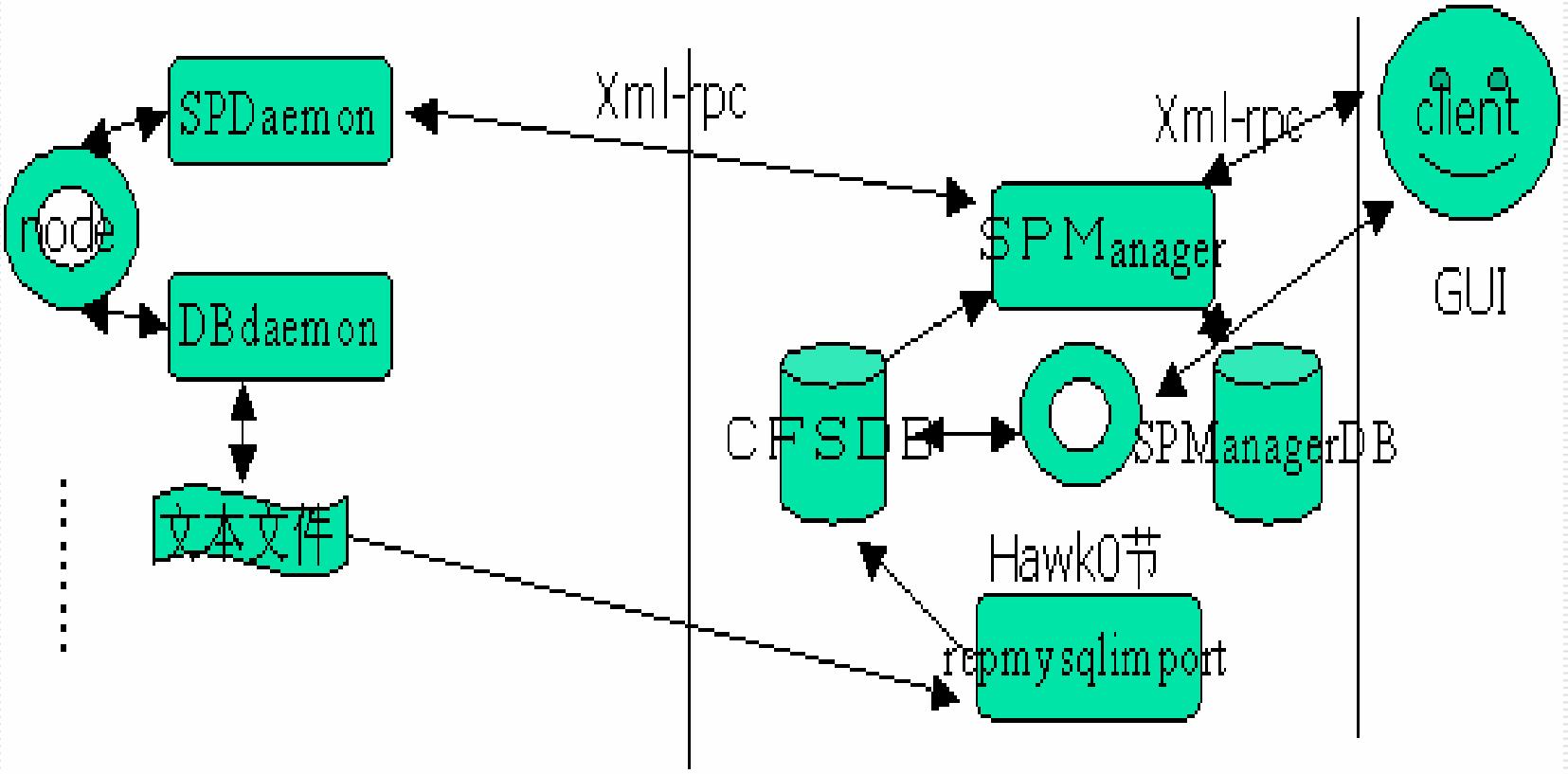
