Lightweight Recoverable Virtual Memory (*LRVM*)

M. Satyanarayanan et al.

Presented by Soonhyeon Kwon

Introduction

- The term **Recoverable**
  - *Transactional guarantees:* database systems, banking, etc.

- Principles for transactions: **ACID**
  - **A**tomicity, **C**onsistency, **I**solation, **D**urability

Motivation

- Coda file system
  - Research project by M. Satyanarayan
  - Originally developed for distributed file system
  - Require each server to ensure the atomicity
  - Main reason why they wanted *recoverable virtual memory*

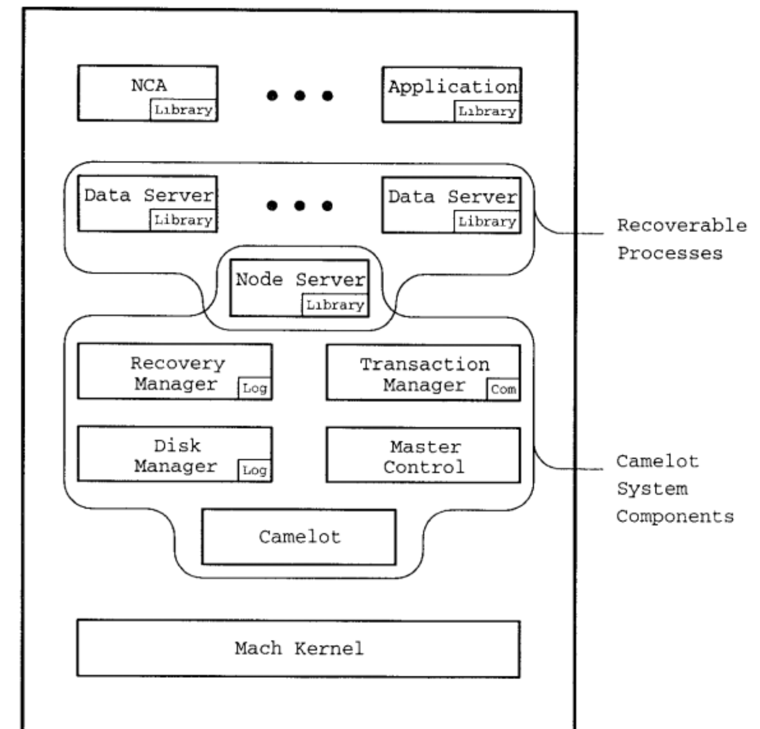
Motivation

- Notable solution before: ***Camelot***
  - Transactional facility for distributed systems
  - Mach kernel

Motivation

- Structure of *Camelot*

- Heavily relying on… (of Mach)
  - External page management
  - Inter-process communication



Motivation

- But *Camelot* is **too heavy**
  - Poor scalability
    - increasing CPU utilization from paging and context switching
  - Programming constraints
    - structural flaw, e.g. mandatory Disk Manager usage in processor
  - Difficulty of maintenance
    - thousands of lines
    - where are bugs from; Camelot? Mach?

- Originally *Camelot* is not designed for RVM

Design

- **Simplicity** over *generality*

- Eliminating support for ***nesting*** and ***distribution***
  - First simplification
  - Each could be better provided as an independent layer on top

Design

- Factoring out **concurrency control**
  - Rather than having RVM insist on a specific technique
  - Allowing applications to use a policy of their choice

Design

- Support for multithread
  - Parallelism
  - Not depending on kernel thread support
  - But they actually used Mach kernel threads

