A Fast Restart Mechanism for Checkpointing in Networked Environments

Yawei Li and Zhiling Lan

Department of Computer Science
Illinois Institute of Technology

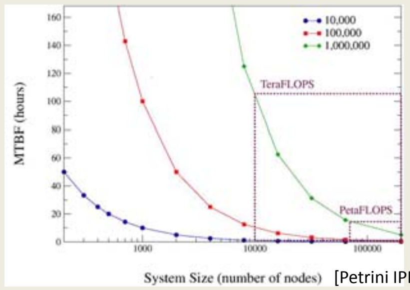
Email: {liyawei,lan}@iit.edu

Outline

- Motivation
- FREM: a Fast REstart Mechanism
  - Design & implementation
- Experiments
- Summary

Reliability Crisis

- Failures are inevitable in large-scale systems
  - MTBF drops dramatically in large systems



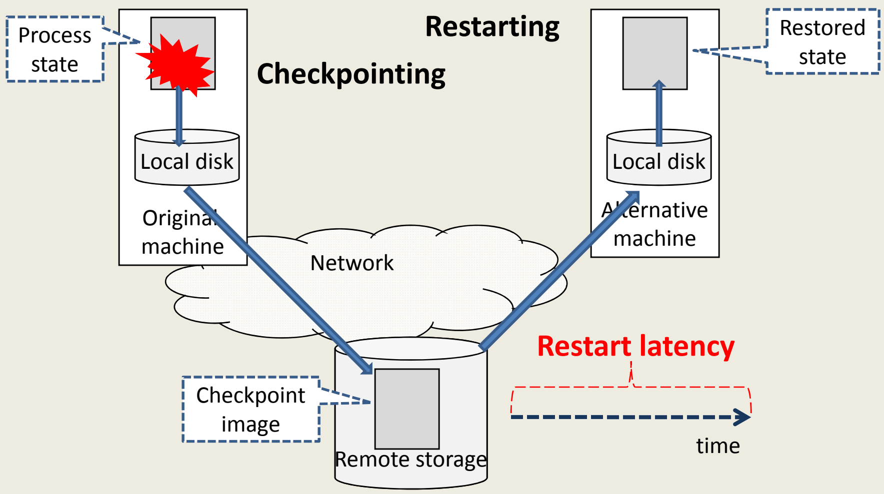
[Petrini IPDPS'04]

- Long-running programs are forced to restart more frequently

Checkpoint/Recovery

- C/R saves program state onto stable storage
  - Reduce rollback cost after restart
  - In networked systems, enable remote process restart on alternative resource
- C/R is not a silver bullet!
  - Extensive studies on reducing checkpointing overhead & determining an optimal frequency
  - Little work on optimizing its restart

An Illustrative Example

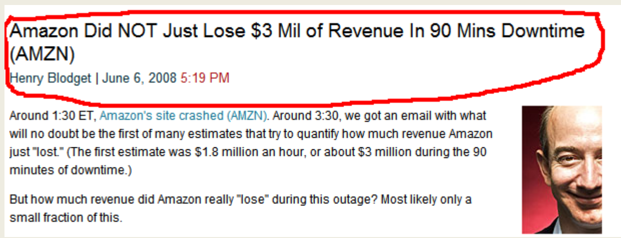


Problem Statement

- Currently, a restart requires the entire checkpoint image before it can proceed
  - Substantial restart latency in networked environments
  - Network transmission and I/O operation time
- Insatiable data demand from applications leads to larger checkpoint size
  - Thus, longer restart latency

The Pain of Restart Latency

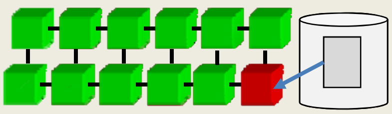
- Restart latency directly contributes to program downtime
  - Avail = (up time)/(up time + down time)


Amazon Did NOT Just Lose $3 Mil of Revenue In 90 Mins Downtime (AMZN)

Henry Blodget | June 6, 2008 5:19 PM

Around 1:30 ET, Amazon's site crashed (AMZN). Around 3:30, we got an email with what will no doubt be the first of many estimates that try to quantify how much revenue Amazon just "lost." (The first estimate was $1.8 million an hour, or about $3 million during the 90 minutes of downtime.)

