

Analyzing Kernel Behavior by SystemTap – Fine Grained Kernel Behavior Monitoring –

NEC

OSS Platform Development Division
Atsushi Tsuji <a-tsuji@bk.jp.nec.com>



Contents

- 1. Background**
- 2. Example for Kernel Behavior Monitoring**
- 3. Issues for Kernel Behavior Monitoring**
- 4. Conclusion**



1. Background

◆ Kernel Behavior Monitoring

System issues like performance degradation, I/O delay or network delay sometimes occurs on working system

- **Fine grained monitoring is required to look into the root cause**

◆ Recent New Tracing Tools

lockstat, latencytop

These tools enable detailed monitoring

But, they cannot be enabled on working system because they need latest kernel or kernel patches and have a large overhead

◆ Existing Monitoring Tools

vmstat, iostat, sar

Very useful, but resolution is low (system-wide or per device)

- **The exhausted resources cannot be determined directly (we cannot determine this process)**

Fine grained kernel behavior monitoring is required



Kernel Behavior Monitoring by SystemTap

◆ Monitoring Resolution Improvement

◆ SystemTap enables fine grained kernel behavior monitoring

- ◆ Associating the events and the information in kernel
- ◆ Peeking in kernel variables

◆ Factors to determine kernel behavior

◆ I/O

Throughput, latency

◆ VM

Usage of anonymous memory, pagecache, slab, shared memory or swap

◆ Network

Throughput, latency

◆ CPU

CPU usage, scheduler



2. Example for Kernel Behavior Monitoring

- A) I/O Statistics (File-by-File Basis)**
- B) Pagecache Usage (File-by-File Basis)**
- C) Network Traffic per Service and Client**

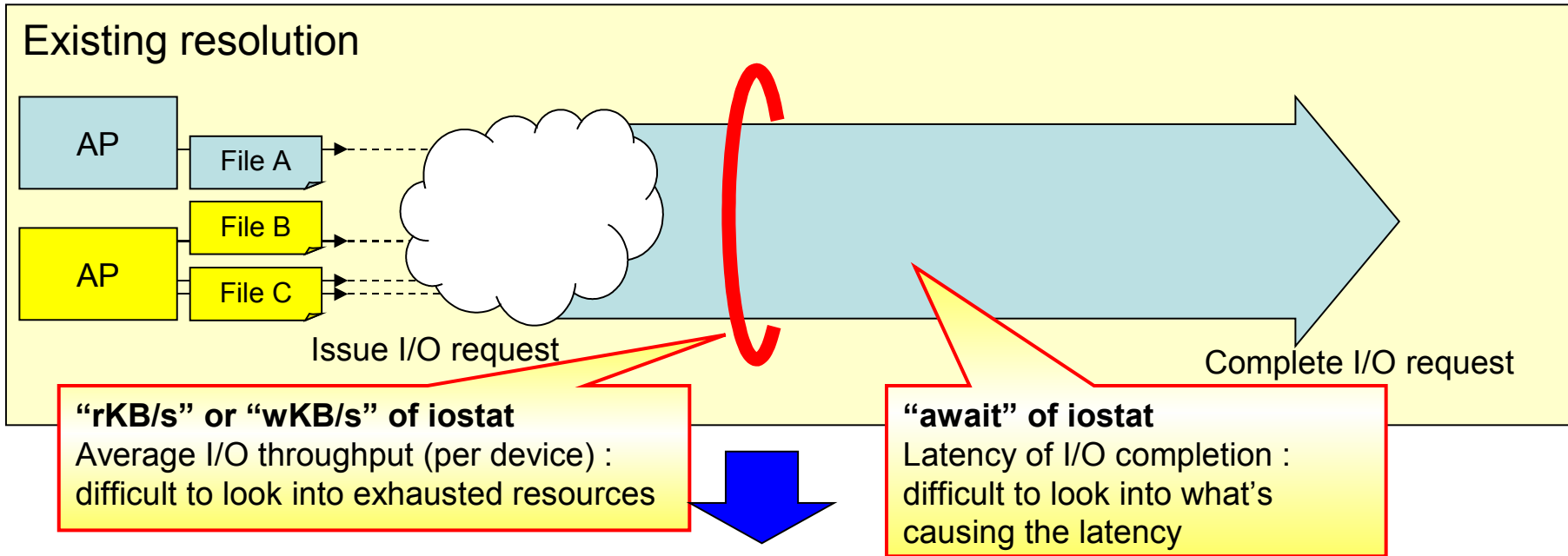


A) I/O Statistics (File-by-File Basis)