Design

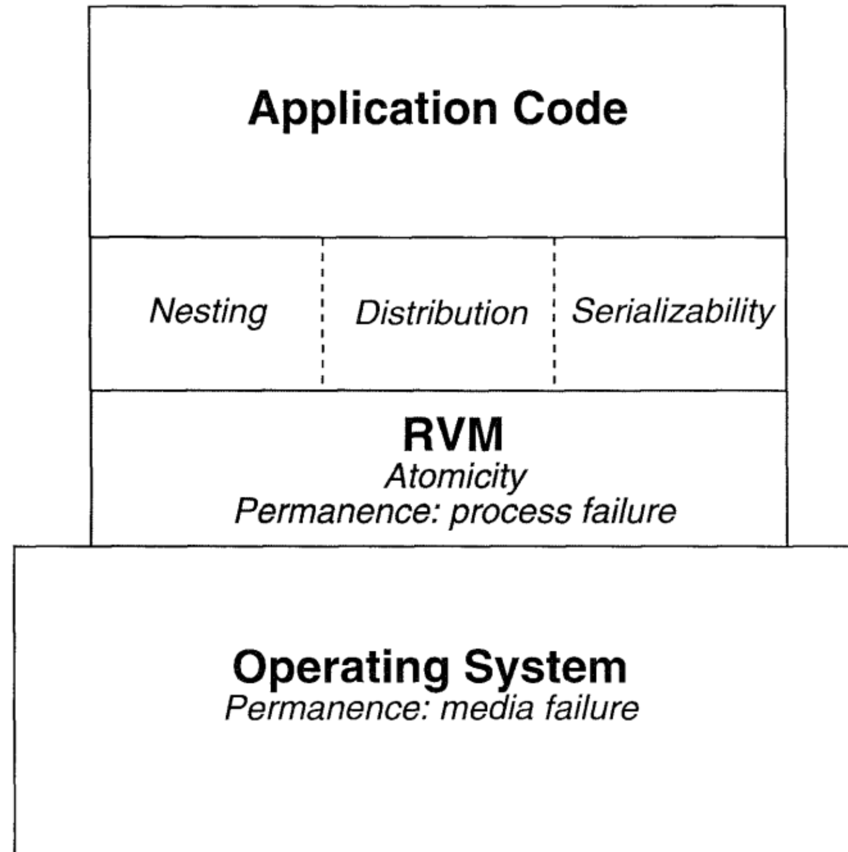


Fig. 2. Layering of functionality in RVM.

Design

- Making portable RVM
  - Relying on a small Unix subset of the Mach system call interface
  - No counting on tight coupling between RVM and the VM
  - No replacement of file systems or databases

Design

- External data segment
  - Backing store for a recoverable region
  - Completely independent of the region's VM swap space
  - Crash recovery relies only on the state of this

> *" It was acceptable to waste some disk space by duplicating the backing store for recoverable regions „*

- ***Complexity*** versus ***resource usage*** tradeoff
  - Keeping RVM simple and portable
    ... by being generous with memory and disk space
  - ***Startup latency***: need to read massive bunch first time

Design

- Insulating RVM from the VM subsystem
  - Hindering the sharing of RVM across address spaces
  - But this is not a serious limitation: Increasing robustness
  - Sharing w/ volatile VM can cause **persistent damage**

Design

- Problem of *Camlot*'s modular decomposition
  - It is predicated on fast IPC
  - But commercial Unix implementation lags far behind that
  - IPC is 600 times more expensive than local procedure call (Stout et al., 1991)

Design

- Structured RVM as a library
  - … that is linked in with an application
  - No external communication
  - Implication

> *“ We have to* ***trust applications*** *not to damage RVM structures and vice versa. „*

Design

- Problem: *inability to share one log*
  - A single write-ahead log on a dedicated disk
  - Not a significant limitation for Coda, though

- Two potential alleviating factors
  - Transaction-processing considerations
    - Placing the RVM log for each application in a separate file
    - No log multiplexor would be needed (file system's role)
  - Trend toward using small form factor disks
    - It will be achieved by using many small disks
    - Less economic incentive for avoiding a dedicated disk per process

Design

- Each process using RVM has a separate log
  - Can be placed in a Unix file, or on a raw disk partition

Architecture

- *Segments*
  - Loosely analogous to Multics *segments*
  - 2^{64} bytes long (hardware constraint in that time: 2^{32})
  - No limitation of the number of *segments*
  - Application explicitly map regions of *segments*