But how much revenue did Amazon really "lose" during this outage? Most likely only a small fraction of this.



Parallel jobs make it worse:
all processes stall during restart!

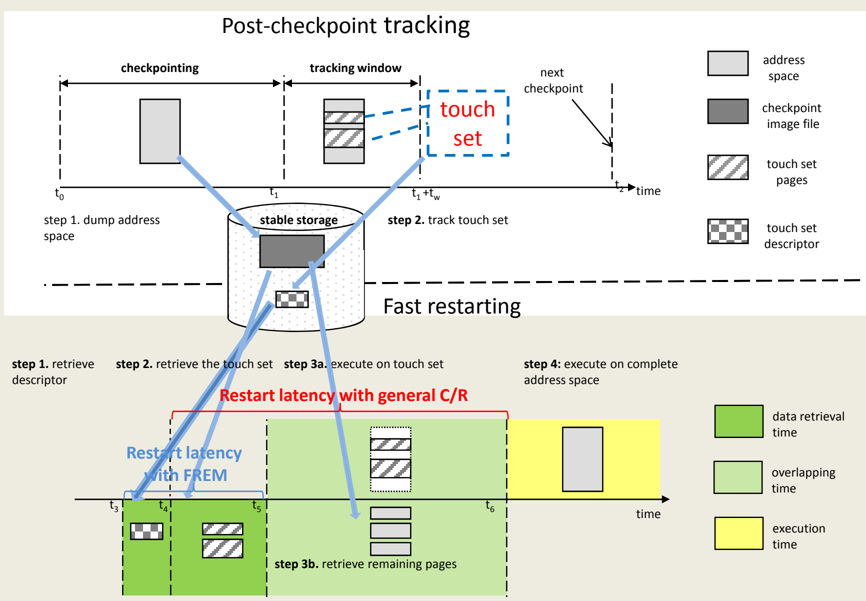
FREM: A Fast REstart Mechanism

- Provide a quick restart through *latency hiding*
  - Overlapping process execution with checkpoint image retrieval
- Based on two key observations
  - Data locality: only a small portion of the checkpoint image is immediately needed after resurrection
  - Predictable pattern: data access patterns of restart from checkpoint can be known

Main Idea

1) Post-checkpoint tracking
   - Record the *touch set* following each CKP within the *tracking window*
     - Touch set: defined as the intersection of the process address space saved in the checkpoint and its working set
     - It shows which data is immediately needed for restart from the image file
2) Fast restarting
   - Restart the process as soon as the touch set is retrieved
   - Overlap the process execution with data retrieval of the remaining image

ILLINOIS INSTITUTE OF TECHNOLOGY Zhiling Lan 9



ILLINOIS INSTITUTE OF TECHNOLOGY Zhiling Lan 10

Technical Challenges

1) How to accurately identify the touch set?
2) How to appropriately set the tracking window?
3) How to effectively load the partial image?

Technical Challenges

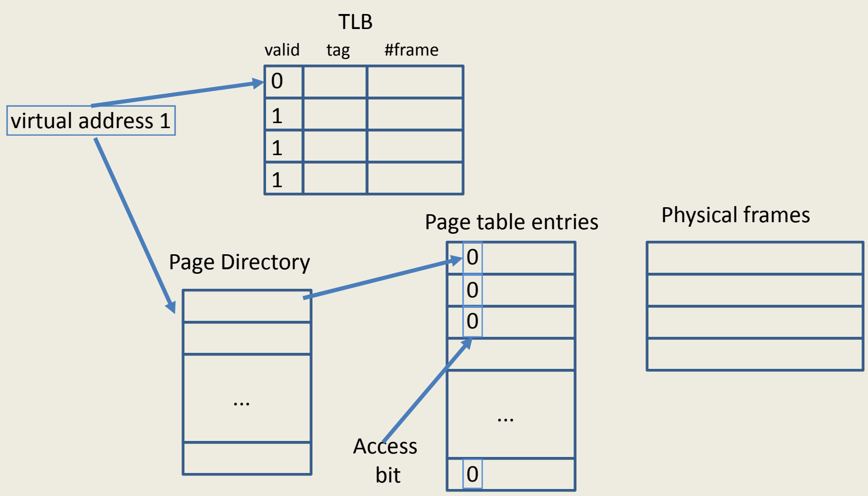
1) How to accurately identify the touch set?
2) How to appropriately set the tracking window?
3) How to effectively load the partial image?