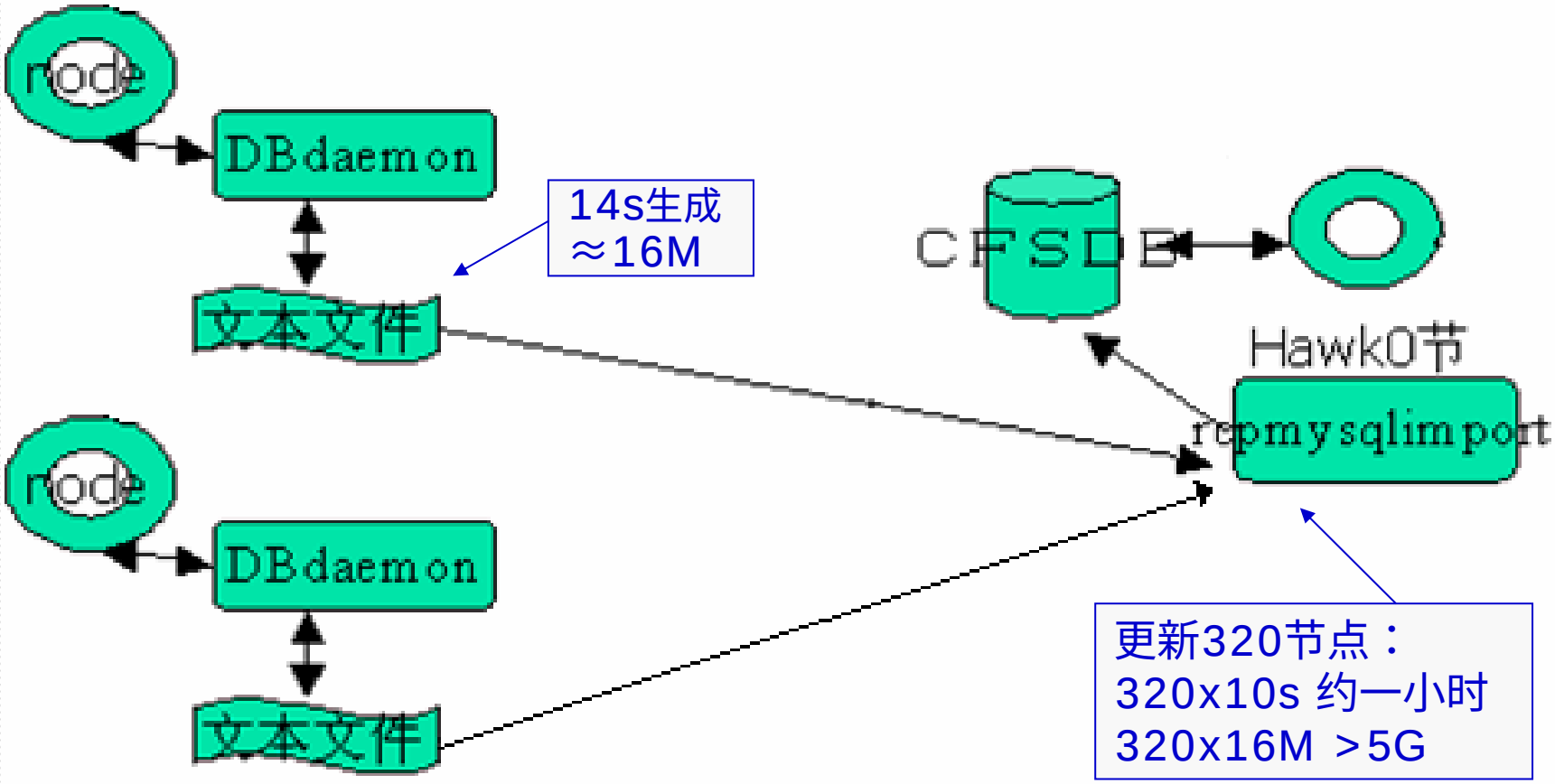
机群环境下文件系统监控的探索

