Linux can be fault-tolerant: Analysis on the Scope of Error Propagation

Takeshi Yoshimura, Hiroshi Yamada and Kenji Kono
Keio University

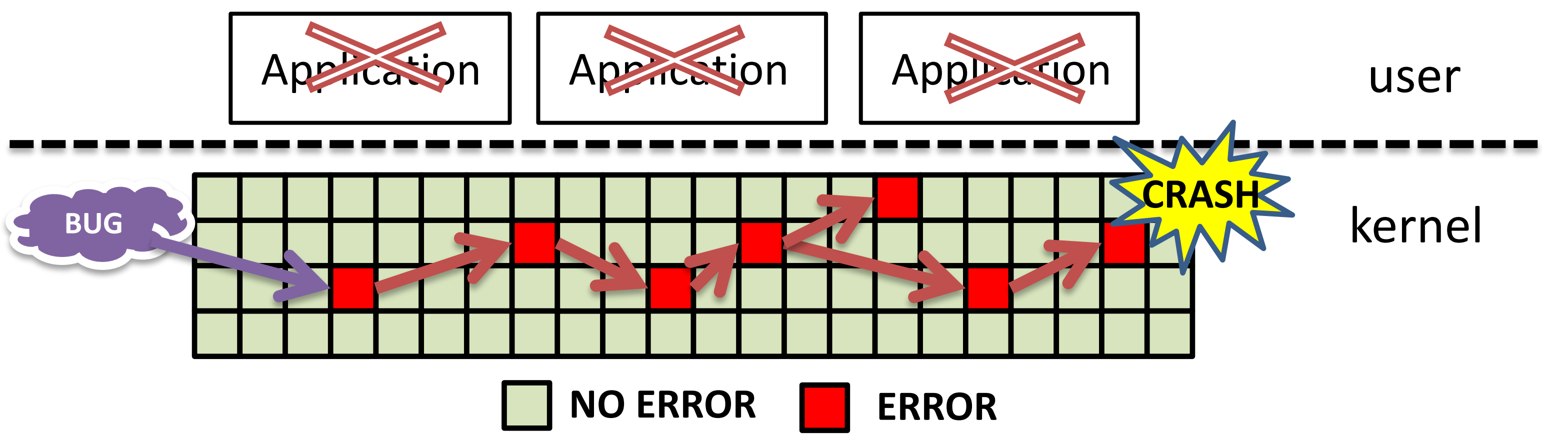
June 6th 2012

OS kernel crash

- Computer systems need to be highly available
  - Downtime costs $200,000 per hour for Amazon [Kembel ’00]
- OSes are crucial for achieving high availability of computer systems
  - A kernel crash can lead to the entire apps outage

Error propagation

- Kernel crashes derive from error propagation
- Error propagation is difficult to avoid
  - Difficult to remove all bugs in Linux kernels
    [Palix et al. ASPLOS'11] [Chou et al. SOSP'01]
- Propagated errors are difficult to fix
  - Need to inspect if each data is corrupted or not



Goal

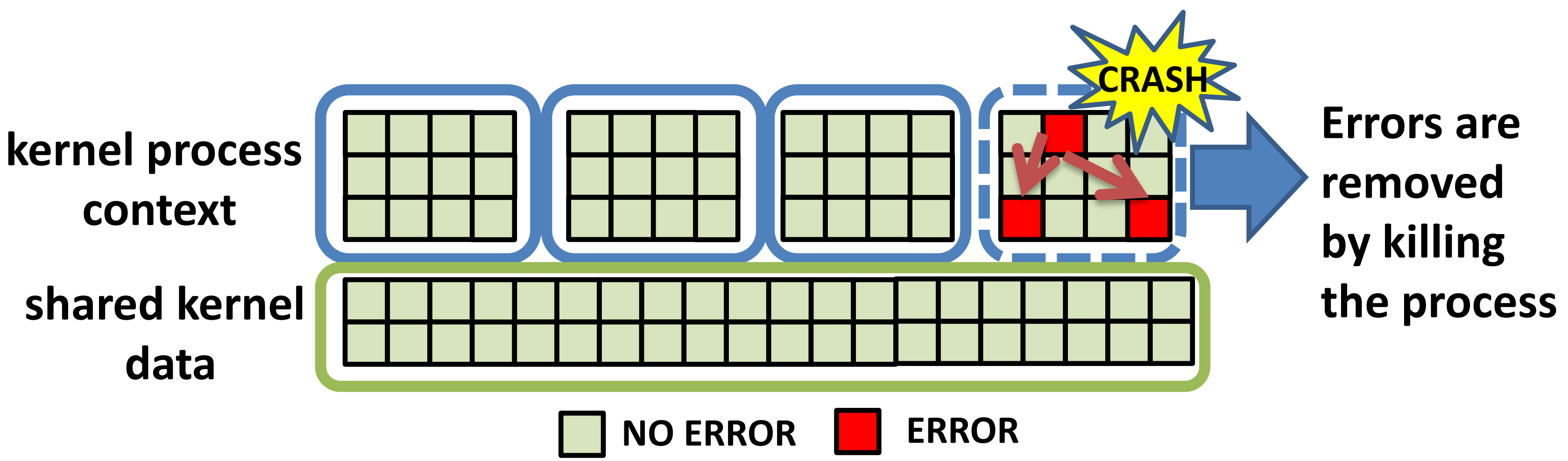
- Analyze error propagation in Linux 2.6.38
  - Corrupt data using fault injection
  - Crash the kernel and analyze the data corruption
- Explore the possibility of efficient crash recovery in Linux