Identifying Touch Set

- Find out the data that is immediately needed for restart from the image file
- Two types of errors
  - False negative – a page of interest is missed
  - False positive – a page not interested is included
- Sources of inaccuracy:
  - Hardware bypassing, page swapping, dynamic memory usage

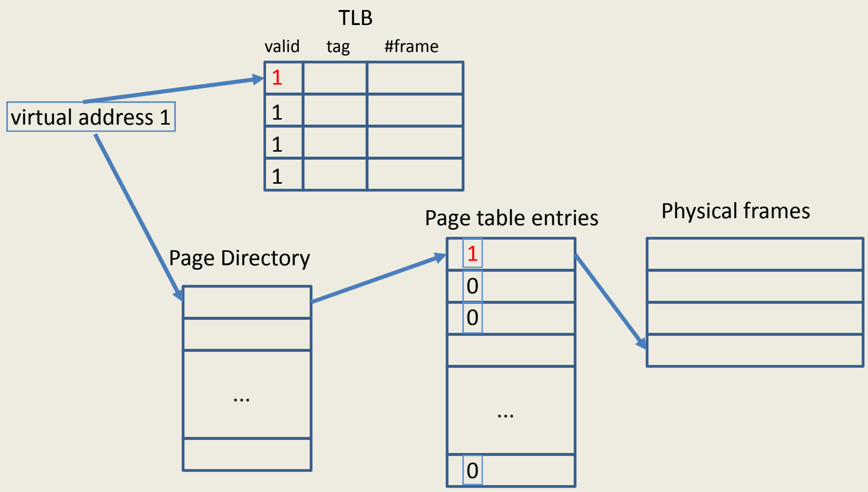
Hardware Bypassing – TLB

- TLB Miss: OK



Hardware Bypassing – TLB

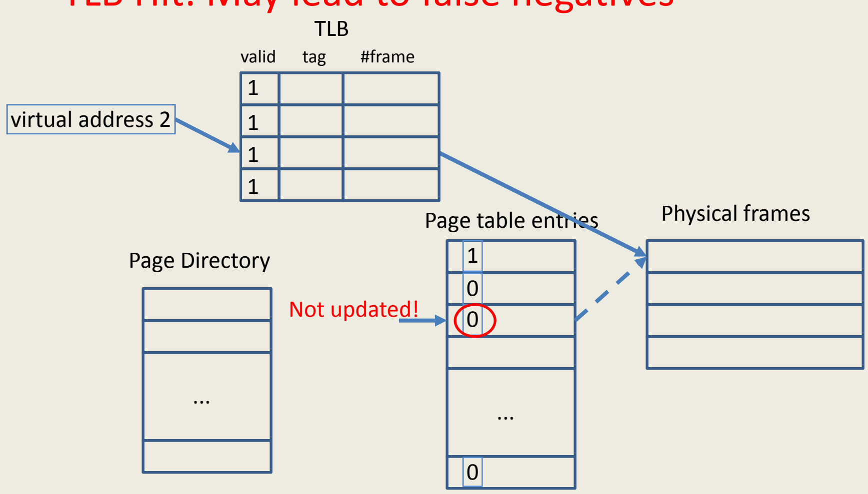
- TLB Miss: OK



ILLINOIS INSTITUTE OF TECHNOLOGY Zhiling Lan 15

Hardware Bypassing – TLB

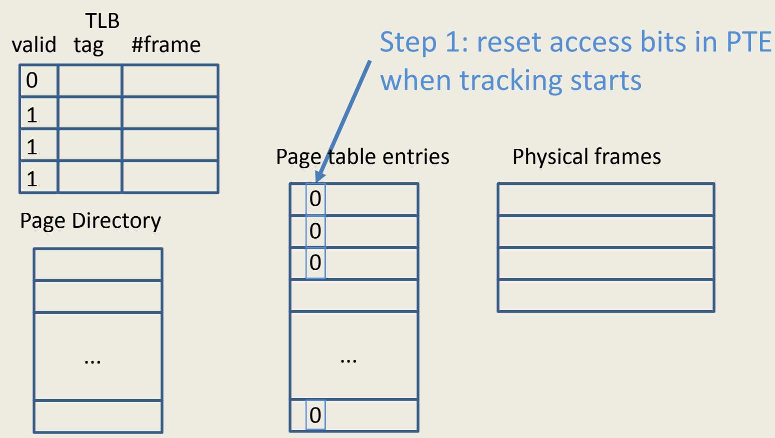
- TLB Hit: May lead to false negatives



