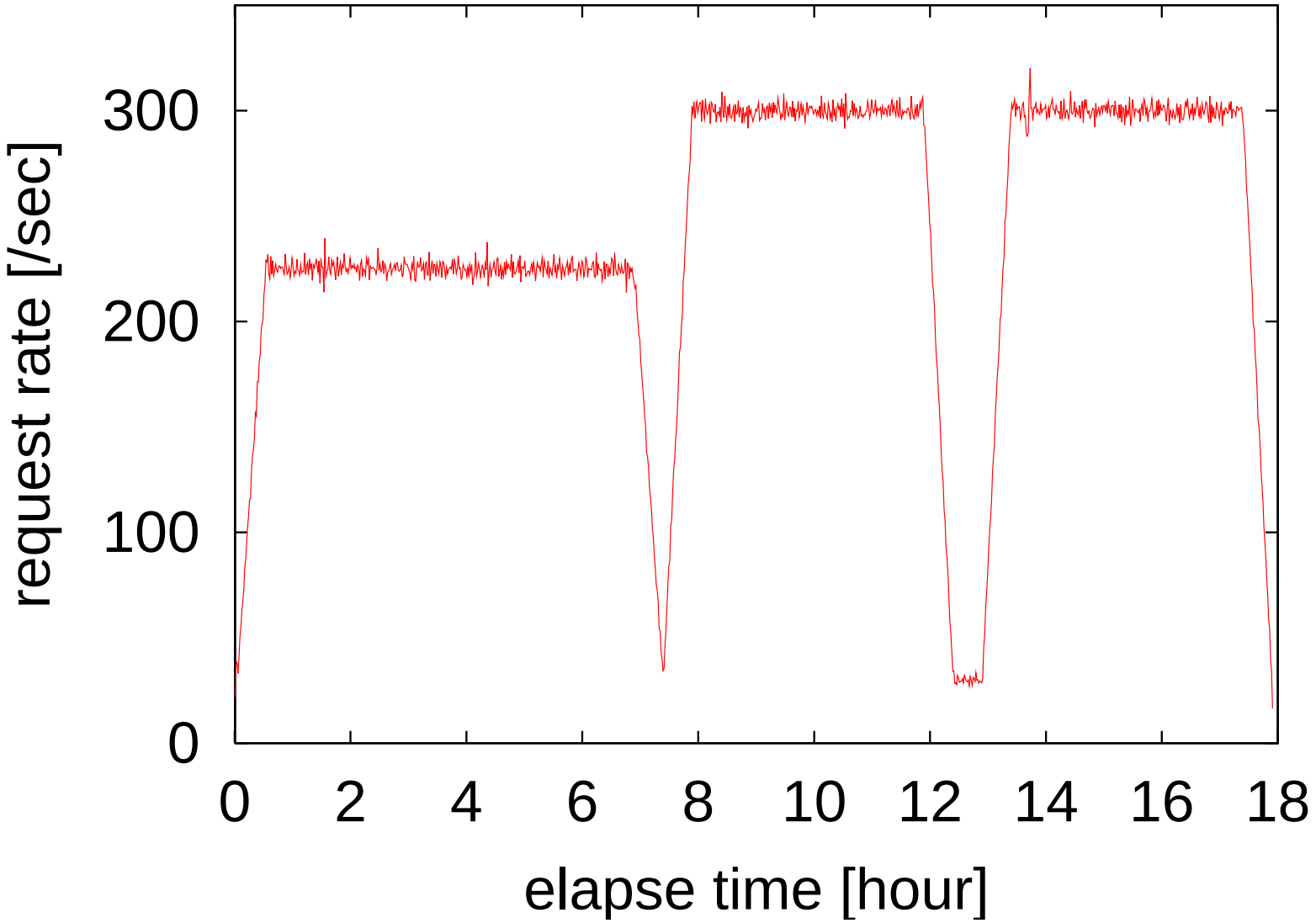
Polymix-3

- Cache Size = 31.3GB
- Peak Rate = 300rps
- Fill ratio = 75% (default)