智能中心 唐欢
htang@ncic.ac.cn
2003-5-13

问题出在哪里？（1）


SPDaemon
node
DBdaemon
文本文件
Xml-rpc
Xml-rpc
client
SPManager
GUI
CFSDB
SPManagerDB
Hawk0节
rcpmysqlimport

问题出在哪里？（2）



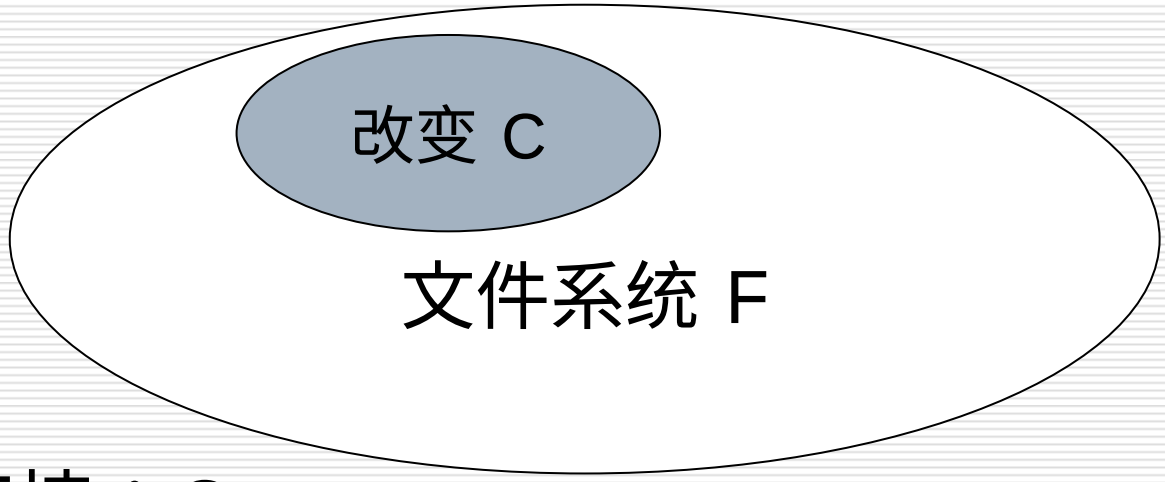
注：在之后的叙述中，将Dbdaemon生成的文本文件称为 “node文本”

目的

- 时间——减少Spreader每次更新文件信息数据库的时间消耗（主要原因：串行传送）；缩短更新间隔，提高数据库更新频率。
- 空间——减少更新数据量，降低对网络带宽的占用。