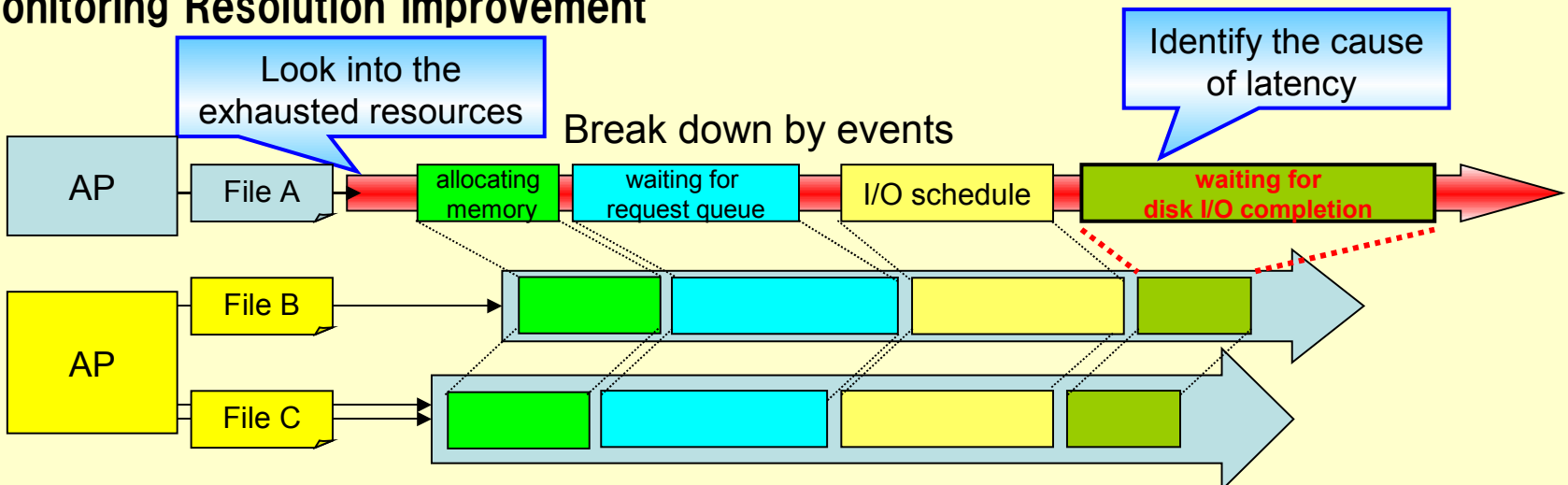
- ◆ **I/O behavior is difficult to understand**
 - ◆ **Various components and layers**
 - ◆ **Many actors influence each other**
 - ◆ **Applications can only know how much time is spent on read (2) or write (2)**
- ◆ **I/O Statistics is accounted on per device (e.g. iostat)**
 - **More detailed accounting is required**

Break down I/O activity into file-by-file basis

Monitoring Resolution Improvement

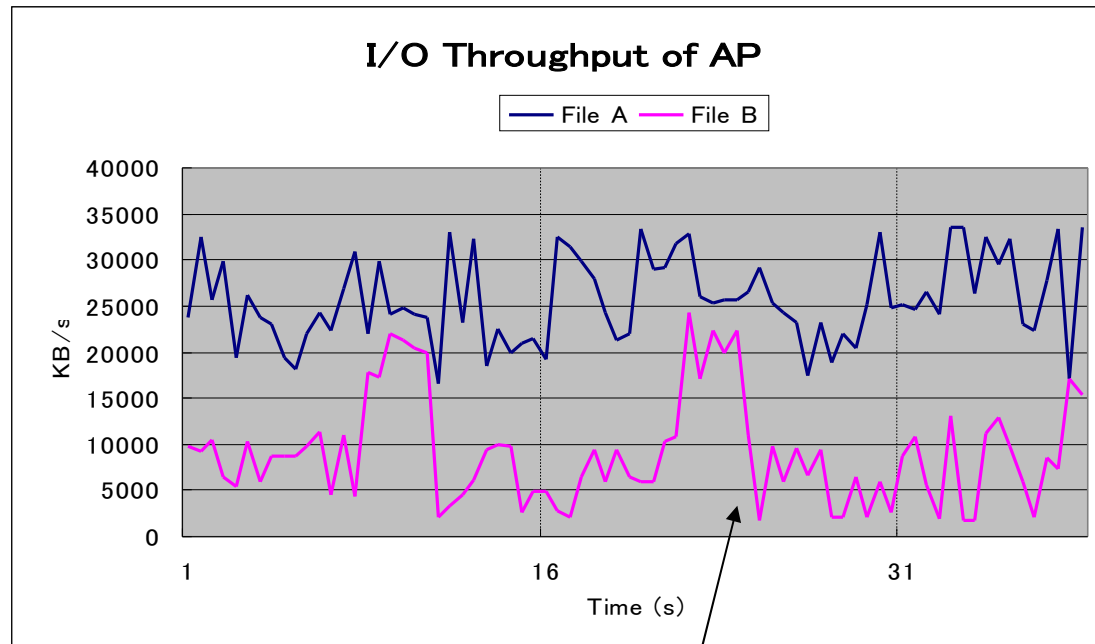


Monitoring Resolution improvement



Example for I/O Statistics (File-by-File Basis)

- ◆ **AP (I/O intensive job) slowed down dependent on output files (File A and File B)**
- ◆ **AP is same in both case**
- ◆ **Cannot grab the cause by coarse grained statistics**

