2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing

An OS-Hypervisor Infrastructure for Automated OS Crash Diagnosis and Recovery in a Virtualized Environment

Joefon Jann, R. Sarma Burugula, Ching-Farn E. Wu, Kaoutar El Maghraoui
IBM T.J. Watson Research Center
Yorktown Heights, NY, USA
{ *joefon, burugula, cwu, kelmaghr* } @ *us.ibm.com*

***Abstract*—Recovering from OS crashes has traditionally been done using reboot or checkpoint-restart mechanisms. Such techniques either fail to preserve the state before the crash happens or require modifications to applications. To eliminate these problems, we present a novel OS-hypervisor infrastructure for automated OS crash diagnosis and recovery in virtual servers. Our approach uses a small hidden OS-repair-image that is dynamically created from the healthy running OS instance. Upon an OS crash, the hypervisor automatically loads this repair-image to perform diagnosis and repair. The offending process is then quarantined, and the fixed OS automatically resumes running without a reboot. Our experimental evaluations demonstrated that it takes less than 3 seconds to recover from an OS crash. This approach can significantly reduce the downtime and maintenance costs in data centers. This is the first design and implementation of an OS-hypervisor combo capable of automatically resurrecting a crashed commercial server-OS.**

***Keywords*-Operating Systems, Reliability, Availability, Computer Crash, System Recovery.**

I. Introduction

Modern server-OSs are becoming increasingly more complex. This complexity increases the chances of OS crashes/failures because more kernel and dynamically-loaded kernel-modules' bugs will likely be introduced, resulting in more downtime and data loss. Transient hardware failures are another source of OS crashes. [4] showed that about 8% of all system crashes and 9% of all unscheduled reboots were caused by transient hardware failures. Disruption of services could have serious consequences for systems running mission-critical applications. Millions of dollars could be lost due to just a few hours of unavailable computing systems [21]. In a study done by Vision Solutions [3], it has been estimated that the cost of one hour of downtime for a brokerage service could exceed 6.4 million US dollars. Therefore, high availability has become a fundamental requirement for enterprise computing.

To recover from OS crashes, most of the commonly-used enterprise server-OSs today resort to rebooting the OS. A reboot is often quite disruptive because it terminates all the running applications with an attendant loss of work. The OS may also dump its memory contents for diagnosis later, and this may take a long time, particularly for an OS instance with a huge memory footprint. Additionally, the dump-space may not be big enough to hold a large dump, incurring further problems. Sometimes, an OS reboot may not solve the crash problem, and a customer-support engineer may need to be dispatched to the site, which is very costly in time and money, and typically results in significant unscheduled computer downtime.

To alleviate the impact of an OS crash, several recovery techniques have been proposed by academia and industry. Typical solutions include replication [2], checkpoint-and-restart approaches [22], [17], [18], and the recovery box technique [5]. Replication and checkpointing approaches are costly solutions and often are not capable of resuming the execution state of the entire OS and applications to where it was before the crash. Recovery box techniques require major changes to applications, kernel design, and code, and hence are not practical solutions for today's enterprise operating systems.

This paper introduces a novel approach [16] to automatically resume the running of the OS after an OS-crash. Our solution attempts to overcome the weaknesses of the existing approaches, and provides a practical solution for enterprise computer data centers. The key benefit and novelty of our approach is that most of the applications can be automatically resumed without an OS reboot after an OS-crash. Modern operating systems are typically composed of several major components so as to support a wide variety of applications. A fault in one component may not affect all the applications running in the OS. By stopping only those applications that are affected by the fault, and enabling all other applications to continue to run, we can significantly reduce the unscheduled downtimes. In situations where the fault is so serious that the OS cannot function without having the fault fixed, this approach will not work. Our examination of the field data from the world-wide deployment of POWER systems using AIX (IBM's UNIX) and PHYP (IBM's hypervisor for POWER systems) shows that these types of faults are rare. A possible conjecture is that this can be due to the extensive testing of the critical parts of AIX and PHYP (and other operating systems in general) before they are released. As a consequence, most of the execution contexts can be restored by just quarantining the offending process(es). With the above as background, we have designed and prototyped the ALDR (Automated Logical-Partition crash Diagnosis and Recovery) infrastructure with the main goal of minimizing unscheduled downtime caused by commonly occurring OS crashes.

