The Impact of Recovery Mechanisms on the Likelihood of Saving Corrupted State

Subhachandra Chandra
Cosine Communications

Peter M. Chen
University of Michigan

Motivation

- Computer software is not reliable
- Recovery from failures is vital for usability and availability
- Successful recovery requires that the system does not save data that has been corrupted by the fault
- The recovery system itself may increase the chances of saving corrupted state

Main factors

- Quality of error detection
- Location of the fault
- Frequency of state saves
- Comprehensiveness of state saved

Comprehensiveness / Frequency of State Commits

Frequency

little	automatic state reconstruction	more
visible	failure transparency	transparent
less likely	commits of corrupt state	more likely

Recovery System Determines Comprehensiveness and Frequency

- Generic mechanisms
  - have to save all state
  - have to save state for all visible events
  - e.g. checkpointing, logging

- Application-specific mechanisms
  - know which state is important
  - know which visible events are important
  - e.g. auto-save

Strategies for Saving State

- Three strategies by varying comprehensiveness and frequency
  - LC/LF - Less Comprehensive / Less Frequent application-specific recovery
  - C/LF - Comprehensive / Less Frequent modified generic recovery
  - C/F - Comprehensive / Frequent generic recovery like Discount Checking

Obtaining Faulty Runs

- Inject faults either into the source code or dynamically into the process address space during execution
- Detect failures by comparing output of the run into which faults have been injected with output from a good run
- If the run did not complete or completed with faulty output then it is counted as a failure or faulty run

Detecting Corrupted Committed State: Application-Specific Recovery

- Have a reference run generate all the possible states saved by the application on the disk
- Compare the final state saved by the faulty run on the disk with the list of reference states
- If the final state does not match any of the reference states then corrupted state was committed by the recovery mechanism

Detecting Corrupted Committed State: Generic Recovery

- Recover the application from the last saved checkpoint
- If the application does not complete with the correct results then the run recovered from corrupted state
- Another way to detect if the committed state was corrupted is to check if the last checkpoint was committed after the activation of the fault

Workload and Fault Models

nvi, postgres, oleo

Fault Type	Example of Programming Error
stack	flip random bit
allocation	move use(ptr) to after free(ptr)
heap	flip random bit
off-by-one	substitute < with <=
initialization	delete i=0;
delete branch	substitute "if" for a "while"
delete random instruction	delete a simple statement "i=j+k;"
destination variable	substitute one dest. variable with another

Results for nvi - Application Faults

Fault	Faulty Runs	App-specific	Low Freq App-Generic	App-Generic	Undetected Errors
Stack	50	0	0	0	0
Alloc	50	24	40	50	0
Heap	50	6	12	35	8
Off by One	50	6	7	9	12
Init Errors	50	0	2	2	0
Delete Branch	50	25	27	34	8
Delete Inst	50	12	14	24	3
Change Dest Var	50	1	5	8	5
Total	**400**	**74 (19%)**	**107(27%)**	**162(41%)**	**36(9%)**

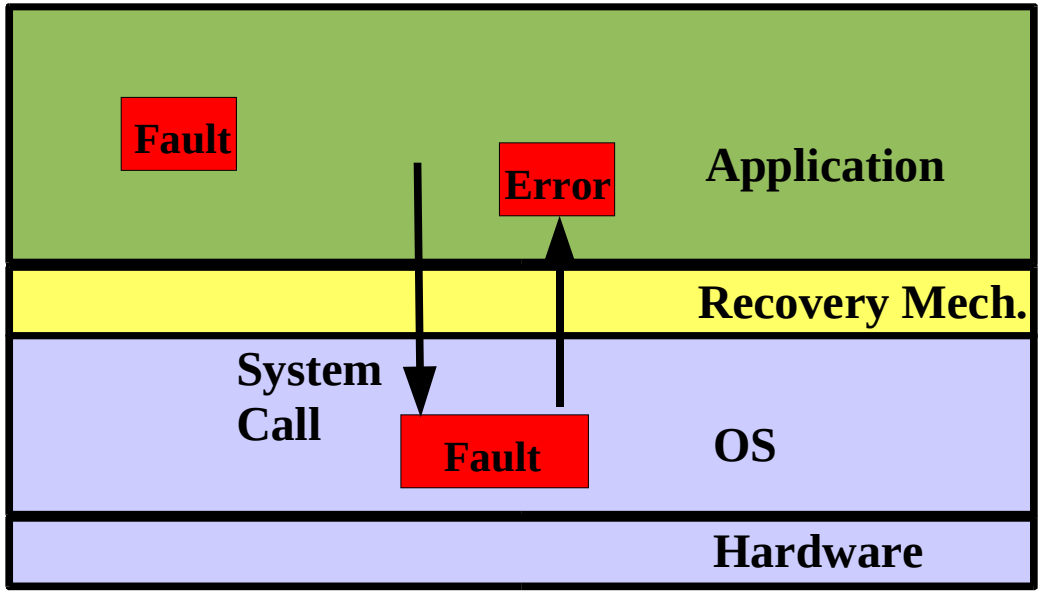
Results for postgres - Application Faults

