A live data synchronization using clsync

Andrey Savchenko

NRNU MEPHI, Moscow, Russia

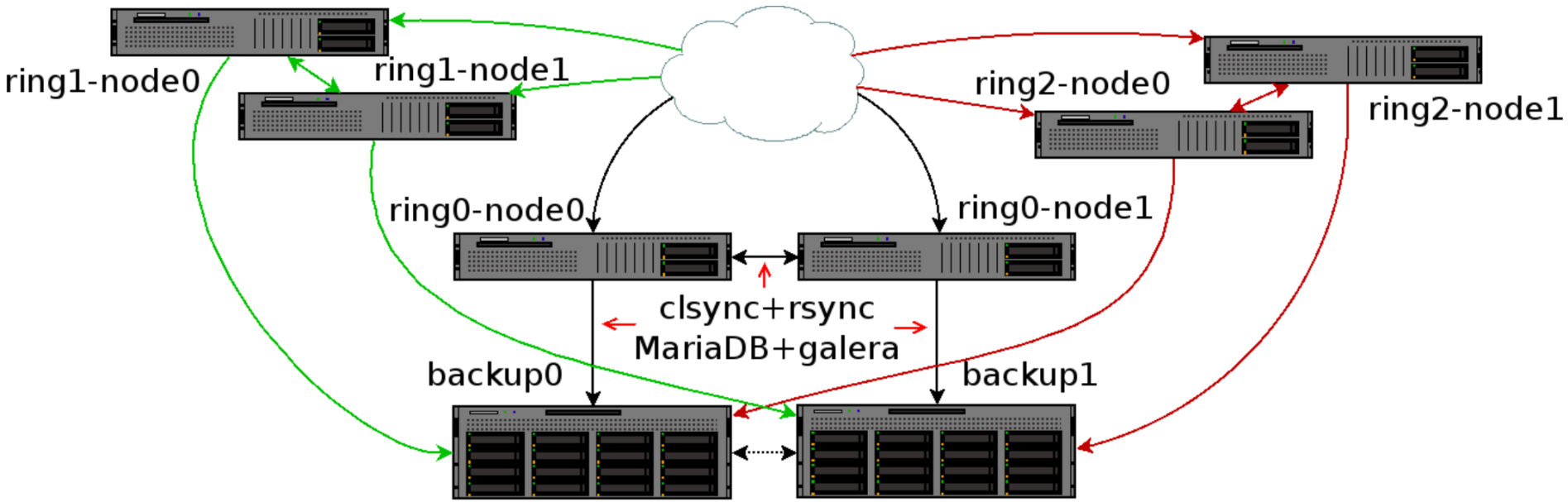
08 Oct 2016

Motivation

Our task:

- Deploy HA/LB clusters:
  - hosting
  - VoIP
  - corporate services
- HPC systems management and synchronization
- Backups of all the above
- ... and we have just limited hardware resources

Infrastructure



- Separation by rings based on importance and trust
- HA and backups for each ring