ALDR's approach of automated kernel diagnosis and repair relies on dynamically creating a repair image (ALDR-AIX) from the healthy running OS instance. This small repair image is stored in a memory area hidden from the running OS, and its main goal is to diagnose and resume the running of the OS and most of its applications when an OS crash occurs. Additionally, ALDR's design allows an unlimited number of OS resurrections.

In this paper, we describe the details involved in the creation of the repair image. We also show the diagnosis capabilities of ALDR-AIX. Our emperical evaluations have shown that the crash-recovery-resurrection cycle takes less than 3 seconds, resulting in the end-users not even noticing that a crash had occurred in most cases. Although we have developed the ALDR technology for the AIX-PHYP-POWER platform, this concept should be applicable to other OS-hypervisor-architecture platforms as well. As far as we know, no other prior art or vendor has achieved such fast recovery of a commercial server-OS after an crash. The rest of this paper is organized as follows: Section II discusses relevant related work and compares them with ALDR. Section III describes ALDR's infrastructure design and implentation using the AIX OS and the PHYP hypervisor. Section IV describes the diagnosis and repair component of ALDR. Section V describes how the OS is resurrected. Section VI presents our experimental evaluations. The paper ends with a summary of the benefits, conclusion and future work in Section VII.

II. Related Work

OS fault tolerance has always been at the forefront of both academic and industrial research. Several research efforts have tried to devise techniques to allow systems and user applications to recover from software errors, hardware errors, and OS crashes. Techniques like replication and periodic checkpointing have traditionally been used to enable the restoration of applications' states from power outages, hardware failures, and system crashes. There has also been some recent work to protect from failures originating from the OS itself.

VMware's Fault Tolerance [2] is a solution that provides high availability and reduces applications' downtime due to hardware failures by creating redundant instances of live virtual machines. The downside of this approach is the extra cost required for additional hardware to host redundant virtual machines and the overhead associated with synchronizing with the original images.

Checkpoint and restart approaches [22], [17], [18] allow us to recover some state of applications after an OS crash. In contrast to our work, they can only recover applications that support checkpointing and do not restore all previously running processes, can lose any state that has not been committed, could be time consuming, and incur a lot of overhead. Techniques like in-memory checkpointing [9], [23] reduce significantly the overhead associated with checkpointing to disks, however they require a large amount of memory (up to half the available memory) to work. The recovery box technique proposed in [5] is less costly than checkpoint/restart techniques. However, it requires modifications to the applications and hence cannot be used for many legacy software and applications that do not have corresponding source code available.

Recovering from kernel crashes due to device driver failures have also received a lot of attention from the OS community [6], [14], [10], [13], [24], [20]. These approaches either rely on address-space isolation or microkernel-based techniques. For instance, the Nook framework [24] uses address-space isolation to encapsulate device drivers in their own domain, thereby restricting bogus driver code from accessing other parts of the kernel. Microkernel-based approaches [6], [13] provide an increased level of fault-tolerance through isolating some subsystems into separate address spaces. Upon a failure, the faulty subsystem is restarted and its state is recovered to the previous stored state. However, the microkernel-based solution does not guarantee protection against fault propagation from one OS subsystem to another. Other approaches rely on virtual machines to isolate device driver errors from the kernel [11], [20]. In IBM's AIX-POWER platform, page protection keys are used to isolate different OS components and device drivers so that they do not corrupt each other [25]. The above mentioned approaches do provide great resiliency by preventing device drivers from corrupting other kernel space data; however, they do not prevent OS crashes emanating from bugs in the device drivers or the kernel itself.