- 时间的减少依赖于空间的压缩，因此压缩node文本信息成为主要目标。

两个途径：



- 直接：C
- 间接：C=F - ¬C

解决办法

- 方案一：

  Module方法截获系统调用。包括：write()、mkdir() ……

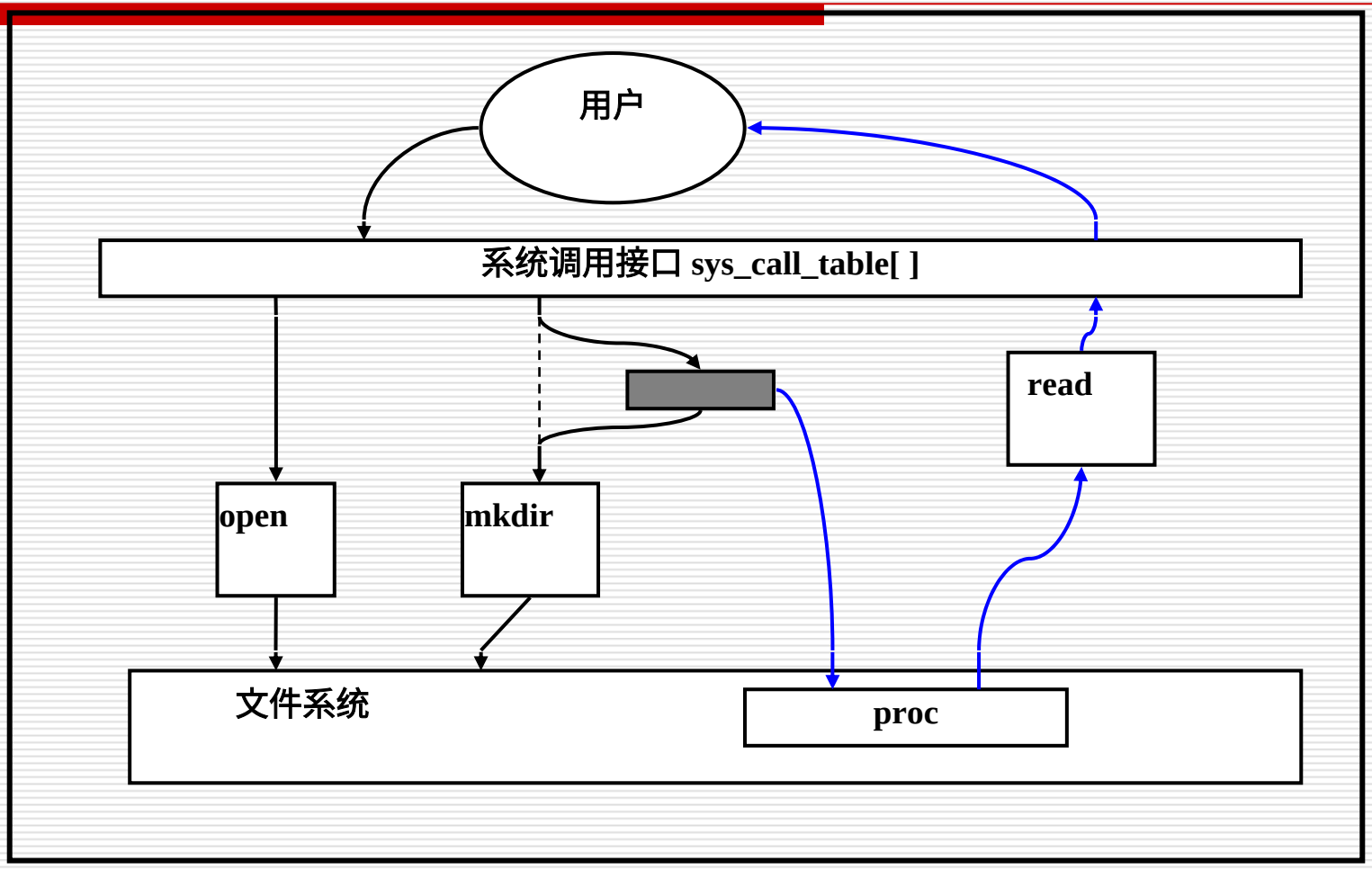
- 方案二：

  FAM（File Alteration Monitor）检测

- 方案三：

  将现有的node文本进行过滤

方案一： Module方法截获系统调用



步骤 1：运用strace跟踪系统调用

#strace mkdir tanghuan 的结果

```
......
open("/usr/lib/locale/en_US/LC_CTYPE",
    O_RDONLY) = 3
fstat64(3, {st_mode=S_IFREG|0644,
    st_size=173408, ...}) = 0
mmap2(NULL, 173408, PROT_READ,
    MAP_PRIVATE, 3, 0) = 0x40166000
close(3)                          = 0
umask(0)                            = 022
umask(022)                  = 0
mkdir("TANGHUAN", 0777)                   = 0
_exit(0)                          = ?
......
```

步骤 2：在asm/unistd.h中查找对应系统调用

/usr/src/linux/include/asm/unistd.h

内核版本：2.4.18-3

……

```
#define __NR_rename             38
#define __NR_mkdir              39
#define __NR_rmdir              40
```

……

```
#define __NR_fremovexattr       237
#define __NR_tkill              238
```

步骤 3：module编码片断

```
int init_module()
{
    create_proc_read_entry("capute_mkdir",      /* entry name */
                0,                              /* default mode */
                NULL,                           /* parent dir */
                me_read_proc,                   /* reda_proc function*/
                NULL                            /* client data */
                );

    original_mkdir = sys_call_table[__NR_mkdir];
    sys_call_table[__NR_mkdir] = me_mkdir;

    printk("<1><<<<<<  Capture sys_mkdir() Module >>>>>>\n");
    return 0;
}
```

记录结果：

++++++++
{egg_0}
++++++++
{egg_1}
{egg_2}
++++++++
{egg_3}
{egg_4}
{egg_5}
++++++++
{egg_6}
{egg_7}
……

问题、难点：

1. 安全性：替换系统调用，黑客用的方法能安全吗？
2. 稳定性：生存于内核中的活动分子
3. 效率：proc存在于内存，但截获所有的系统调用后是否仍然高效?
4. 能力：如何得到当前路径？能否截获所有更改文件系统的调用？如何截获文件转向‘>’、管道操作‘|’？是否有无法避免的干扰项(socket、swp)？
5. 隐患：替换自己；被别人无意替换；