Synchronization approaches

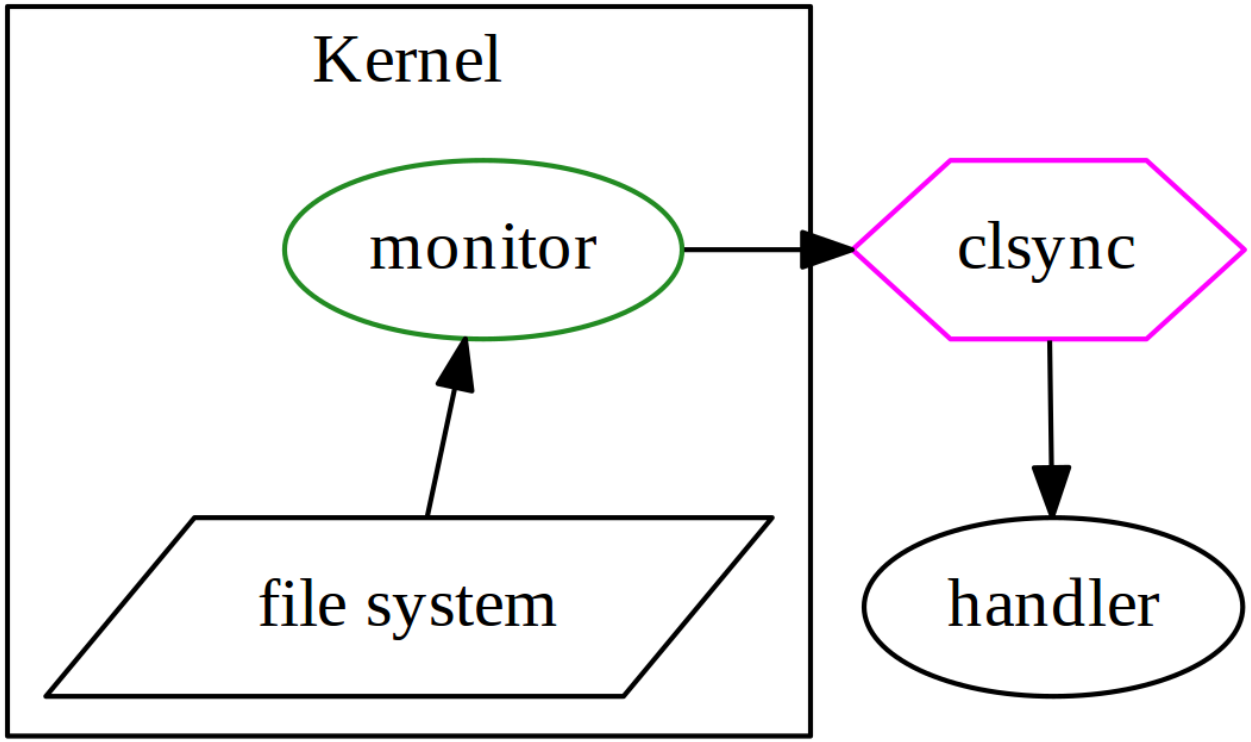
Possible solutions:

- RO file systems
  - limited application area
- block replication
  - unacceptable performance (DRBD + OCFS2)
  - not resistant to split brain
- network file systems (e.g. CEPH)
  - high latency (on 1 Gb/s)
  - kernel panics (year 2012)
- file level replication

Choosing software

- lsyncd: best match, but:
  - high CPU load ($>$ ½ code on LUA)
  - bugs and complexity of LUA code support
  - no threading, bad for large dirs ($> 10^6$ objects)
  - limited events aggregation
  - no *.so support
  - no BSD support
- incron
  - no recursion, no events
- csync2
  - no events, extremely slow
- librnotify
  - not existed at that time :)
  - bare inotify interface

How clsync works



Handler can be anything, but common case is rsync.

Choosing monitoring subsystem

Linux:

dnotify

- \+ can trace any event
- \- individual files can’t be monitored
- \- umount is blocked
- \- signal-based notification
- \- requires man stat(), only fd is provided

inotify [choice]

- \+ epoll-based notification
- \+ full event info provided
- \- no recursion
- \- watch $\Rightarrow$ path map required

Choosing monitoring subsystem

fanotify

+ recursion support
+ returns fd and pid *Rightarrow* path known
- no support for delete, move, rename events ;−(

FreeBSD

- kqueue/kevent
- libinotify (using kqueue)
- BSM API (not original purpose)
- dtrace (unusable, path can't be extracted)

There are no worthy Linux inotify replacements in FreeBSD. Best match is kqueue interface.

How clsync works

On default settings:

1. Initialization
2. Set inotify watches
3. Full file tree sync
4. Resync for new events
5. Wait, event aggregation and $\Uparrow$

Handlers

synchandler modes:

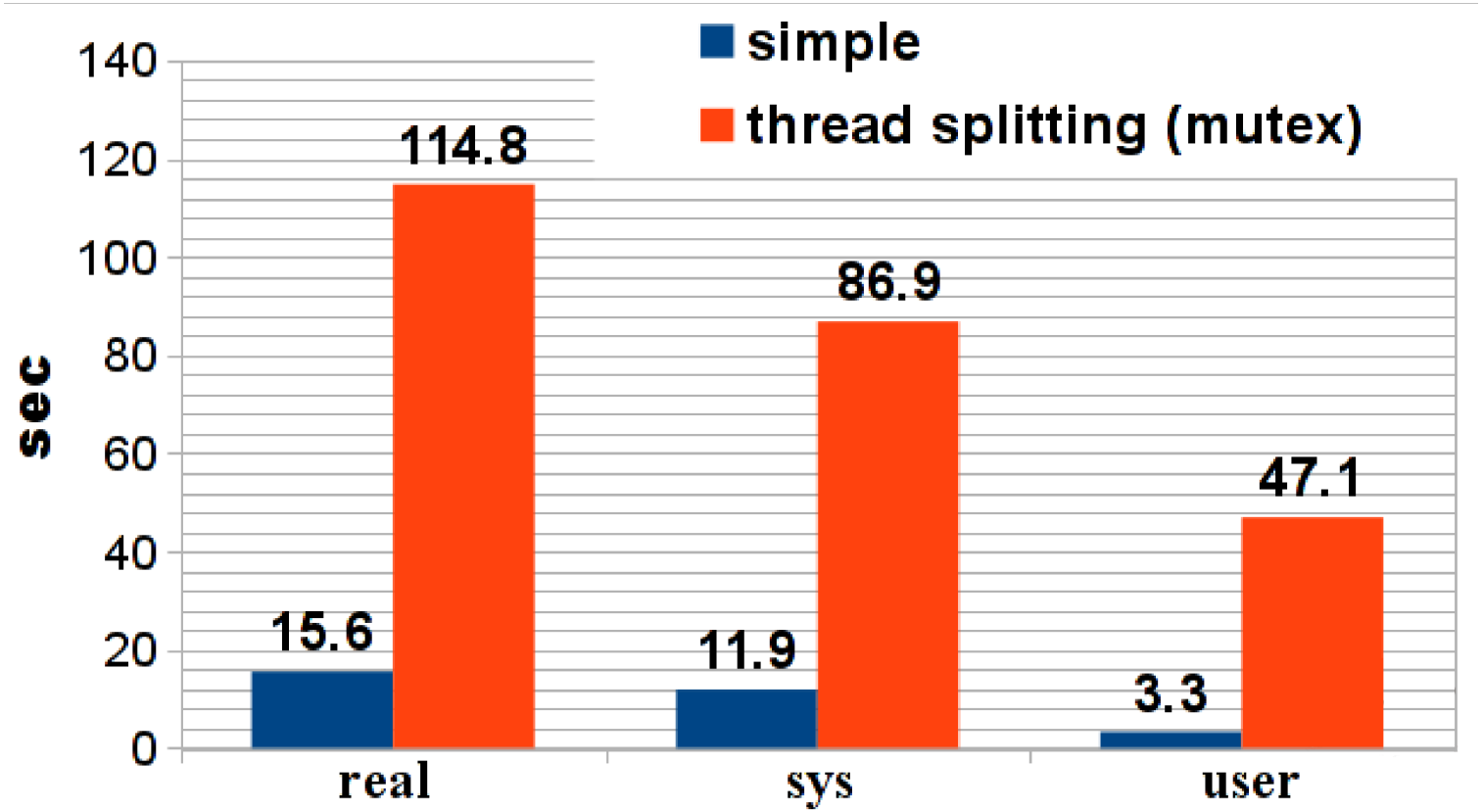
- *simple* — per event per file app call
- *direct* — per event app call
- *shell* — shell call on any sync event
- *rsyncdirect* — direct rsync call
- *rsyncshell* — call for rsync wrapper
- *rsyncso* — clsyncapi_rsync() callback with listing suitable for rsync
- *so* — load *.so with generic clsyncapi_sync() callback using simple file listing

Security

Security features optionally engaged:

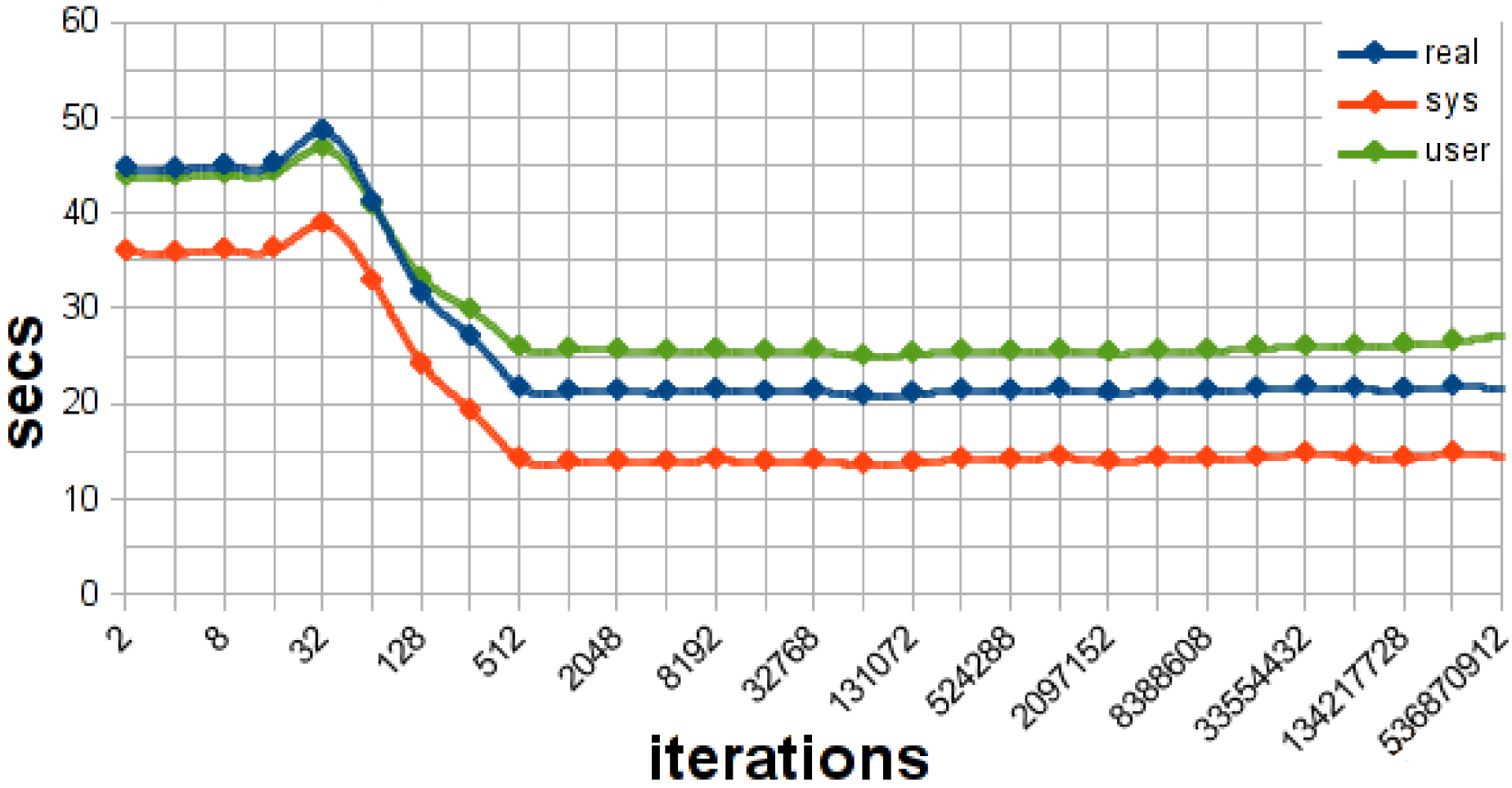
- privilege drop
- use of capabilities
- namespace isolation
- cgroups isolation
- thread splitting (priv + common)
- seccomp isolation
  - mprotect() banned $\Rightarrow$ no thread splitting
- process splitting (priv + common)