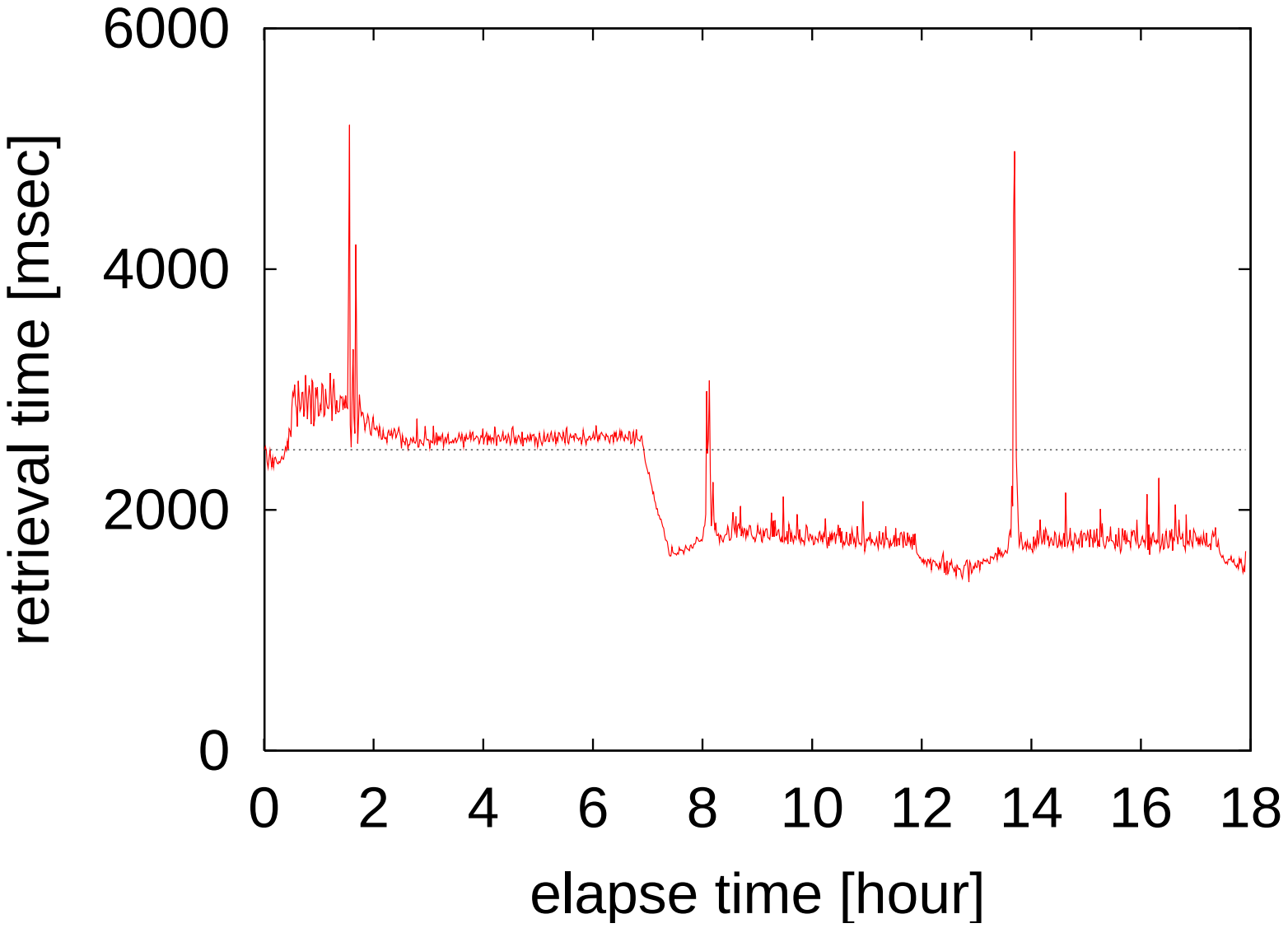
Result

- Response time = 1.792 sec (server reply = 2.5sec)
- Hit ratio = 42.91 %
- Error = 0.10 %