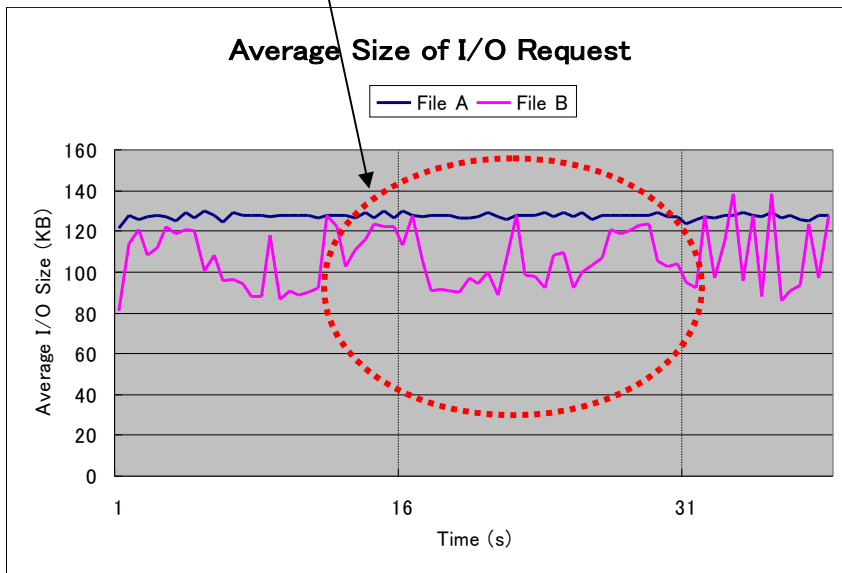


I/O throughput of File B is less than File A

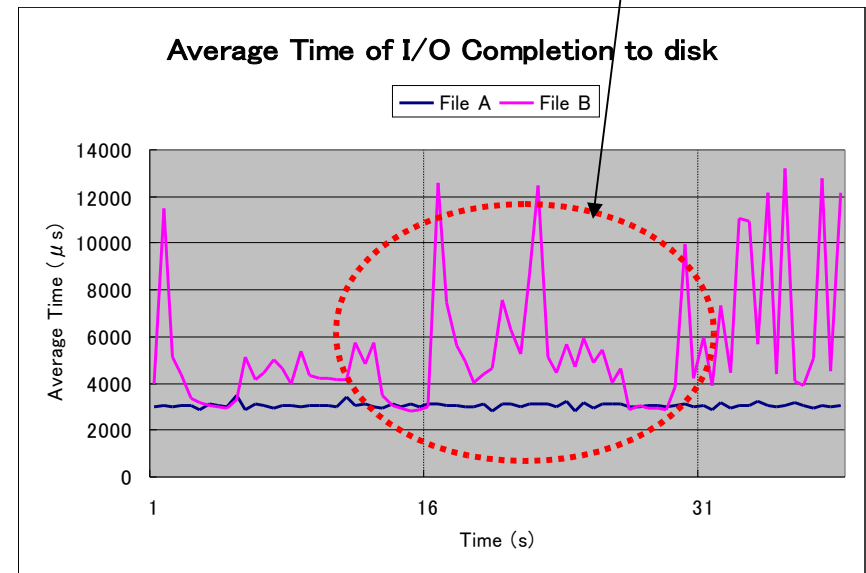
Visualization for I/O Statistics (File-by-File Basis)

Monitoring by breaking down I/O into File A and File B

I/O request size is reduced



I/O completion time to disk is increased

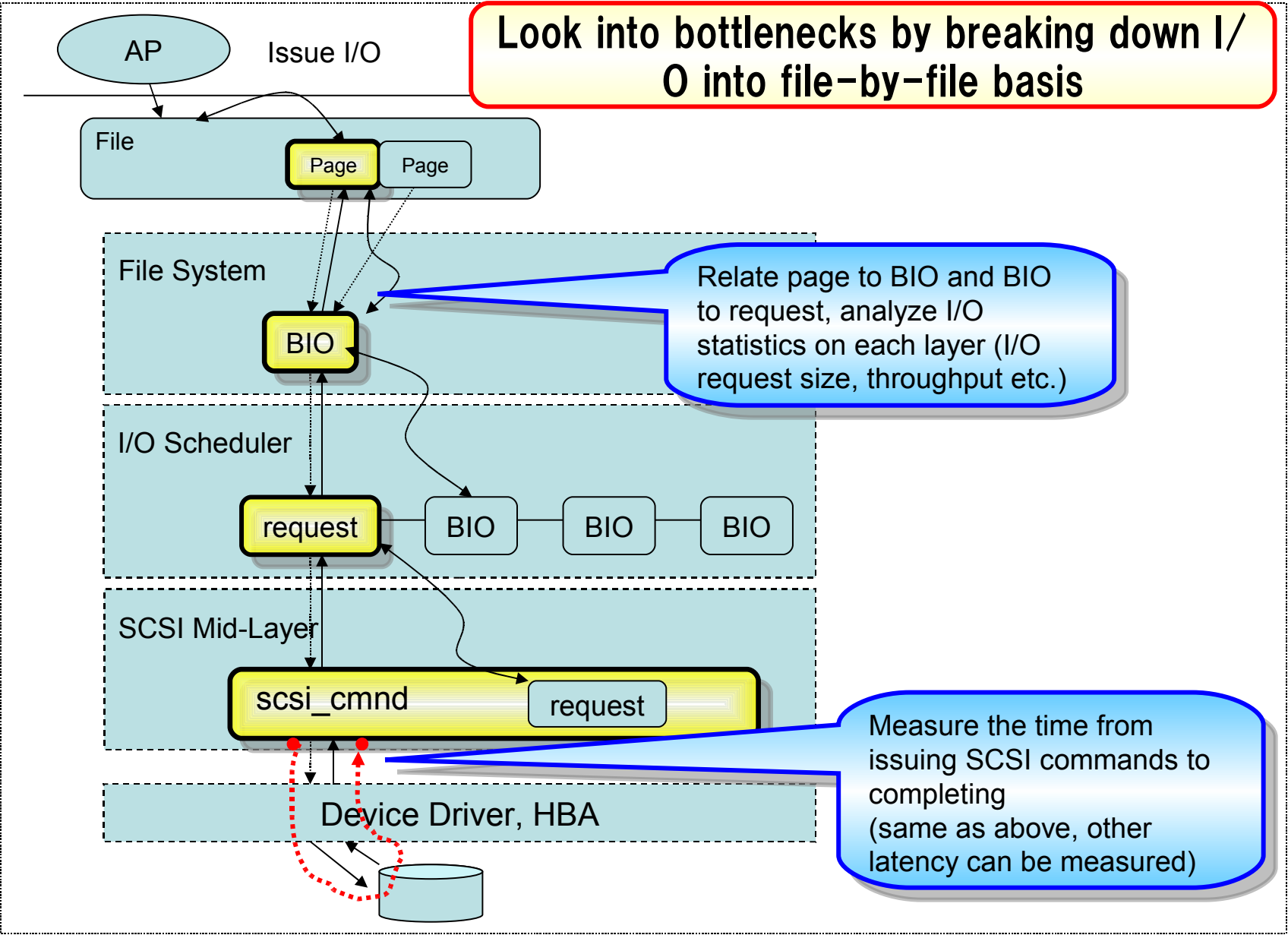


I/O request size of File B is reduced and I/O completion time is increased
=> Disk response become worse by bad placement of filesystem block and increasing disk seek