Threading performance



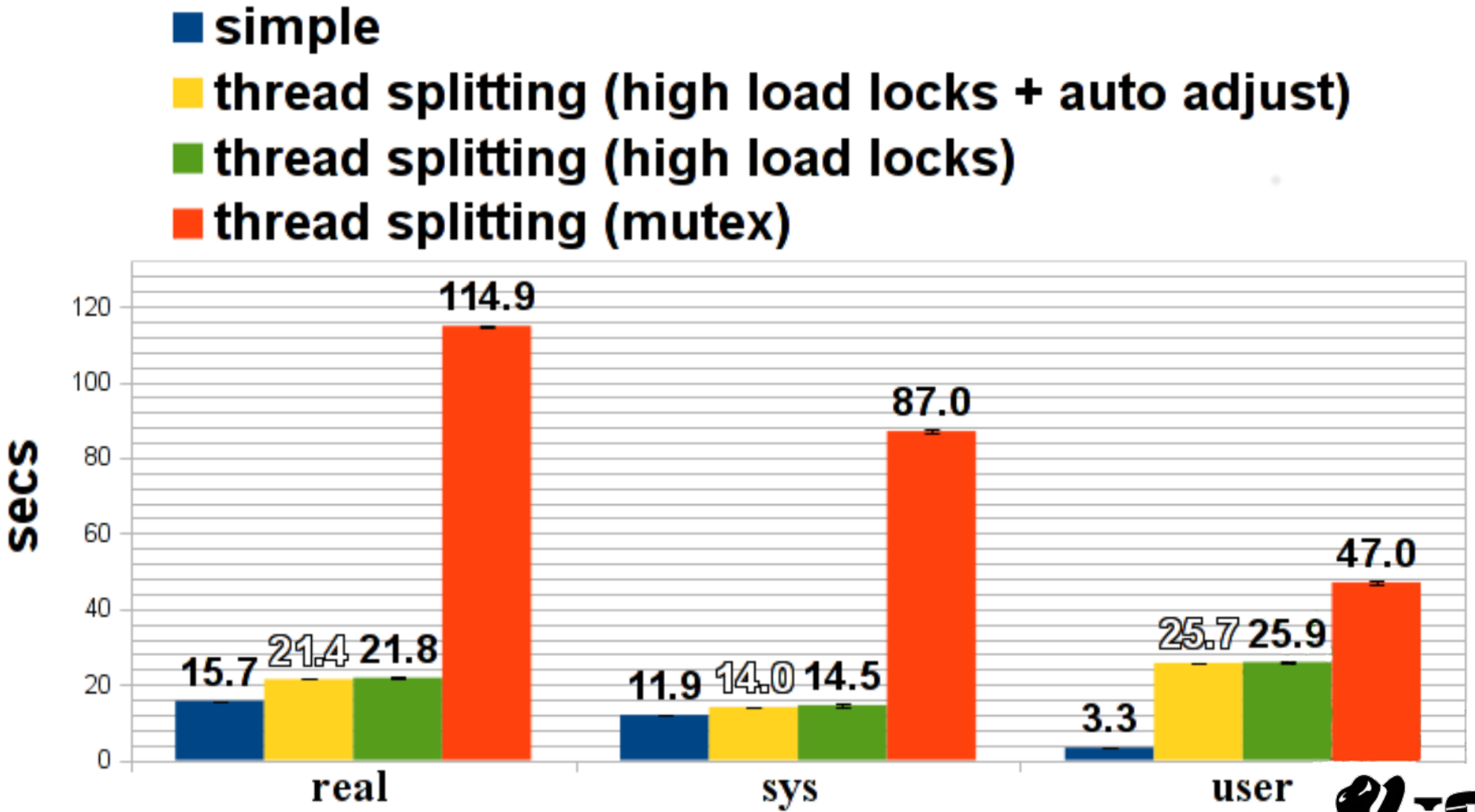
pthread mutex is not designed for high-speed locks