ILLINOIS INSTITUTE OF TECHNOLOGY Zhiling Lan 16

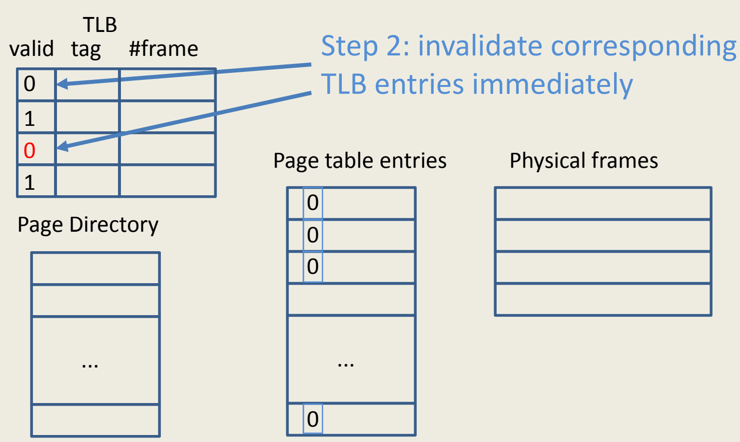
Hardware Bypassing – TLB

- Solution: invalidate TLB entry to enforce TLB miss



Hardware Bypassing – TLB

- Solution: invalidate TLB entry to enforce TLB miss



Hardware Bypassing – DMA

- Hardware devices access system memory for reading/writing independently of the CPU
  - Direct I/O, zero copy optimization

- Solution: include the mapped DMA pages in the touch set

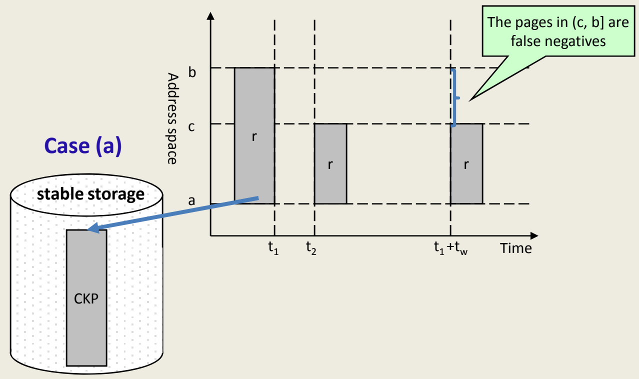
Page Swapping

- OS page swapping may cause false negatives
  - Page swapping algorithm clears the access bits when do the page table walk