Analysis of file-by-File basis I/O activity becomes possible

Implementation of I/O Statistics

Look into bottlenecks by breaking down I/O into file-by-file basis



B) Pagecache Usage (File-by-File Basis)

- ◆ **Pagecache behavior influence filesystem and I/O activity**
 - ◆ **Pagecache misses make response time worse (memory access vs. disk access)**
- ◆ **Pagecache usage is accounted on the system-wide (“Cached” of /proc/meminfo)**
 - ⇒ **Accounting pagecache usage on a file-by-file basis by using SystemTap**
- ◆ **I/O efficiency decreases when accessed files are not on pagecache**

How and which files are using pagecache?



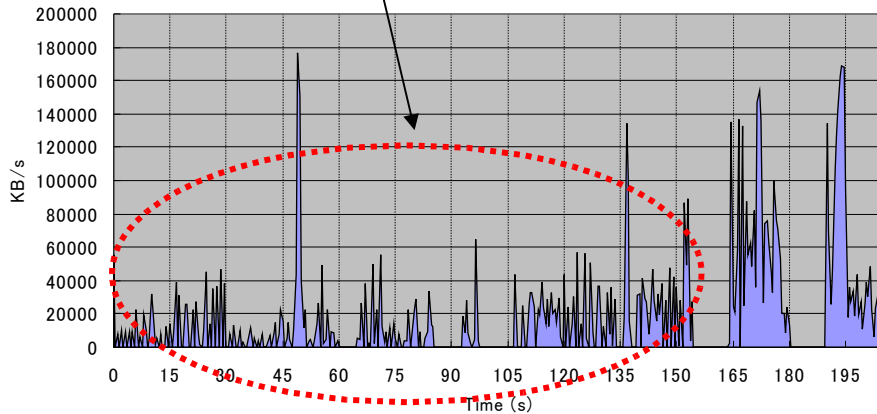
Example for Pagecache Usage

- ◆ **AP (I/O intensive job) issues I/O regularly**
- ◆ **I/O throughput of AP degraded**
- ◆ **The root cause cannot be determined because of low resolution**