Spinlock to mutex switch



tested on init sync procedure

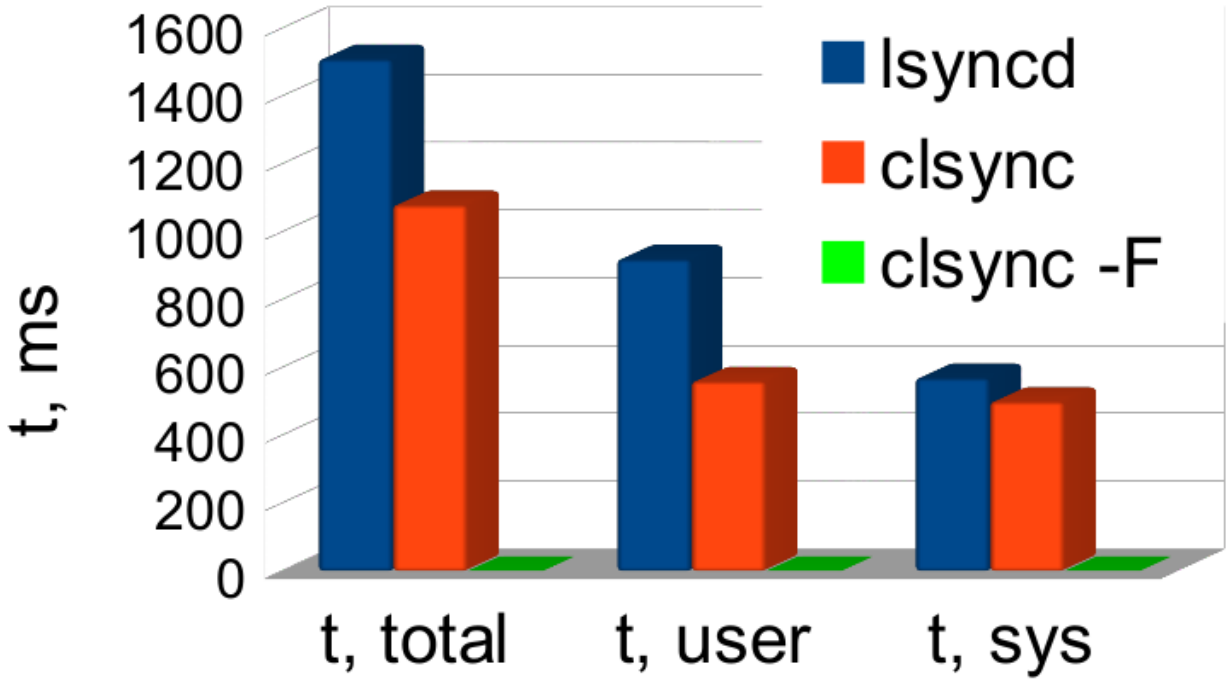
Smart locks


simple
thread splitting (high load locks + auto adjust)
thread splitting (high load locks)
thread splitting (mutex)
secs
120
100
80
60
40
20
0
15.7
21.4
21.8
114.9
11.9
14.0
14.5
87.0
3.3
25.7
25.9
47.0
real
sys
user

More features

- threading support (thread per sync event)
- regexp support and fs object type selection
- UNIX-socket control interface
- auto switch between spin_lock и mutex
- fast initial sync

Overhead measurement



- lsyncd v.2.1.x
- clsync v.0.4.x
- 4789558 files and dirs on tmpfs
- -F allows to bypass rules on initial sync

Usage

Mirror directory

```
clsync -Mrsyncdirect -W/path/to/source_dir \
-D/path/to/destination_dir
```

One-time sync

```
clsync --exit-on-no-events --max-iterations=20 \
--mode=rsyncdirect -W/var/www_new -Srsync -- \
%RSYNC-ARGS% /var/www_new/ /var/www/
```

Usage

Live correction of web site perms

```
clsync -w1 -t1 -T1 -x1 \
-W/var/www/site.example.org/root \
-Mdirect -Schown --uid 0 --gid 0 -Ysyslog -b1 \
--modification-signature uid,gid -- \
--from=root www-data:www-data %INCLUDE-LIST%
```

Troubleshooting

```
Cannot inotify_add_watch() on [...]:
No space left on device (errno: 28)
```

Increase sysctl **fs.inotify.max_user_watches**:

- one watch per dir
- no recursion
- kernel default is 8192

Usage

Live correction of web site perms

```
clsync -w1 -t1 -T1 -x1 \
-W/var/www/site.example.org/root \
-Mdirect -Schown --uid 0 --gid 0 -Ysyslog -b1 \
--modification-signature uid,gid -- \
--from=root www-data:www-data %INCLUDE-LIST%
```

Troubleshooting

```
Cannot inotify_add_watch() on [...]:
No space left on device (errno: 28)
```

Increase sysctl **fs.inotify.max_user_watches**:

- one watch per dir
- no recursion
- kernel default is 8192

Status

- Main functionality is implemented
- Maintenance mode
- Latest release $0.4.2$ a week ago $;-)$

Distro support:

Gentoo	official (w/o *-fbsd)
Debian	official (w/o fbsd)
RH-based	RPM available
FreeBSD	ports available

The only mandatory dep is glib.

Summary

Contacts:

Github	https://github.com/xaionaro/clsync
IRC	Freenode: #clsync
e-mail	dyokunev@ut.mephi.ru aasavchenko@ut.mephi.ru

Acknowledgements:

- Dmitry Yu. Okunev — main clsync author
- Artyom A. Anikeev and Barak A. Pearlmutter for packaging
- oldlaptop and Enrique Martinez for their help

Thank you for your attention!

Backup slides

Backup slides follow...

Limited resources

What we have:

- Limited hardware:
  - About 10 blades
  - 1 Gb/s interconnect (not very stable)
  - short on RAM and HDDs
- > 100 tasks (containers):
  - main university web site
  - departments sites
  - VoIP
  - VPN and Wi-Fi for students
  - internal services (e-mail, ntp, sks,...)

Possible approaches

What can be done?