Fault	Faulty Runs	App-specific	Low Freq App-Generic	App-Generic	Undetected Errors
Stack	50	0	16	17	1
Alloc	50	0	22	24	0
Heap	50	0	0	44	2
Off by One	50	0	0	0	8
Init Errors	50	0	2	3	2
Delete Branch	50	0	0	38	6
Delete Inst	50	1	2	6	5
Change Dest Var	50	2	2	3	0
Total	**400**	**3(1%)**	**44(11%)**	**135(34%)**	**24(6%)**

Results for oleo - Application Faults

Fault	Faulty Runs	App-specific	Low Freq App-Generic	App-Generic	Undetected Errors
Stack	50	0	0	3	0
Alloc	50	0	2	34	9
Heap	50	0	0	12	19
Off by One	50	0	0	10	7
Init Errors	50	0	3	15	8
Delete Branch	50	0	0	19	7
Delete Inst	50	0	2	9	18
Change Dest Var	50	3(1%)	3	5	20
Total	**400**	**3(1%)**	**10(3%)**	**107(27%)**	**88(22%)**

Faults in the Operating System



Results for nvi - OS Faults

Fault	Faulty Runs	App-specific	Low Freq App-Generic	App-Generic	Undetected Errors
Stack	50	0	1	6	0
Alloc	50	1	5	19	0
Heap	50	2	3	4	0
Off by One	50	0	6	11	0
Init Errors	50	3	2	8	1
Delete Branch	50	1	2	12	0
Delete Inst	50	0	1	6	0
Change Dest Var	50	2	0	5	0
Total	**400**	**9(2%)**	**20(5%)**	**71(18%)**	**1(0%)**

Results for postgres - OS Faults

Fault	Faulty Runs	App-specific	Low Freq App-Generic	App-Generic	Undetected Errors
Stack	50	0	5	5	0
Heap	50	1	3	3	0
Off by One	50	0	0	0	0
Init Errors	50	0	0	0	0
Delete Branch	50	1	2	2	0
Delete Inst	50	0	1	2	0
Change Dest Var	50	0	0	0	1(0%)
Total	**350**	**2(1%)**	**11(3%)**	**12(3%)**	**1(0%)**

Results for oleo - OS Faults

Fault	Faulty Runs	App-specific	Low Freq App-Generic	App-Generic	Undetected Errors
Stack	50	4	0	3	0
Alloc	50	0	0	0	0
Heap	50	1	1	1	0
Off by One	50	3	0	0	0
Init Errors	50	0	1	1	0
Delete Branch	50	1	3	4	0
Delete Inst	50	5	0	1	0
Change Dest Var	50	3	4	4	0
Total	**400**	**17(4%)**	**9(2%)**	**14(3%)**	**0(0%)**

Conclusions

- Generic recovery mechanisms are of little use in the presence of application-level faults as they save corrupted state very frequently
- The increased frequency seems to be more due to the frequency of state saves than the comprehensiveness
- When the faults are in the operating system layer the likelihood of saving corrupt state is reduced significantly. Generic recovery mechanisms can be useful in such cases.