解决办法

- 方案一：

  Module方法截获系统调用。包括：write()、mkdir() ……

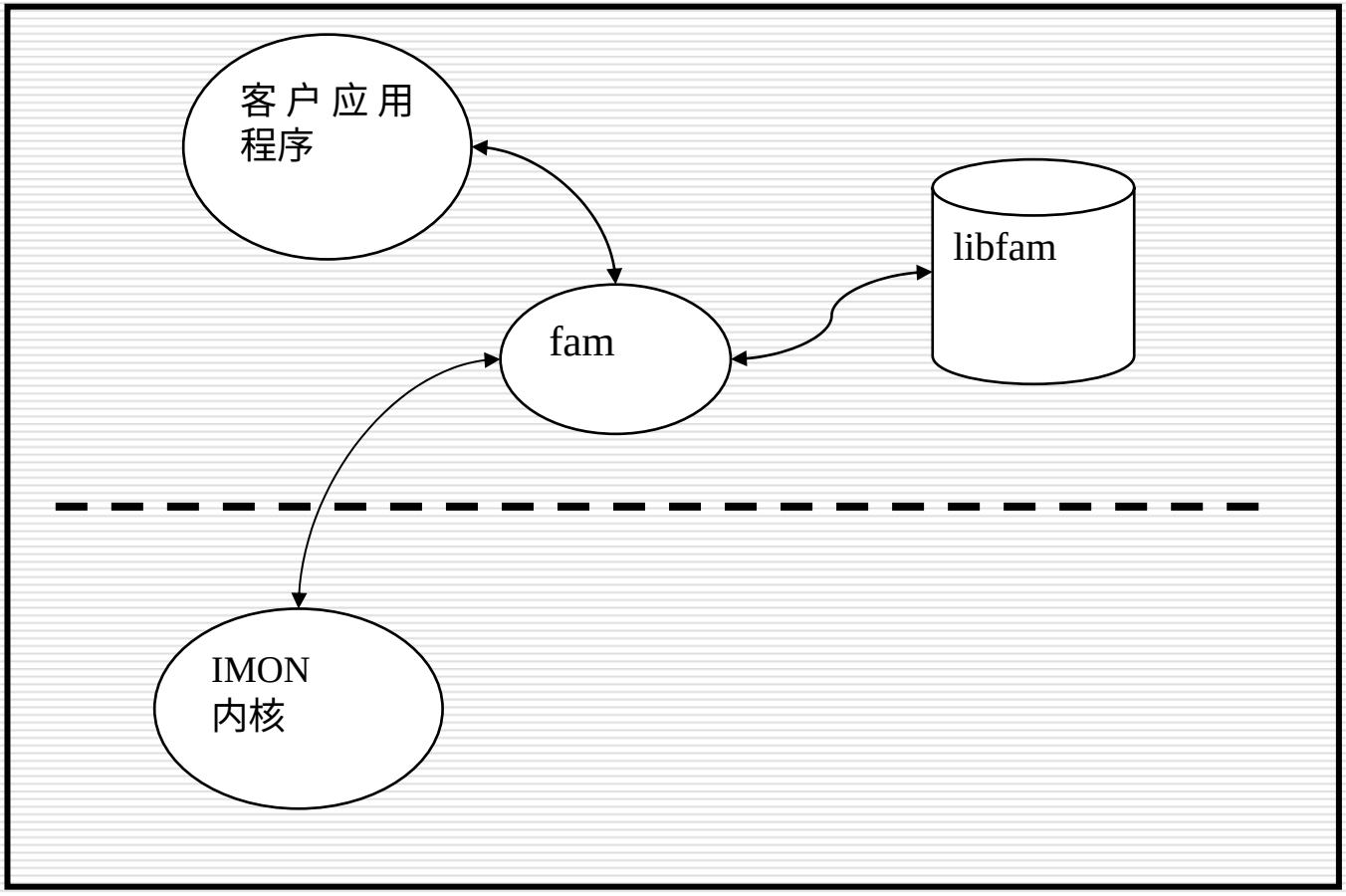
- 方案二：

  FAM（File Alteration Monitor）检测

- 方案三：

  将现有的node文本进行过滤

方案二：FAM（File Alteration Monitor）检测



FAM各部分功能：

- Imon(inode monitor)信息节点监视模块，它是内核的一部分，当文件有变动时由它通知fam。
- fam(file alteration monitor)后台守护程序。它负责：
  - 接收客户请求和向客户发送通知；
  - 向imon注册，接收imon通知；
- 库文件libfam，对用户透明。

实现：FAM－API

- 可记录的事件：

“FAMChanged”,	更改
“FAMDeleted”,	删除
“FAMStartExecuting”,	运行
“FAMStopExecuting”,	运行结束
“FAMCreated”,	创建
“FAMMoved”,	（未实现）
“FAMAcknowledge”,	关闭反馈、错误
“FAMExists”,	监视对象存在
“FAMEndExist”	监视对象列表结束

结果记录：

host	file&dir	events
……		
(null)	test1	FAMDeleted
(null)	newfile	FAMCreated
(null)	newfile_newname	FAMCreated
(null)	newfile_newname	FAMDeleted
(null)	newfile	FAMChanged
……		