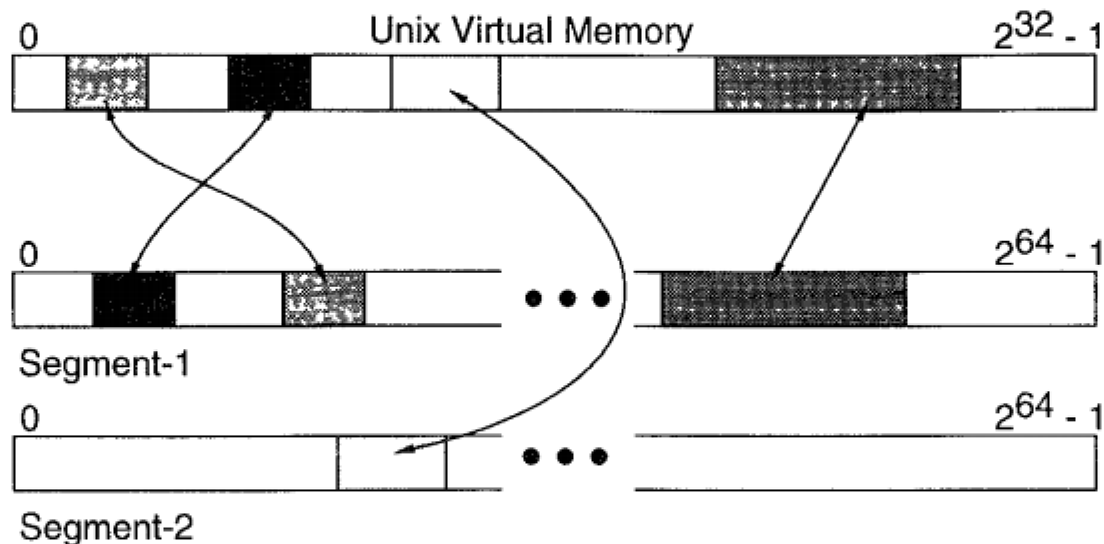


Fig. 3. Each shaded area mapping regions of segments specified during mapping.

Architecture

- Limitation of *segments*
  - Startup latency
  - No regions of a segment may mapped more than once by the same process
  - Mappings cannot overlap in virtual memory
  - Mapping must be done in multiples of page size
  - Regions must be page aligned

- Unmapping regions
  - Can be done at any time, as long as they have no outstanding uncommitted transactions
  - After unmapping, RVM retains no more information

Architecture

- Managements of *segments*
  - *Segment loader*: built on top of RVM
    - Allowing the creation and maintenance of a load map
    - Taking care of mapping a *segment* into the same base address
    - By this, use of absolute pointers in segments can be simplified
  - *Recoverable memory allocator*: layered on RVM
    - Supporting heap management of storage with in a *segment*

Architecture

- RVM primitives

```
initialize(version, options_desc);
map(region_desc, options_desc);
unmap(region_desc);
terminate();
```

(a) Initialization & Mapping Operations

```
begin_transaction(tid, restore_mode);
set_range(tid, base_addr, nbytes);
end_transaction(tid, commit_mode);
abort_transaction(tid);
```

(b) Transactional Operations

```
flush();
truncate();
```

(c) Log Control Operations

```
query(options_desc, region_desc);
set_options(options_desc);
create_log(options, log_len, mode);
```

(d) Miscellaneous Operations

Fig. 4 RVM primitives.

Implementation

- No-undo/redo value-logging strategy (Bernstein et al., 1987)
  - No reflection uncommitted changes to a segment
  - Implying adequate buffer space is available in VM
    - Recording old-value of uncommitted transactions

- Crash recovery
  - Idempotency by delaying update log status block

Implementation

- Log truncation
  - Reclaiming space
  - Epoch truncation: same code as for recovery
    - Flaws: increasing log traffic, bursty system performance
    - Alternative solution: incremental truncation (not ready)

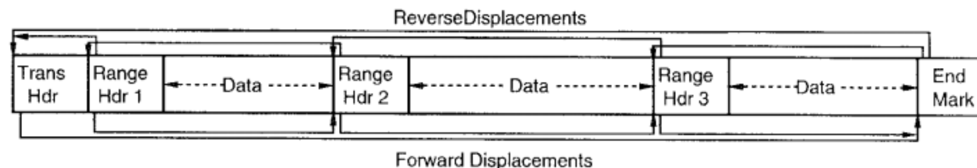


Fig. 5. This log record format of a typical log record read either way.

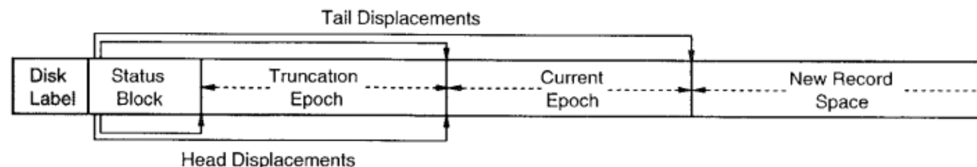


Fig. 6. This figure shows epoch truncation freed for new log records.

Expriences

- Very usual mistakes: forgot *set_range*
  - Disastrous result: cannot create a new-value record
  - …, which means modification cannot be restored and reflected

Optimization

- Intra-transaction
  - Ignoring duplicated *set_range* calls in single transaction
  - Clustering overlapping and adjacent log records

- Inter-transaction
  - Only for *no-flush* transactions
  - Discarding older log records

Evaluation

- Complexity
  - Based on lines of C code
  - RVM: mainline (10K lines) + auxiliary code (10K lines)
  - Camelot: mainline (60K lines) + auxiliary code (10K lines)
    - Does not including Mach features such as IPC or external pager