- Solution: intercept *kswapd* to record the page per swapping

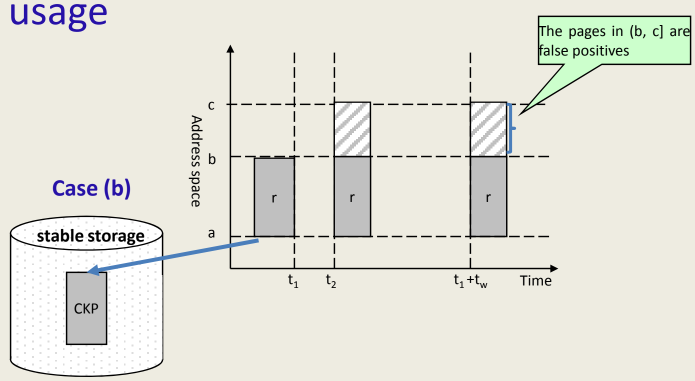
Dynamic Memory Usage

- Complications incurred by dynamic memory usage



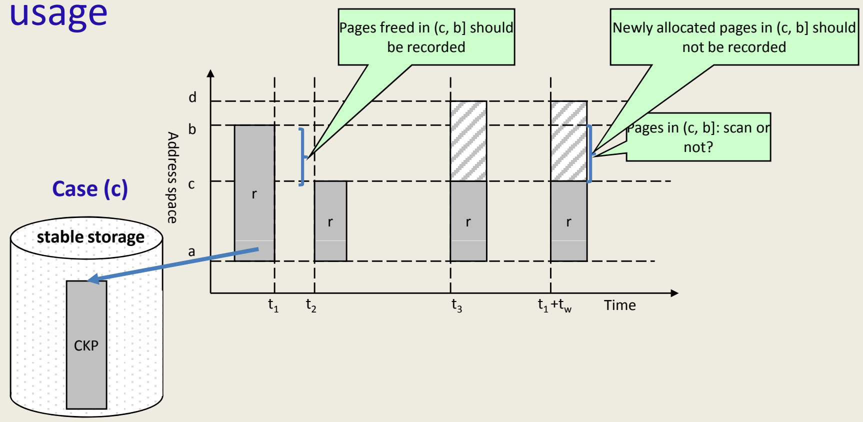
Dynamic Memory Usage

- Complications incurred by dynamic memory usage



Dynamic Memory Usage

- Complications incurred by dynamic memory usage



Dynamic Memory Usage

- Observation:
  - The touch set is always a subset of the checkpoint image, which monotonically decreases
- Solution
  - Record the pages saved in the checkpoint image as the *candidate pages*;
  - Intercept *do_munmap* to capture memory shrinking
  - Scan the intersection of the *candidate pages* and the pages to be freed
  - Update the *candidate pages* to exclude the intersection

Experiments

- A prototype implementation with the BLCR tool in Linux 2.6.22
  - Use a red-black tree for fast touch set tracking
  - Instrument the kernel in a non-intrusive way
- FREM-enhanced BLCR vs. BLCR
  - Evaluated with the SPEC CPU2006 suite
  - Two network settings

Network	Network Latency	Network Bandwidth
SLOW	200 ms	7MB/s
FAST	70 ms	32MB/s

Checkpoint image size

Restart latency

Application (input set)	W (MB)	Touch set size (MB)		RL with BLCR (s)		RL with FREM (s)	
		FAST	SLOW	FAST	SLOW	FAST	SLOW
1: astar (1)	280	34	44	49.4	80.1	6.0 (87.88%)	12.6 (84.26%)
2: bzip2 (5)	847	45	152	161.3	254.1	8.6 (94.64%)	45.6 (82.04%)
3: bzip2 (6)	609	64	244	109.3	176.0	11.5 (89.49%)	70.5 (59.97%)
4: dealII	239	12	28	31.0	57.2	1.6 (94.80%)	6.8 (88.13%)
5: gamess (1)	629	5	9	112.0	180.9	0.8 (99.28%)	2.6 (98.57%)
6: gcc (4)	311	48	82	53.4	87.6	11.6 (78.20%)	23.0 (73.75%)
7: gcc (6)	771	216	211	136.7	221.2	38.2 (72.05%)	60.6 (72.62%)
8: lbm	409	402	402	73.6	118.5	72.3 (1.80%)	116.3 (1.80%)
9: mcf	839	394	827	151.0	242.8	70.9 (53.03%)	239.5 (1.37%)
10: perl (1)	171	31	50	24.7	43.5	4.4 (82.16%)	12.8 (70.66%)
11: soplex (2)	490	186	191	89.3	142.9	33.9 (62.05%)	55.7 (61.06%)
12: wrf	685	37	346	117.8	192.9	54.5 (53.78%)	97.9 (49.25%)