The scope of error propagation

- Analyze the scope of error propagation
  - ***Process-local*** errors
    - Errors are confined in a kernel process context
  - ***Kernel-global*** errors
    - Errors propagate to data shared among the kernel
- If an error is process-local,
  - The system is expected to keep running correctly even after the kernel crashes

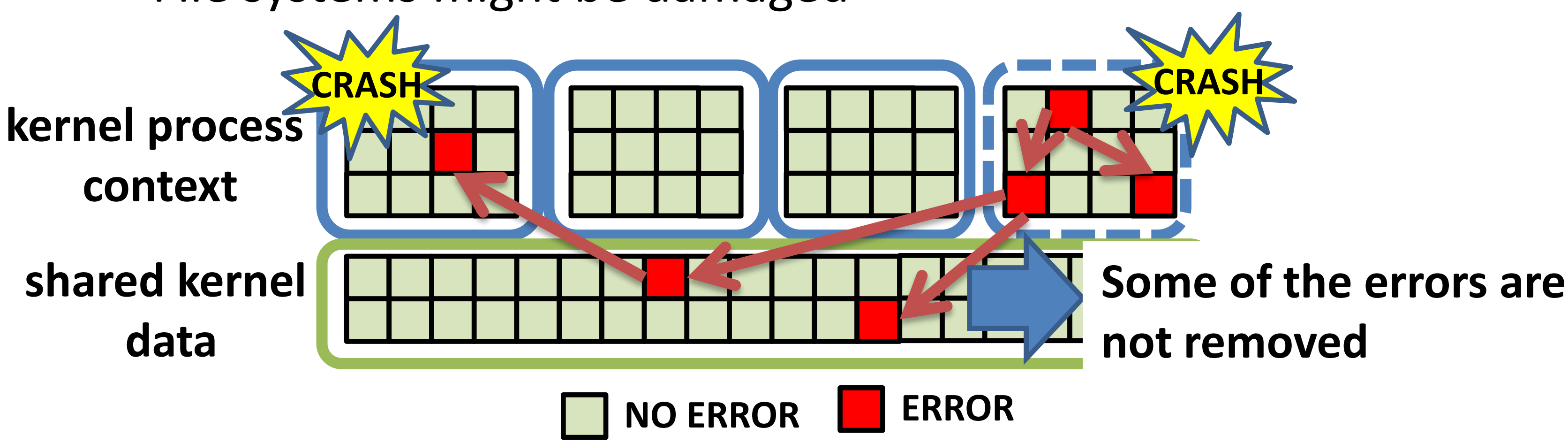
Process-local error

- Error propagation only within the kernel context of a process
  - e.g., data corruption in a kernel stack
- The other procs are expected to keep running
  - Killing a faulty proc removes all the corrupted data



Kernel-global error

- Error propagation in data shared among kernel contexts
  - e.g., data corruption in task_struct or mm_struct
- The other procs might behave incorrectly
  - Killing a faulty proc cannot remove all the corrupted data
  - The corrupted data can produce incorrect outputs
    - File systems might be damaged



Analyze the scope of error propagation

- Conduct 6738 experiments with Linux 2.6.38
  - Inject a fault in the kernel text segment
  - Run a workload in 6 benchmarks for each fault
    - UnixBench on {ext4, fat, USB}, Netperf, Aplay, Restartd
  - Investigate the scope if the kernel crashes
    - Investigate where memory is written with KDB

The fault injector

- Emulate 15 fault types by mutating an instr
  - Used for evaluation of previous researches in OS
  - Imitate bugs reported in Linux kernels
    - [Castro et al. SOSP ’09], [Palix et al. ASPLOS ’11], etc.

Examples of the Injected Fault

Fault types	before	after
init	int x = 1;	int x;
irq	arch_local_irq_restore()	deleted.
off by one	while (x < 10)	while (x <= 10)
bcopy	memcpy(ptr, ptr2, 256);	memcpy(ptr, ptr2, 512);
size	ptr = kmalloc(256, GFP_KERNEL);	ptr = kmalloc(128, GFP_KERNEL);
free	kfree(ptr);	deleted.
null	if (ptr == NULL) return;	deleted.