问题、限制：

- 监视一层目录；
- 同时向一个进程提供1000个服务；
- 提供从监视路径开始的相对路径；
- 升级imon要重新编译内核！

退一步：监控proc文件系统

- 用fam监控proc文件系统下的所有进程文件夹，其中的fd目录中会存放所有的文件描述符。记录该描述符，当进程终止后更新描述符所指向的文件。

监控proc带来的问题：

1. 进程的创建和删除不会马上更新；why？
2. 检测不到的命令－cp rm …
   shell命令一定在/proc/下生成目录吗？
3. 文件描述符 －> socket 怎么办？

解决办法

- 方案一：

  Module方法截获系统调用。包括：write()、mkdir() ……

- 方案二：

  FAM（File Alteration Monitor）检测

- 方案三：

  将现有的**node**文本进行过滤

方案三：将现有的node文本进行过滤

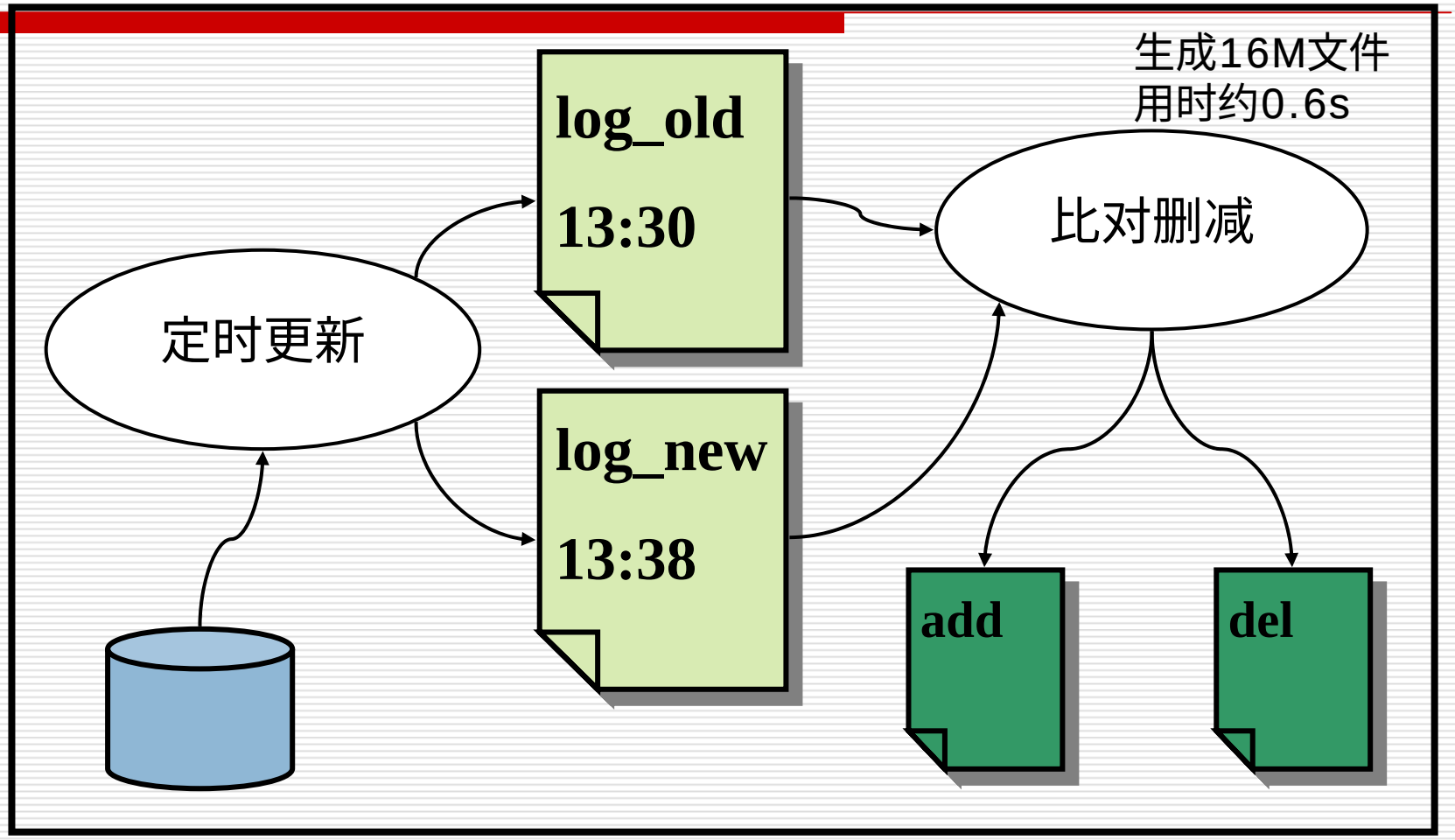
两个方向：

- 直接检测改动的文件（或目录），以此生成node文本。module、FAM
  - 优点：node文本直接生成，快速
  - 缺点：限制条件多、实现难度较大
- 将已有的node文本的冗余记录（即无改动的记录）去除，得到一个无冗余的node文本。
  - 难点：去除冗余的策略