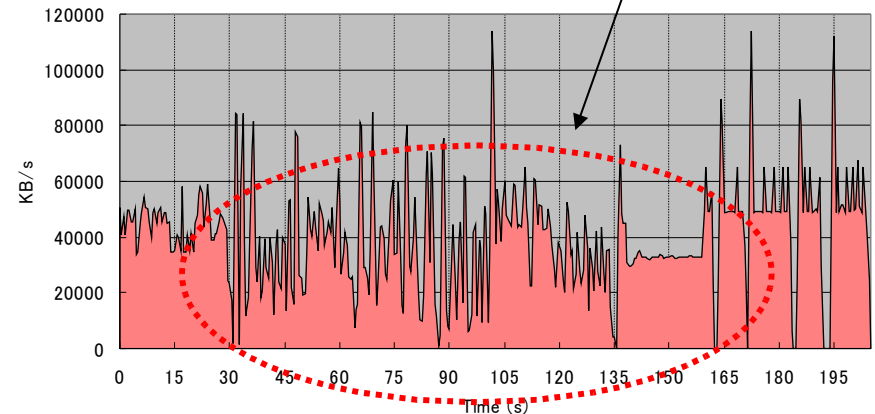
I/O throughput of AP degraded

I/O throughput for device looks normal:
multiple I/O cannot be broken down

I/O Throughput of AP

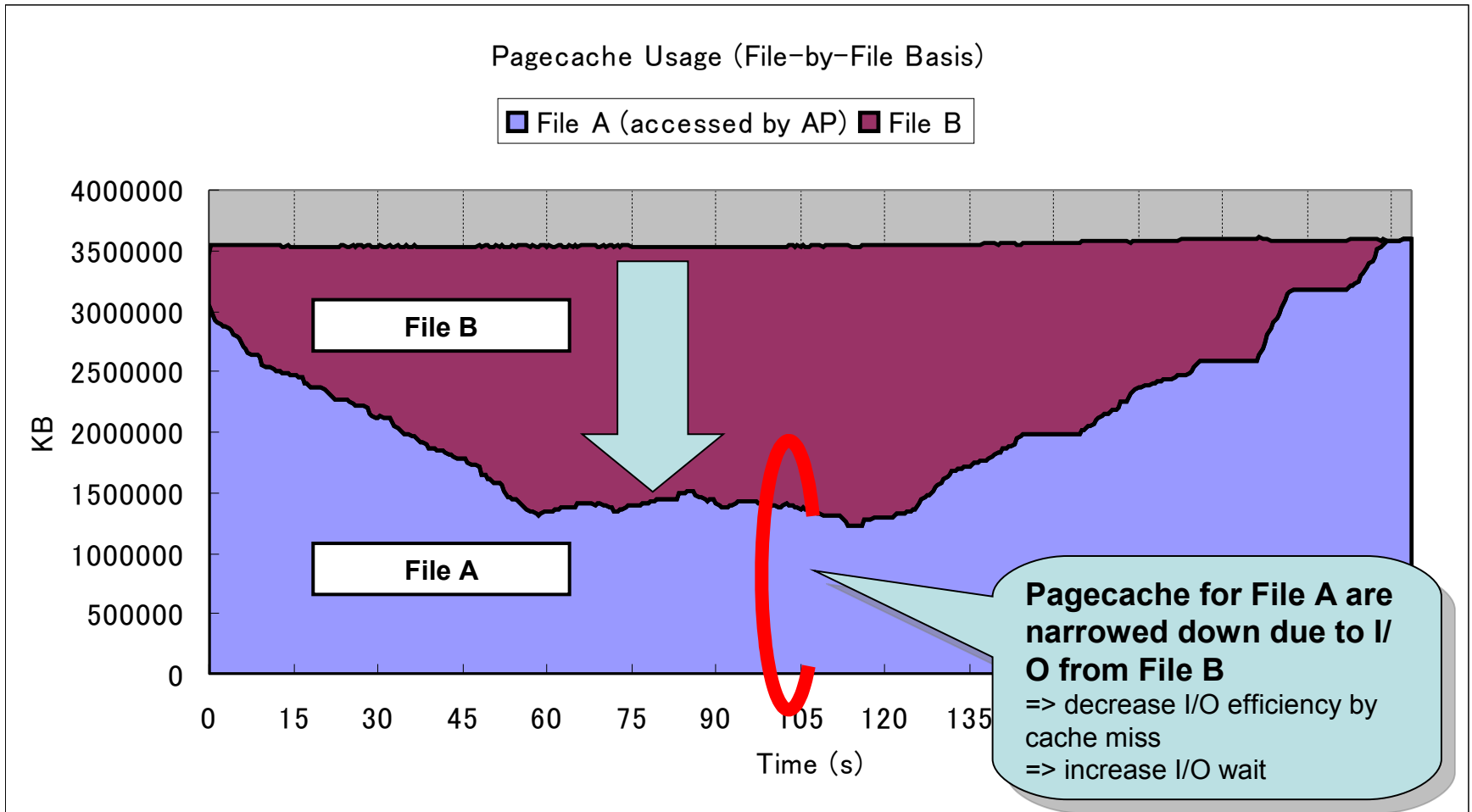


I/O Throughput for disk (iostat)



Difficult to look into root cause by existing tools

Visualization for Pagecache Usage (File-by-File Basis)

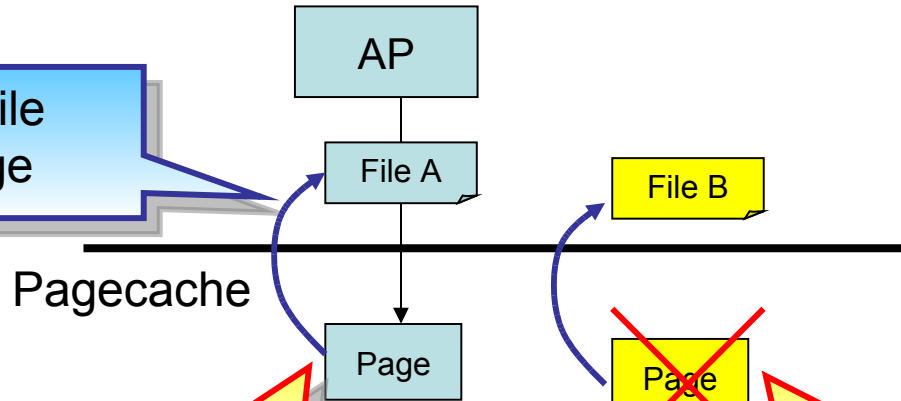


Analysis of file-by-file pagecache usage becomes possible



Implementation of Pagecache Usage

Retrieve the file including page



Increase Counter of Pagecache:
Page is added to pagecache when AP issues I/O

```
counter[file]++;
```

Decrease Counter of Pagecache:
Page is removed from pagecache, when page is reclaimed

```
counter[file]--;
```

Trace adding and removing to pagecache

=> Relate page to file including it

=> **Pagecache usage on a file-by-file basis**

Script Image of Pagecache Usage

Script Image of the Trace Function

```
probe kernel.function("add_to_page_cache_locked")
```

```
{  
  inode = $page->mapping->host->i_ino;  
  dev = $page->mapping->host->i_sb->s_dev;  
  cache[dev, inode]++;  
}
```

Retrieve inode
from page

inode = \$page->mapping->host->i_ino;
dev = \$page->mapping->host->i_sb->s_dev;
cache[dev, inode]++;

Pagecache is counted each
inode and device

Retrieve device
from page

```
struct page {  
  union {  
    struct {  
      unsigned long private;  
      struct address_space *mapping;  
      ...  
    }  
  }  
}
```

```
struct address_space {  
  struct inode *host;  
  struct radix_tree_root page_tree;  
  rwlock_t tree_lock;  
  unsigned int i_mmap_writable;  
  ...  
}
```

Output Example

```
# stap pgcache_stat.stp  
major=8 minor=3 inode=4482806 page=3004260, major=8 minor=3 inode=4482817 page=537964  
major=8 minor=3 inode=4482806 page=2930416, major=8 minor=3 inode=4482817 page=613248  
major=8 minor=3 inode=4482806 page=2901888, major=8 minor=3 inode=4482817 page=641408  
major=8 minor=3 inode=4482806 page=2890672, major=8 minor=3 inode=4482817 page=652920  
major=8 minor=3 inode=4482806 page=2879408, major=8 minor=3 inode=4482817 page=662136  
major=8 minor=3 inode=4482806 page=2868148, major=8 minor=3 inode=4482817 page=677496  
...
```