Evaluation

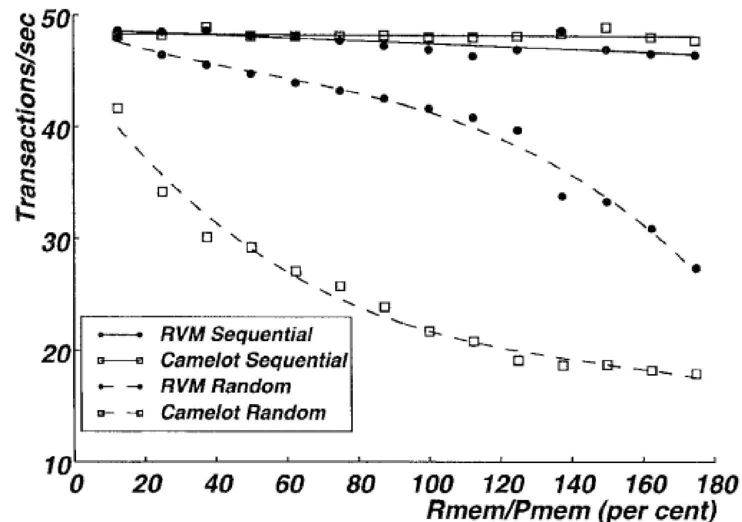
- Performance: throughput

Table I. This Table Presents Transactional Throughput Significantly

No. of Accounts	Rmem / Pmem	RVM (Trans/Sec)			Camelot (Trans/Sec)		
		Sequential	Random	Localized	Sequential	Random	Localized
32768	12.5%	48.6 (0 0)	47.9 (0 0)	47.5 (0 0)	48.1 (0.0)	41.6 (0 4)	44.5 (0 2)
65536	25.0%	48.5 (0 2)	46.4 (0 1)	46.6 (0.0)	48.2 (0.0)	34.2 (0.3)	43.1 (0 6)
98304	37.5%	48.6 (0 0)	45.5 (0.0)	46.2 (0 0)	48.9 (0 1)	30.1 (0 2)	41.2 (0 2)
131072	50.0%	48.2 (0.0)	44.7 (0.2)	45.1 (0 0)	48.1 (0 0)	29.2 (0.0)	41.3 (0 1)
163840	62.5%	48.1 (0 0)	43.9 (0 0)	44.2 (0.1)	48.1 (0.0)	27.1 (0.2)	40.3 (0 2)
196608	75.0%	47.7 (0 0)	43.2 (0 0)	43.4 (0 0)	48.1 (0 4)	25.8 (1 2)	39.5 (0 8)
229376	87.5%	47.2 (0.1)	42.5 (0.0)	43.8 (0 1)	48.2 (0.2)	23.9 (0 1)	37.9 (0 2)
262144	100.0%	46.9 (0 0)	41.6 (0.0)	41.1 (0.0)	48.0 (0 0)	21.7 (0 0)	35.9 (0 2)
294912	112.5%	46.3 (0 6)	40.8 (0 5)	39.0 (0 6)	48.0 (0 0)	20.8 (0 2)	35.2 (0 1)
327680	125.0%	46.9 (0 7)	39.7 (0 0)	39.0 (0.5)	48.1 (0.1)	19.1 (0.0)	33.7 (0.0)
360448	137.5%	48.6 (0 0)	33 8 (0 9)	40.0 (0 0)	48.3 (0 0)	18.6 (0 0)	33.3 (0 1)
393216	150.0%	46.9 (0 2)	33.3 (1 4)	39.4 (0 4)	48.9 (0 0)	18.7 (0 1)	32.4 (0 2)
425984	162.5%	46.5 (0 4)	30.9 (0 3)	38.7 (0 2)	48.0 (0 0)	18.2 (0 0)	32.3 (0 2)
458752	175.0%	46 4 (0 4)	27.4 (0 2)	35.4 (1 0)	47.7 (0 0)	17.9 (0 1)	31.6 (0 0)