Runtime Overhead during Post-Checkpoint Tracking (in milliseconds)

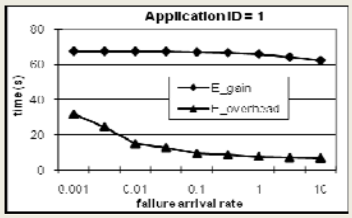
Application (input set)	Scan time	Search and insertion time	Descriptor I/O time	Tracking overhead
1: astar (1)	18.747	13.517	1.360	33.632
2: bzip2 (5)	48.790	0.257	0.184	49.231
3: bzip2 (6)	38.763	0.448	0.149	39.360
4: dealII	4.663	1.718	0.220	6.601
5: gamese (1)	36.711	1.678	0.298	38.687
6: gcc (4)	29.297	4.641	0.549	34.487
7: gcc (6)	43.734	14.954	1.488	60.176
8: lbm	21.832	0.462	0.119	22.413
9: mcf	3.390	0.175	0.161	34.234
10: perl (1)	18.573	13.630	1.350	33.553
11: soplex (2)	29.749	14.359	1.459	45.566
12: wrf	41.594	4.643	0.608	46.845

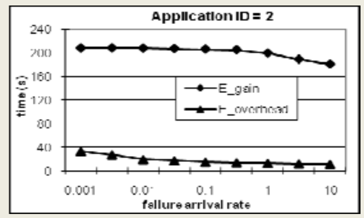
Runtime Overhead during Fast Restart (in seconds)

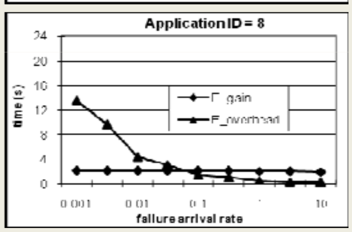
Application (input set)	Remaining image (MB)	Duration of overlapping	Fast restart overhead
1: astar (1)	162	19	7.1
2: bzip2 (5)	476	55	13.1
3: bzip2 (6)	250	48	11.8
4: dealII	144	14	10.2
5: gamese (1)	424	60	21.7
6: gcc (4)	157	19	10.6
7: gcc (6)	560	77	22.7
8: lbm	5	0.8	0.1
9: mcf	8	1.4	0.3
10: perl (1)	83	12	6.9
11: soplex (2)	205	30	4.8
12: wrf	231	35	7.8

Performance Evaluation

- statistically evaluate application performance between two restarts







E_{gain}= expected restart improvement between 2 restarts;
$E_{overhead}$=expected runtime overhead introduced between 2 restarts

Related Work

- Existing work on fast restart
  - OS, Database, Internet services, ...
  - *The novelty of FREM is its ability to reduce restart latency by recording data accesses after CKP*
- Traditional work on C/R
  - Focus on checkpoint optimization, e.g., determining an optimal checkpoint frequency
  - *FREM complements these studies*
- Demand paging
  - *FREM selectively restores the pages needed by tracking data access patterns*

Summary

- Have presented a novel mechanism for fast recovery of C/R applications
  - Through a user-transparent system support
- Have demonstrated its effectiveness
  - FREM can reduce process restart latency by 61.96% on average
- Future work
  - Adaptive tracking window estimation
  - Better image loading mechanism
  - Integration with checkpointing tools

Questions?

FENCE Project Website:

http://www.cs.iit.edu/~zlan/fence.html