File A

File B

Pagecache of File A decreases

Pagecache of File B increases



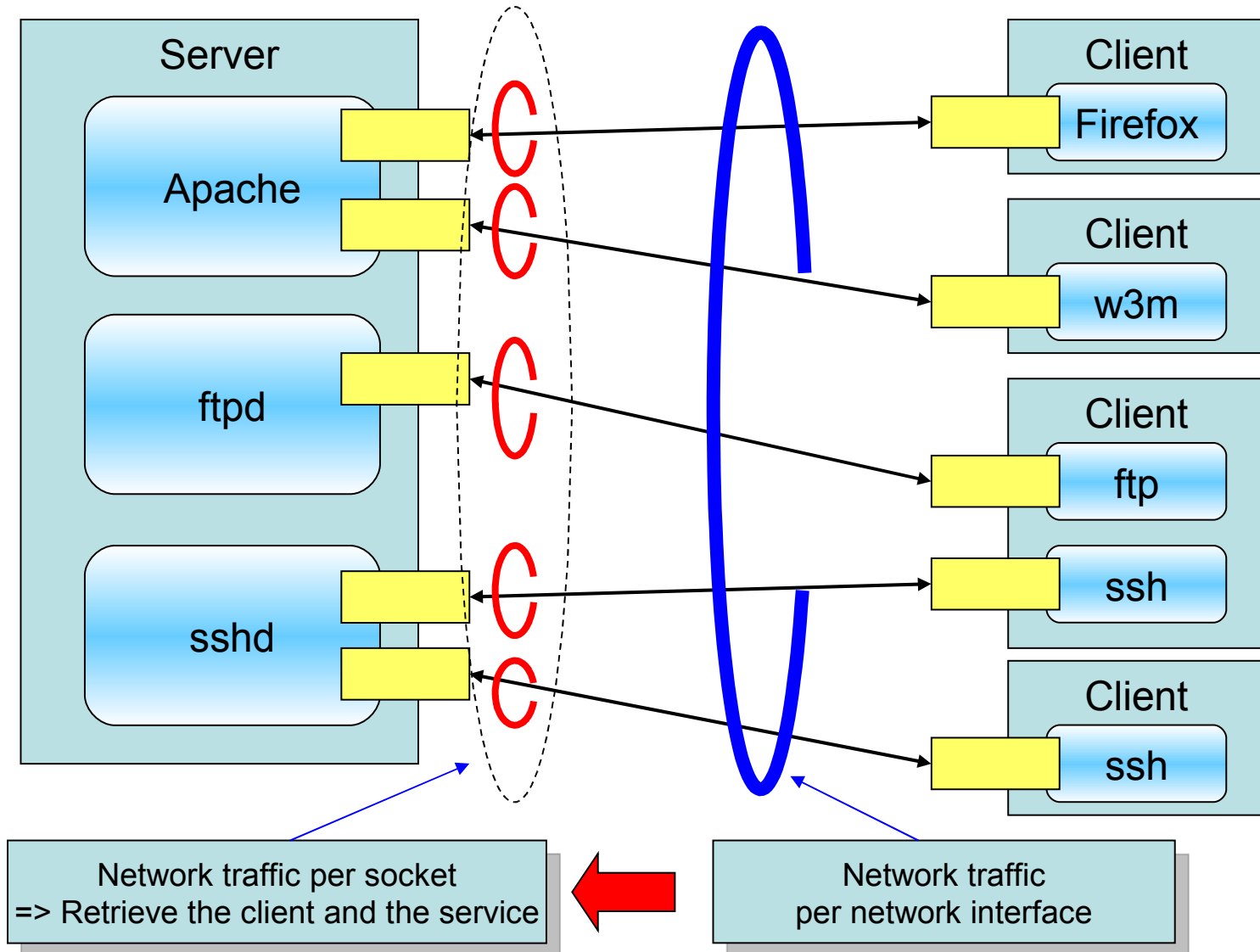
C) Network Traffic per Service and Client

- ◆ **The behavior of Network service is difficult to understand**
 - ◆ **Only the amount of network traffic per interface can be understood**
- ◆ **Traffic imbalance makes the performance of client AP worse**
- ◆ **The behavior of servers and clients can be understood by accounting traffic per service and client**

What processes are using network?