依据：

- 整个文件系统的变化不是平均分布的，比较集中。例如：/home 变动较频繁；
- 相当的部分是变化缓慢的，甚至是长时间不变的。没有必要在每次更新的时候都将他们的信息重新提交数据库
- 对数据库的使用，现在只是删除表、创建表，操作粒度在“表”。应充分利用数据库的特性，将操作粒度深化为“记录”。

示意：得到变动记录



总体对比：

	已有的方案	新的方案
node文本内容	整个文件系统信息	两次更新间隔中有改变的记录信息
node文本大小	约16M(cnode1)，依赖于文件系统的大小	依赖于两次更新间隔中变化的多少
数据库操作	删除、创建表	添加、删除记录

记录级的效能对比：

记录数变化对两种方案的比较			
操作	原有方案	新方案	相对变化
创建	+1	+1	0
删除	－1	＋1(删除项)	＋2
更改	＋0	＋2(先删除再增加)	＋2

理论分析：临界点1/2

1. 对于处理创建操作，新方案一定比原有方案更有效率；
2. 对于处理删除操作和更改操作，当操作记录数低于原有记录总数的1/2 的时候，新方案将优于原有方案;

例子说明一切：临界点的大小直接影像到方案优劣

假设：文件系统中，文件数与目录数总和为90,000			
操作	原有方案	新方案	新方案是否优
创建10,000	100,000	10,000	√
删除10,000	80,000	10,000	√
删除45,000(1/2)	45,000	45,000	×
更改10,000	90,000	20,000	√
更改45,000(1/2)	90,000	90,000	×

补充说明：临界点X>1/2

以上例子得到有关“临界点1/2”的结论仅仅是对记录的数目而言，并不是记录的大小；而在实际方案比较中，我们关心的是记录文件的大小。

对于要创建的文件记录，每个记录格式：

“xmlrpc.h 25 r root root rw-rw-r-- 163522 2056 1 2003-1-7-17-21-7 163521 2056 ”

二元组<ino,dev>可以唯一确定一条记录，那么当需要删除该文件，则只需记录：

”163522 2056“

由此可得：新方案是否优的临界点X>1/2

遗留问题：关于更新频率的设置

由于文件记录长度的不确定性，以及创建、删除、修改三种操作发生的无规律性，使得无法在理论上算出新方案“是否为优”的临界值；而对于不同类型的操作集合来说，这个临界值也是不断变化的。