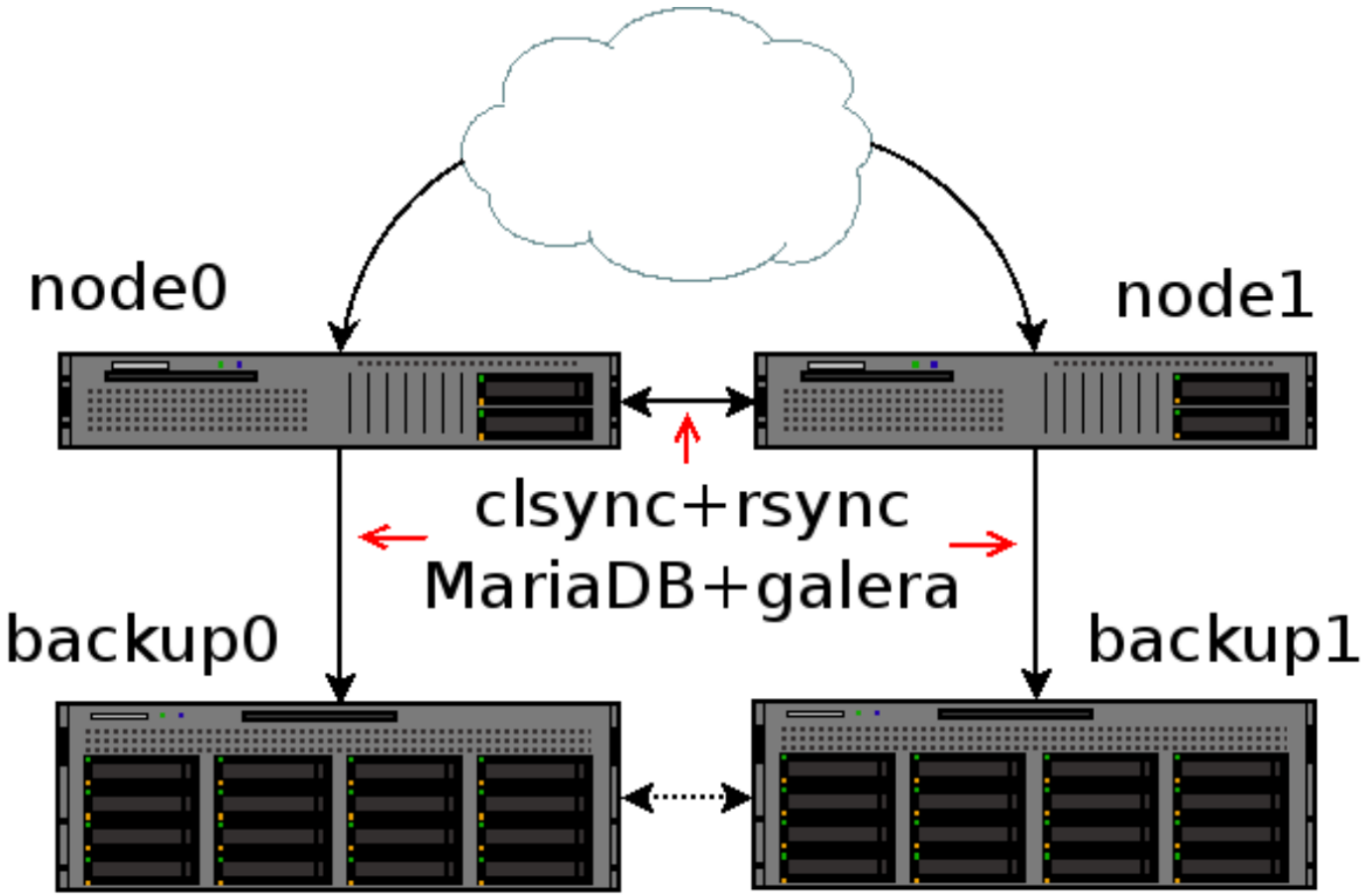
- Virtual machines $\Rightarrow$ containers (LXC)
- Disk data deduplication:
  - base image + aufs/overlayfs
- Cheap HA/LB clusters

File synchronization

Requirements:

- performance (minimal overhead)
- high availability (max few seconds downtime)
- reliability (failure minimization)
- versatility (wide range of use cases)
- fine tuning over aggregation

Infrastructure core



- LXC
- Ext4 c clsync + rsync
- Percona

Spinlock on single core