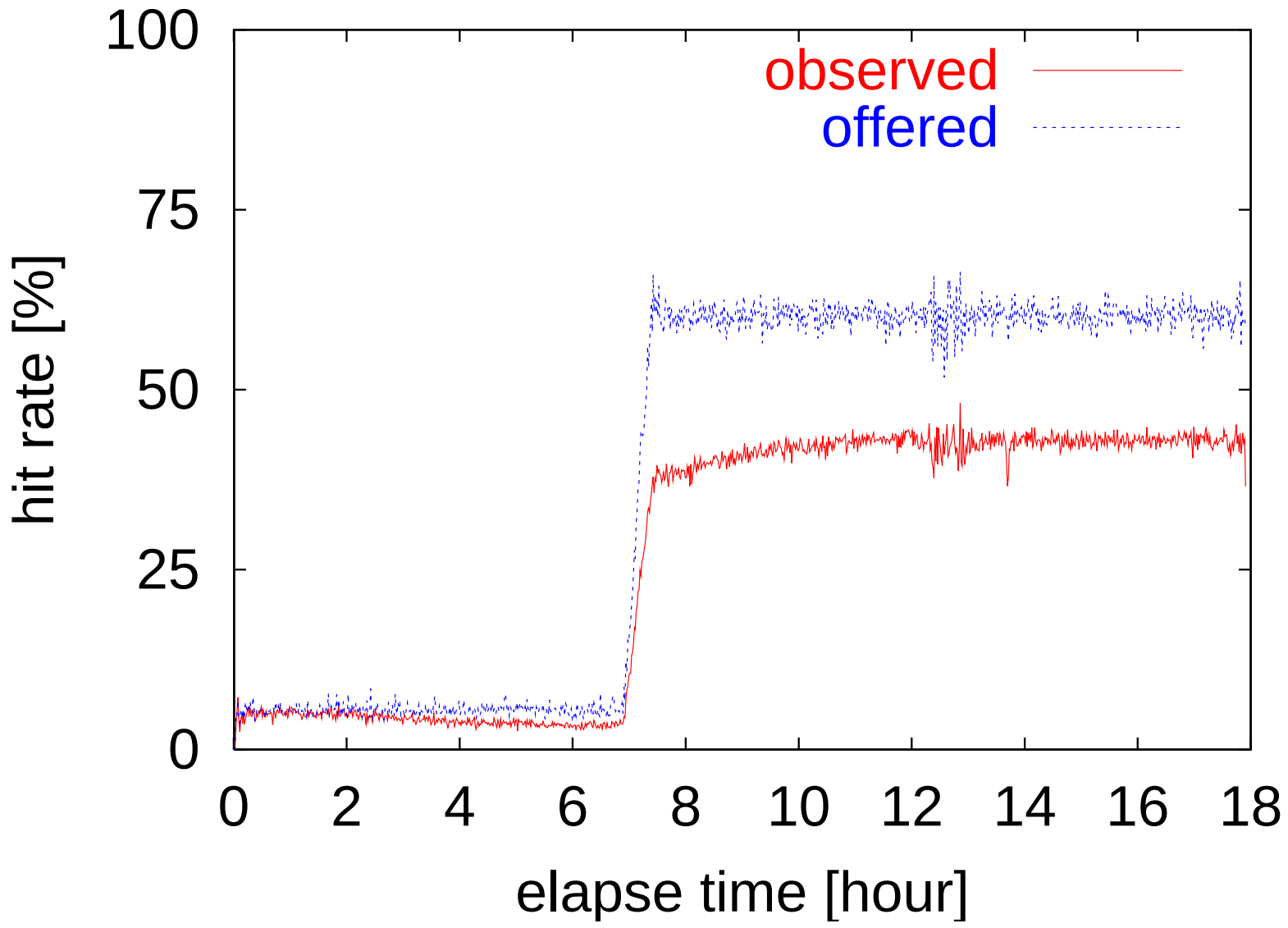
Request rate



Retrieval time



Hit rate


observed
offered
hit rate [%]
100
75
50
25
0
0
2
4
6
8
10
12
14
16
18
elapse time [hour]

Summary

- Design and implementation of WWW cache
  - ◇ Using general OS
  - ◇ Using typical filesystem interface
- Bottle-neck in WWW $\Rightarrow$ filesystem
- Our techniques to optimize disk I/O
- Evaluation

$\Rightarrow$ Achieves 300rps in benchmark

Future Works

- Operation in actual environments
- More public/authorized results

  $\rightarrow$ we register TMF's 4th Cache-Off
- Tunning

Contact Points

Our system is available at following page

```
http://infonet.aist-nara.ac.jp/products/kotetu/
http://infonet.naist.jp/products/kotetu/
```

Please contact me

k-chinen @ is . aist-nara . ac . jp

History of Our Team

1992:

- NAIST was funded

1993:

- Our laboratory was born
- We meet WWW (Mosaic)

1994:

- Proxy Server (CERN httpd) was born
- I made a prototype of prefetching proxy
  - ◇ network is slow, and narrow bandwidth

1995-1998:

History of Our Team (*cont.*)

- We developed the proxy (Wcol)
  - ◇ support ICP

1999:

- I start developing of brand new proxy
  - ◇ code-name is FALCON

2000:

- FALCON was released as ’KOTETU’

and here.

Object Packing - Example

- Cache capacity = 30GB
- Average object size = 10KB

1) Store to single file

```
30GB/10KB = 3M files
```

2) Store to packed file

- Block size = 8K
- The number of blocks in file = 1500

```
1500 / ceil[10KB/8KB] = 750 objects/file
750 * 8KB = 6MB/file
30GB / 6MB = 500 files
```

Necessary or not

- Date attributes (create, modify, access)
  - ◇ WWW cache have to record dates
  - ◇ But It needs many types
    - ⋆ (HTTP) Last-Modified, Expire, Date, If-Modified-Since
    - ⋆ (internal) create, requested, delete
  - ◇ Generic date attributes is not useful
- Directory

Rigstered System for 4th Cache-Off

Current plan

- Dual 1GMHz Pentium 3
- Four 256MB Registered ECC memory
- Five 10Krpm 9G SCSI disks for caching
- Single channel SCSI RAID card
- Single 10Krpm 18G SCSI disk for boot and logs
  - ◇ 5600 US$
  - ◇ Reach 300rps under polymix-4