要想达到原有目的，则要以这个临界值为参考设置更新频率。

设置方式：1、人工；2、自动调整

下一步做什么？

- MySQL的并行访问能力有多强？极限？
- 提高MySQL数据库的并行访问能力。将成倍的缩短数据库的更新时间！

假设：更新一个节点用时10s，

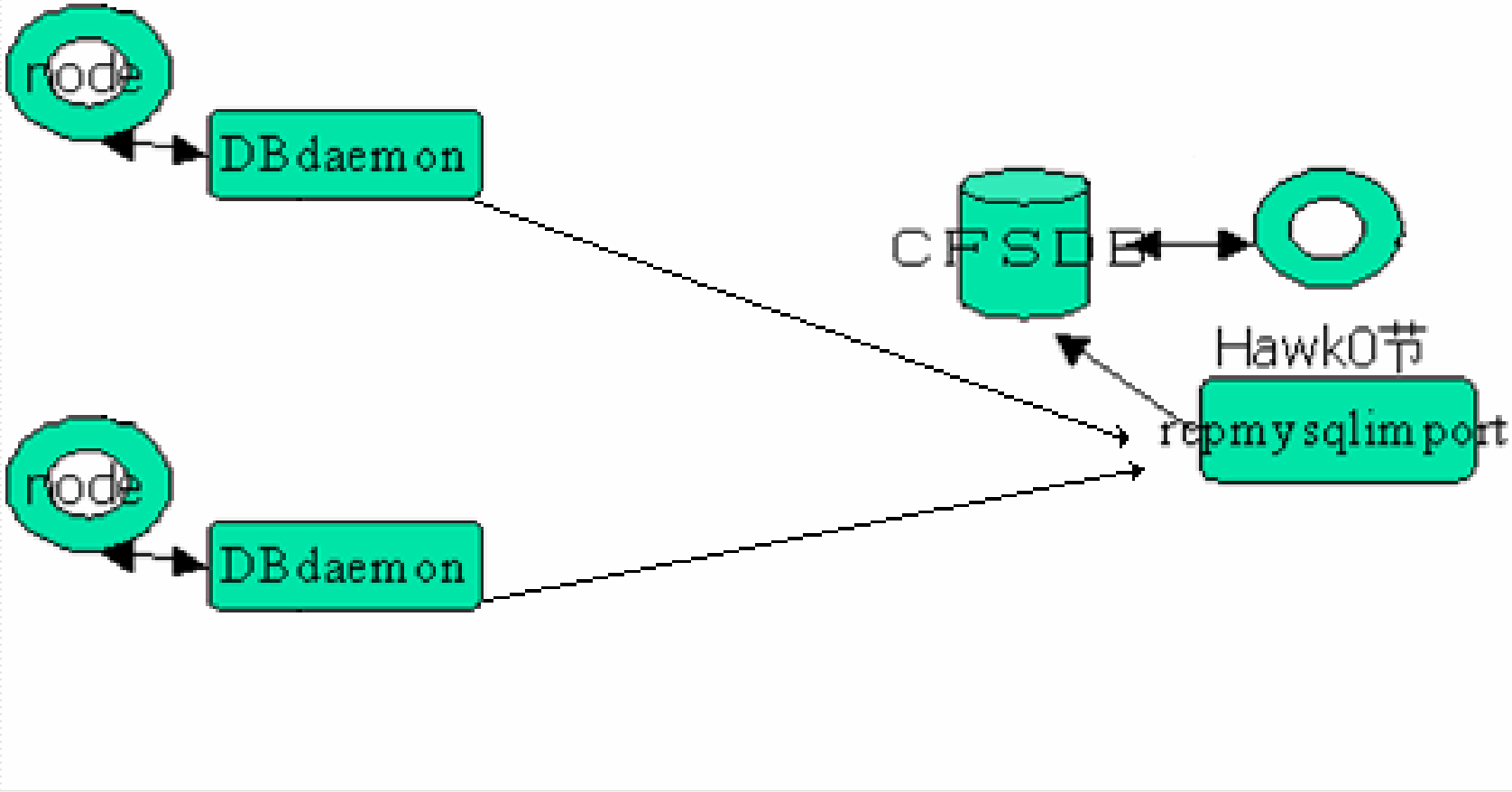
那么：串行更新320个节点：3200s

如果：每两个节点并发更新用时 18s（省2s）

那么：更新320节点：18s X 160＝2880s

省时：320s，10%！

更高目标：取消node文本



更多的思考：

- Module存在于内核空间，可以运用内核中的数据结构与函数；
- Module编程的过程是基于对源代码的分析理解，但是module的实现并没有修改源代码。可以在任何时候将module加载、卸载；
- FAM中的imon模拟出一个设备驱动，用socket 与用户空间的守护进程通信；恰恰module最广泛的用途就是编写设备驱动程序；

换个角度看： inode是如何创建与改变的？

```
[tang@localhost tmp]$ stat file.hello
  File: "file.hello"
  Size: 66              Blocks: 8
  IO Block: -4611691722143952896 Regular File
  Device: 802h/2050d
  Inode: 135420      Links: 1
  Access: (0644/-rw-r--r--)
  Uid: (    0/    root)   Gid: (    0/    root)
  Access: Wed May 14 09:59:40 2003
  Modify: Wed May 14 10:42:26 2003
  Change: Wed May 14 10:42:26 2003
```

Module＋FAM＝？

- 运用module方法进入内核，但是不截获所有的系统调用，而是找到对inode信息更改的处理函数（内核实现中是否是集中更改？），试图对其进行信息的截获。
- 深入研究FAM的实现过程，特别是imon的实现，突破其“一层目录”、“1000个服务”的限制，将其运用于module当中。

谢谢！

曙光3000一个最大的技术特点是，在不改变操作系统源码的基础上创新。

——孙凝晖

《民族精神 大成智慧--曙光3000超级服务器诞生叙事》