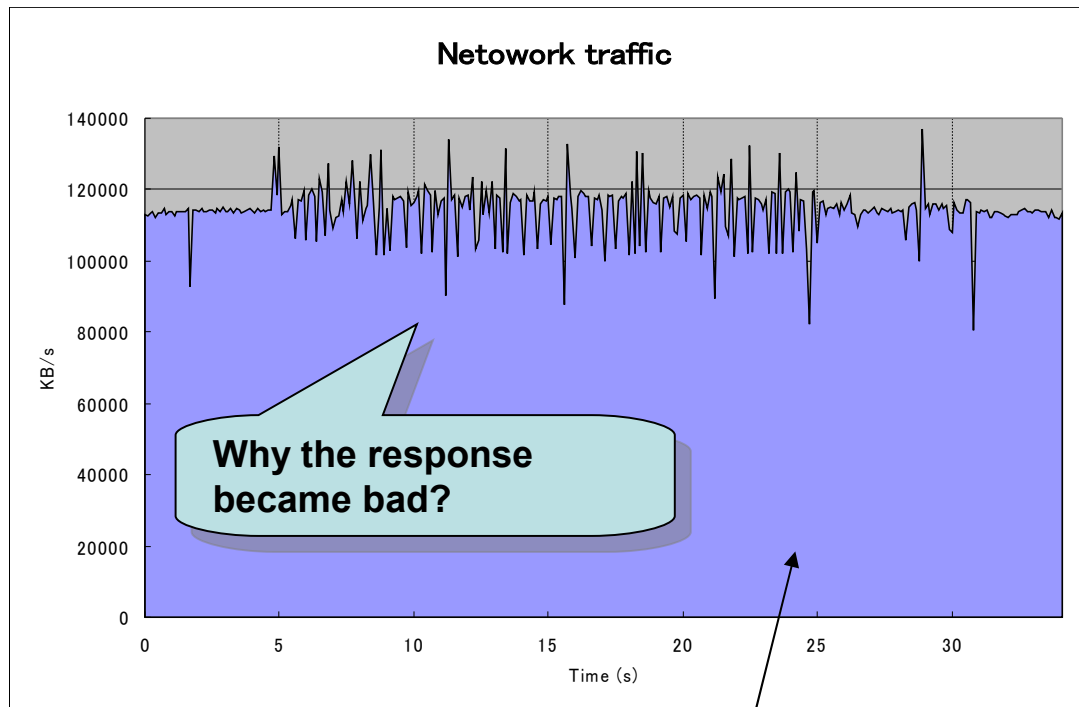


Network Traffic per Socket



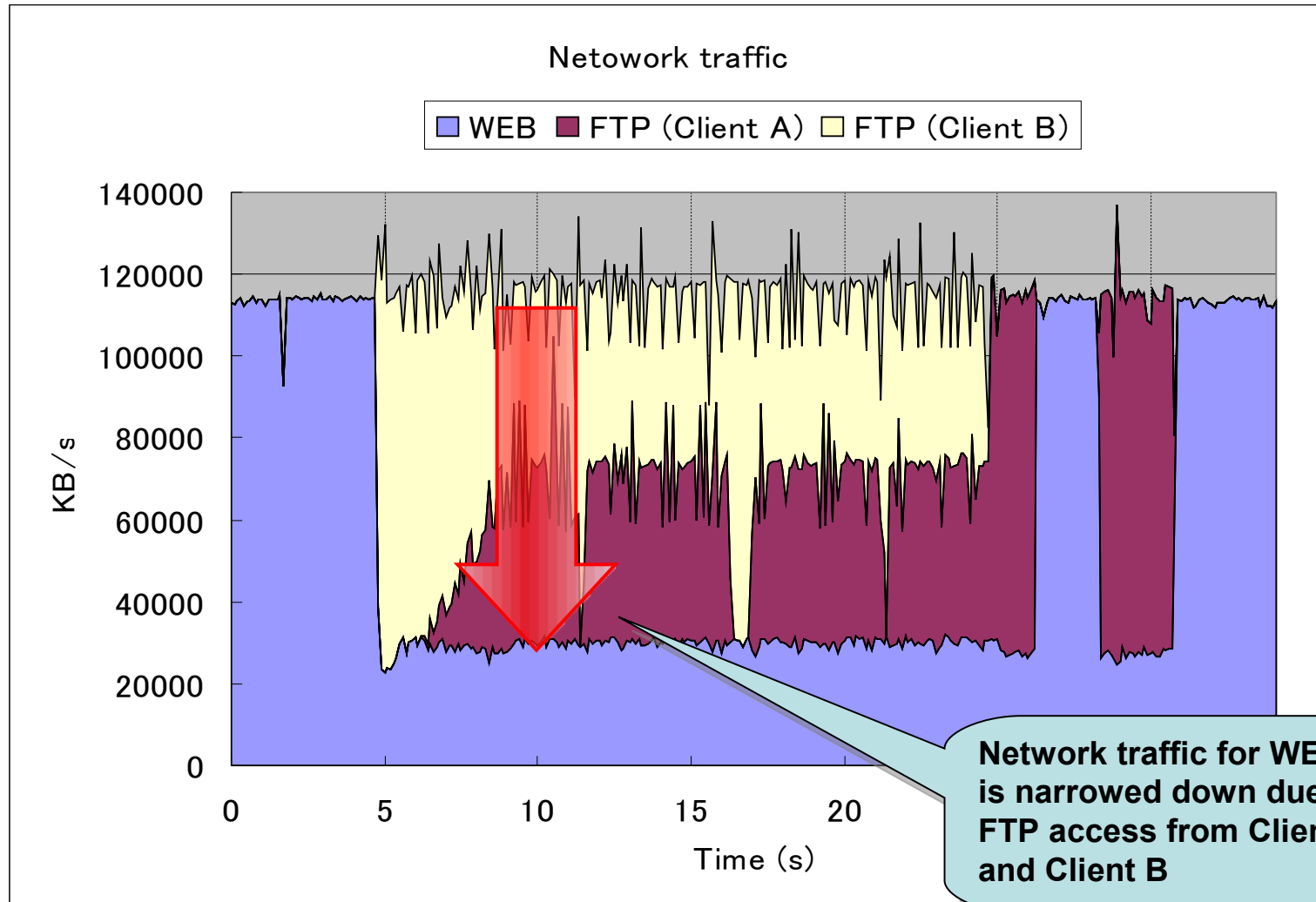
Example for Network Traffic

- ◆ **The server is used as Web and FTP server**
- ◆ **The response of web access became bad**
- ◆ **The cause of bad response cannot be determined**



Only the amount of inbound/outbound traffic per network interface is understood

Visualization for Network Traffic per Service and Client



Analysis of traffic per service and client becomes possible

3. Issues for Kernel Behavior Monitoring

SystemTap is very useful for kernel behavior monitoring, but...