Evaluation

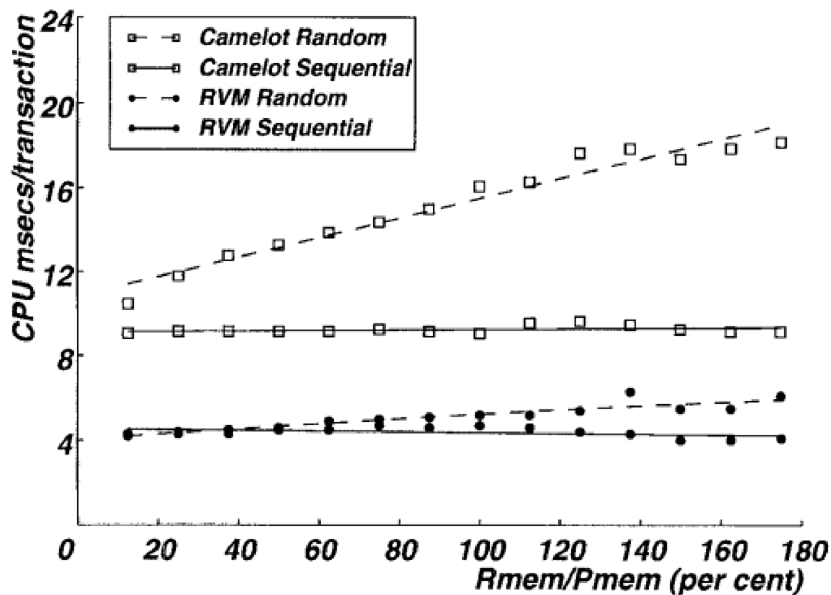
- Throughput
  - Significantly better in all case of random set (worst case)
  - Still better in all case of localized set (average case)
  - Performance drops in case of sequential set (best case)
    - Affected by memory size: from RVM's memory size became 75% of physical memory size
    - *Startup latency*



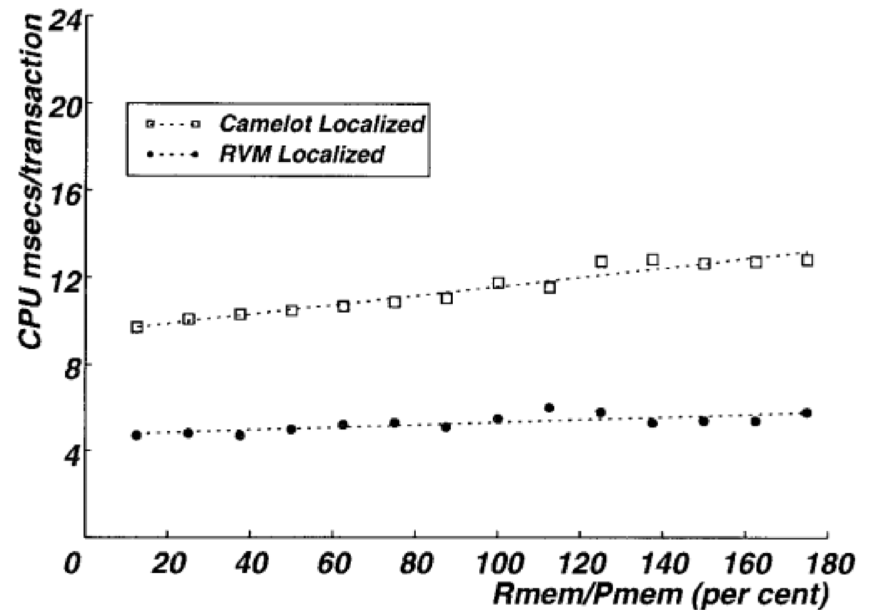
(a) Worst and Best Cases

Evaluation

- Scalability: CPU cost per transaction
  - *Camelot*'s poor scalability: paging, context changing, ….
  - RVM has way more better scalability than *Camelot*



(a) Worst and Best Cases



(b) Average Case

Evaluation

- Effect of optimizations
  - 20.7–81.6% savings observed
    - Intra-transaction: 17.3–41.6%
    - Inter-transaction: *20.9–64.3%* (*when occurs*)

Table II. This Table Presents Savings Due to RVM Optimizations, Clients, and Servers

Machine name	Machine type	Transactions committed	Bytes Written to Log	Intra-Transaction Savings	Inter-Transaction Savings	Total Savings
grieg	server	267,224	289,215,032	20 7%	0 0%	20 7%
haydn	server	483,978	661,612,324	21 5%	0 0%	21.5%
wagner	server	248,169	264,557,372	20 9%	0 0%	20.9%
mozart	client	34,744	9,039,008	41.6%	26.7%	68 3%
ives	client	21,013	6,842,648	31.2%	22 0%	53 2%
verdi	client	21,907	5,789,696	28 1%	20 9%	49 0%
bach	client	26,209	10,787,736	25 8%	21 9%	47 7%
purcell	client	76,491	12,247,508	41.3%	36 2%	77 5%
berlioz	client	101,168	14,918,736	17.3%	64.3%	81 6%

Summary

- Design principle
  - ***Simplicity*** *over* ***generality***
  - ***More resource usage*** *rather than* ***complexity***
  - Library, layered structure

- Weak point
  - Trusting applications: programmers ***can mistake***
  - Lack of description about additional layers

Thank you!