Result

- 134 kernel crashes are observed
  - 98/134 : process-local errors
  - 36/134 : kernel-global errors
    - Overrun, corrupt list_head or callback ptr, etc.
- Killing a faulty process removes all the corrupted data with 73% probability

	branch	inverse	ptr	dst src	init	irq	off by one	size	bcopy	loop	var	null	total
process-local	3	4	26	15	10	1	6	1	6	8	3	15	98
kernel-global	1	3	8	10	1	0	0	1	10	1	1	0	36
total	4	7	34	25	11	1	6	2	16	9	4	15	134

Experiment

- Examine if the system can survive kernel crashes by killing a faulty process
  - Crash the kernel
    - Use 134 kernel crashes in the scope analysis
    - A faulty process is killed by the kernel oops procedure
  - Run a workload in 6 benchmarks for each crash
    - In some cases we cannot run the workload
  - Examine the kernel reaction against the errors

Result (1/3): Kernel-global

- Examine 126 kernel reactions
  - 32/126: Workloads can keep running
    - Workloads use a subsystem unrelated to the error
  - 91/126: Workloads stop or do not start
    - Due to deadlock, oops again, a required process is killed and abort with errors detected
  - 3/126: Panic due to failure killing a faulty proc
    - Init and interrupt contexts cannot be killed

	not manifest any failures	deadlock	oops	no proc	detect error	panic	total
kernel-global	32	59	26	3	3	3	126

Result (2/3): Process-local

- Examine 463 kernel reactions
  - 314/463: Workloads can keep running
  - 142/463: Workloads stop or do not start
    - Deadlock, a required process is killed
  - 7/463: Panic due to failure killing a faulty proc

	not manifest any failures	deadlock	oops	no proc	detect error	panic	total
process-local	314	132	0	10	0	7	463

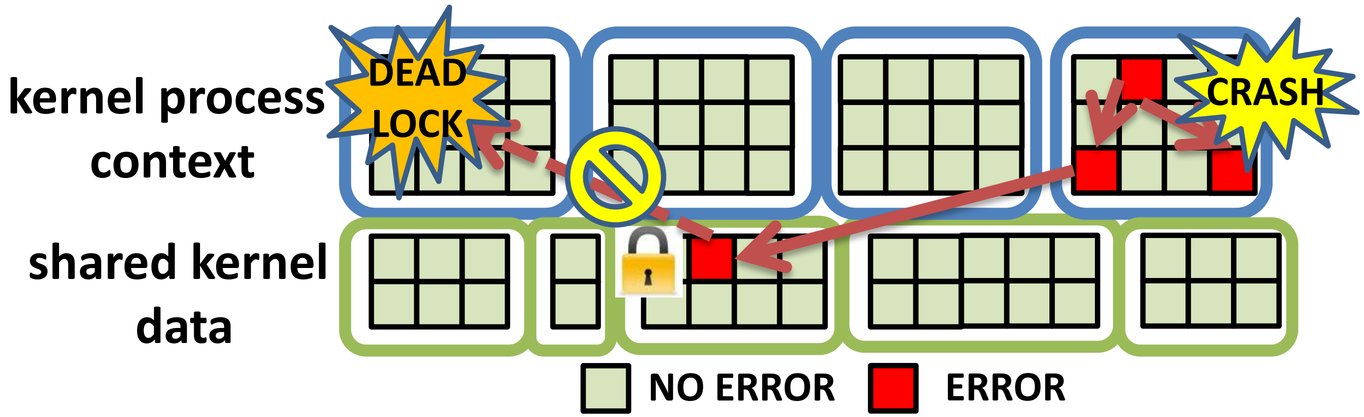
Result (3/3): Summary

- The system can survive the kernel crash by killing a process in 579/589 cases
  - A faulty proc cannot be killed in 10/589 cases
  - Incorrect kernel behavior is not observed
    - The kernel is expected to stop before reading the corrupted state, even if the errors are kernel-global

	not manifest any failures	deadlock	oops	no proc	detect error	panic	total
kernel-global	32	59	26	3	3	3	126
process-local	314	132	0	10	0	7	463
total	346	191	26	13	3	10	589

The kernel can fail-stop

- Kernel-global errors can be unreadable due to deadlock
  - The mutual execution is done to write shared data
  - A context killed in a critical section holds the lock
- Kernel-global errors soon cause kernel crashes
  - Corrupted list_head pointers soon cause invalid memory access



Related work

- A study of Linux behavior under errors [Gu et al. DSN '03]
  - Conduct fault injection experiments
  - Show error propagation among subsystems
- A study of bugs in Linux [Palix et al. ASPLOS '11]
  - Use a static analyzer to Linux kernels
  - Show the life-time and the distribution of bugs in Linux
- Reboot-based recovery with apps' state reserved [Depoutovitch et al. EuroSys '10]
  - Switch to the slave kernel when the master kernel crashes
  - Take downtime & need to re-design apps

Conclusion

- OS kernels need to be prevented from crashing
  - Error propagation makes crash recovery difficult
- We analyze the scope of erorr propagation in Linux 2.6.38
  - 98/134 errors are process-local
  - The kernel stops before reading kernel-global errors in 91/126 cases
- Our analysis indicates Linux can be fault-tolerant
  - Killing a faulty process is effective to survive kernel crashes