Techniques for Efficient In-Memory Checkpointing

9th Workshop on Hot Topics in Dependable Systems
November 3, 2013, Nemacolin Woodlands Resort (PA), USA

Dirk Vogt, Cristiano Giuffrida, Herbert Bos, Andrew S. Tanenbaum

VU University Amsterdam
void server_main()
{
    while (1) {
        event_t e;
        wait_for_event(&e);
        handle_event(&e);
    }
}
Motivation

```c
void server_main()
{
    while (1) {
        event_t e;
        wait_for_event(&e);
        take_checkpoint();
        handle_event(&e);
    }
}
```
In-Memory Checkpointing

Crash recovery
Debugging
Backtracking
Software Transactional Memory
Forensics
Outline

Techniques & Comparison

*COW-based checkpointing*

*Undo log*

*Shadow state approach*

Evaluation

Conclusion and Future work
Techniques
What Do We Seek?

Userspace techniques with ...

... low performance degradation
... small memory footprint
... protection of checkpoint data
Page Granular COW Checkpointing

Use MMU to protect pages

On fault copy the pages

fork() based implementation

fork() on checkpoint

mprotect() based implementation

Protect mapping and copy in signal handler
Page Granular COW Checkpointing

Performance degradation:
79.5 % on nginx

Memory overhead:
+4K for each page written to

Checkpoint protection:
Protected by MMU
Static Instrumentation

```c
int foo;
char bar[64];

void func()
{
    foo = 5;
    memset(bar, 0 , 64);
}
```
Static Instrumentation

```c
int  foo;
char bar[64];

void func()
{
    backup(&foo, sizeof(foo));
    foo = 5;
    backup(bar, 64);
    memset(bar, 0 , 64);
}
```
Undo Log

Instrument every store instruction
Save the previous content in log

Address, size and previous content
No duplicate detection for performance reasons
Undo Log

Performance degradation:

11.7 % on nginx

Memory overhead:

Unbounded
Depending on number of duplicate writes

Checkpoint integrity:

Checkpoint resides in application's address space
Additional check in backup function necessary
Shadow State

Division of address space in *primary* and *shadow* state

Check *tagmap* if primary location has to be saved to shadow state
Shadow State

4GB

Kernel

0x48 0x45 0x4c 0x4c
0x4f 0x48 0x4f 0x54
0xff 0xff 0xff 0xff

3GB

TagMap

0x48 0x45 0x4c 0x4c
0x4f 0x48 0x4f 0x54
0x53 0x50 0x4f 0x54

1.5GB

Guard Region

... 0 4 0 5 5 0 0 ...
... 5 1 4 0 0 4 5 ...

Current Epoch = 5
Shadow State

Current Epoch = 5

Guard Region

... 0 4 0 5 5 0 0 ...

... 5 1 4 0 0 4 5 ...

Kernel

TagMap

4GB

3GB

1.5GB

0x44
Shadow State

Current Epoch = 5

Kernel

TagMap

Guard Region

4GB

3GB

1.5GB

0x48 0x45 0x4c 0x4c
0x4f 0x48 0x4f 0x54
0x53 0x50 0x4f 0x54

0x48 0x45 0x4c 0x4c
0x4f 0x48 0x4f 0x54
0x44 0x50 0x4f 0x54

... 0 4 0 5 5 0 0 ...
... 5 1 4 0 0 4 5 ...
Shadow State

Kernel

TagMap

0x48 0x45 0x4c 0x4c
0x4f 0x48 0x4f 0x54
0x53 0x50 0x4f 0x54
0x48 0x45 0x4c 0x4c
0x4f 0x48 0x4f 0x54
0x44 0x45 0x50 0x00

Guard Region

... 0 4 0 5 5 0 0 ...
... 5 1 4 0 0 4 5 ...

Current Epoch = 5
Shadow State

Performance degradation:

9.5 % on nginx

Memory overhead:

Virtual: 1.5 GB + tagmap size
Physical: Fork + tagmap pages

Checkpoint Protection:

 Writes into shadow state/bitmap are caught by guard pages
Evaluation
Evaluation

Macro benchmarks

*Server applications:*

- nginx, lighttpd, Apache httpd

*Scientific computation: hmmr*

Micro benchmarks

*Influence of duplicate writes*
Web Server Throughput

Throughput Degradation in %

httpd
- 64K: 7.4 shadow state, 10.4 undo log, 66.7 fork
- 4K: 11.6 shadow state, 14.2 undo log, 75.2 fork

lighttpd
- 64K: 0.2 shadow state, 0.9 undo log, 81.4 fork
- 4K: 14.0 shadow state, 16.6 undo log, 87.2 fork

nginx
- 64K: 1.0 shadow state, 2.2 undo log, 76.0 fork
- 4K: 9.5 shadow state, 11.7 undo log, 79.5 fork
Size of undo log
~1.5 GB
Effect of Write Redundancy

Normalized execution time

Redundancy Factor

Undo log
Shadow state
Discussion
Discussion

Use case determines the right technique:

*Checkpoint interval?*

*Number of duplicate writes?*

Instrumentation based techniques well suited for short checkpoint intervals
What next?

Kernel based techniques?
Varying checkpoint intervals:
   *Function level? Basic block level?*
Extend to on disk checkpoints
What if recovery time plays a role?
Thank you!

Questions?

Now …

… or d.vogt@vu.nl