- ◆ **Some Issues**

- ◆ **SystemTap influence working system**

- A) **System Overhead**

- B) **Scalability**

- ◆ **Restriction of tracing kernel behavior**

- A) **Kernel Data Accessibility**



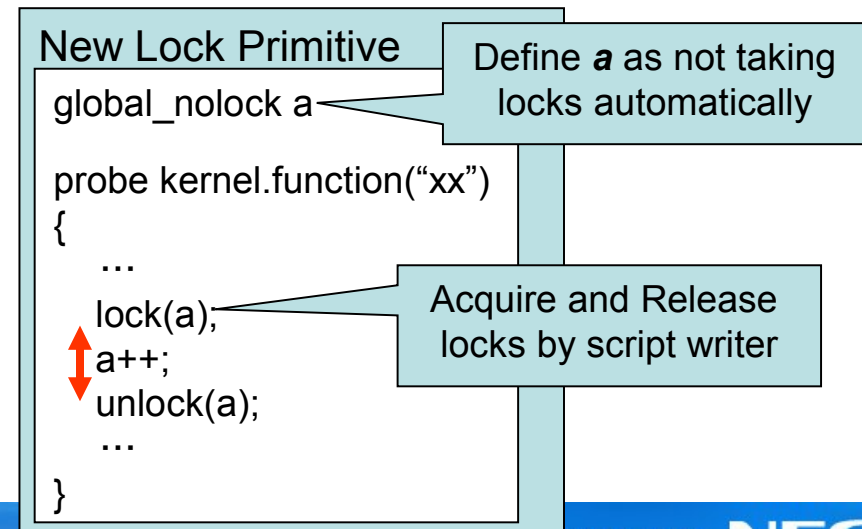
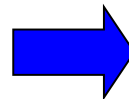
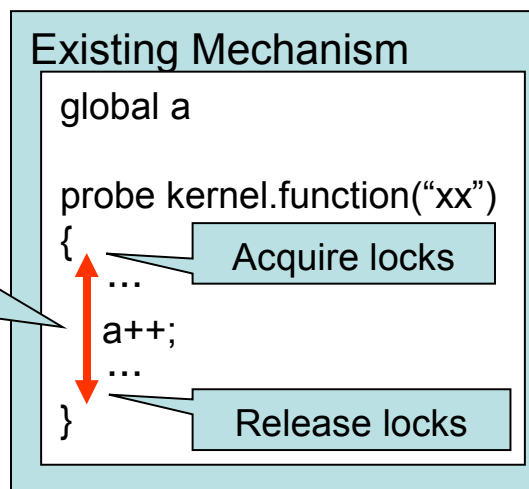
A) System Overhead

- ◆ **The overhead increases in proportion to the call frequency of probes**
 - ◆ **In the case of tracing system call, the frequency is determined by user operations**
=> The overhead is unpredictable
- **Solution by implementation :**
Trace internal kernel functions (independent from system call) to reduce probe frequency
 - **Probe function for pagecache management is called per page**
 - **The system overhead is 1% in practical use and 10%+ in worst case (all accessed files are on memory, I/O completes without disk access)**
- **Solution by infrastructure :**
Kernel Markers, djprobe
 - **The system overhead using markers can be greatly reduced (10%+ -> 4.5% in pagecache management)**



B) Scalability

- ◆ **When using global variables in stap scripts, locks are taken automatically during entire the whole of probes**
 - ◆ **Not need to care locks**
 - ◆ **No deadlocks**
 - ◆ **Probes are always atomic**
- ◆ **In many Core/CPU system (8 or 24 CPUs), coarse grained locks cause heavy lock contention**
- **Solution by implementation : Using embedded C function**
- **Solution by infrastructure (not implemented yet, just my idea) : New lock primitive using by script writer**



C) Kernel Data Accessibility

- ◆ **Not all kernel variable can be necessarily accessed**
 - ◆ **Inline functions**
 - ◆ **Local variables**
- ◆ **Some variables and inline functions could be removed by optimization**
- **Solution : Kernel Markers**
 - The needed kernel variables can be certainly accessed, because they are explicitly accessed through markers**

Advantages and Difficulties of Kernel Markers

◆ Advantages

- ◆ Reduce system overhead
- ◆ More detailed kernel data access

◆ Difficulties

- ◆ Kernel patches are needed
- ◆ Maintainers need to maintain markers because it is placed into kernel code

Need enough explanation for proposing Kernel Markers

Discussion Against New Trace Points

- ◆ **I Have Proposed Trace Points of Pagecache Management to LTTng and Upstream**
 - ◆ **These trace points merged in LTTng**
 - Need to appeal how useful the trace point is
 - ◆ **Frank gave comments :**
 - “It inspires more thinking about more places and ways for graphical data visualization”**
 - ◆ **Need proposing other trace points for kernel behavior monitoring**
 - ◆ **I will keep trying to propose useful trace points**
 - ◆ **Still discussing on upstream**
 - ◆ **Christoph and Kosaki said any tracer using this trace points is needed because not all the people use SystemTap**

If you find useful trace points when debugging, discuss them on community!



5. Conclusion

- ◆ **SystemTap enables fine grained analysis for various issues including kernel behavior**
- ◆ **SystemTap is powerful tool and could be used in many ways**
 - ◆ **Even for fault injection and games!**
 - **The SCSI Fault Injection Test**
<http://sourceforge.net/projects/scsifaultinjst/>
 - **SystemTap Game Collection**
<http://sourceforge.net/projects/stapgames/>

**Let's Use SystemTap for
Describing, Analyzing,
Understanding Kernel Behavior!**



References

◆ **SystemTap Project**

- <http://sourceware.org/systemtap/>

◆ **SystemTap Wiki**

- <http://sourceware.org/systemtap/wiki/>

◆ **LTTng Project**

- <http://ltt.polymtl.ca/>