In the academic project named Otherworld [7] developed for the Linux-x86 platform, the authors propose to replace the entire kernel with the repair-kernel instead of rebooting separate subsystems. They achieve this by having a hidden crash kernel that takes control once the main kernel crashes. The crash kernel performs the needed recovery procedures and the original kernel is discarded. Otherworld's approach is different from ALDR's approach, as Otherworld tries to revive individual processes one at a time while ALDR revives the whole OS instance. Otherworld also requires some applications to register a crash procedure that will be invoked by the crash kernel for recovery. In addition, they do not have any support from the hypervisor, which limits the applicability of their approach. ALDR does not require the applications to register for anything, nor does it require changes to the applications.

Wang et al, [26] proposed XenPR, a process recovery mechanism based on Xen. Similar to Otherworld and ALDR,

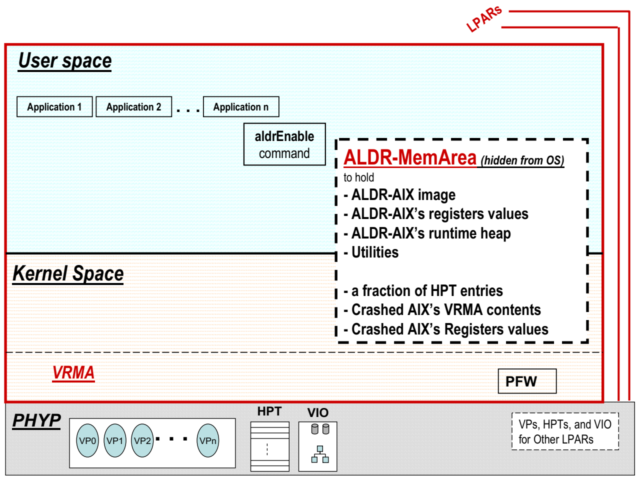

LPARs
User space
Application 1
Application 2
Application n
aldrEnable command
ALDR-MemArea (hidden from OS)
to hold
- ALDR-AIX image
- ALDR-AIX's registers values
- ALDR-AIX's runtime heap
- Utilities
Kernel Space
- a fraction of HPT entries
- Crashed AIX's VRMA contents
- Crashed AIX's Registers values
VRMA
PFW
PHYP
VP0
VP1
VP2
VPn
HPT
VIO
VPs, HPTs, and VIO for Other LPARs

Figure 1: **High-level overview of ALDR**

XenPR also uses a small kernel image that takes control when a crash occurs. However, XenPR relies on saving the process state using files, which are used later to restore the processes to their pre-crash states. This is similar to the checkpoint/restart technique. XenPR currently does not handle a large number of process types (e.g., processes with open sockets and processes that use IPC mechanisms). The approach also incurs overhead because of the need to start the small kernel and save the state into files. In contrast, ALDR does not need to do any state saving into files, as ALDR directly accesses in memory the state of the entire crashed-OS instance with the help of the hypervisor. Also, because ALDR resurrection does not require a full LPAR (Logical PARtition) initialization and OS re-IPL (Initial Program Load), the whole process is very fast.

In general, microkernel-based approaches require a complete re-design of an existing OS, and are not practical to apply to complex commercial server-OSs. For example, Giuffrida et al [12] advocate an event-driven microkernel OS-design, and claim that such a design allows controlled recovery from arbitrary crash failures. In their proposal, the OS consists of a set of separate components, where each component owns its private state, runs in user-space, has its own private data protected by the memory management unit (MMU), and exposes a well-defined interface. CuriOS [6] is another microkernel-based OS that relies on having components store part of their address space in their client's address space. Upon an OS failure, the affected component is restarted and the state is restored from the previous client state. CuriOS's design does not protect from an error that is being propagated from one component to another.

III. ALDR Infrastructure Design & Implementation

Figure 1 shows a high-level overview of ALDR. Each POWER server runs one or more guest OSes or LPARs using the PHYP hypervisor. The AIX LPARs require a small portion of the kernel space memory to be accessed in address-translation-off mode. This region, called RMA (Real Mode Area) is used by OS boot code before translation tables are set up, by the translation-fault handling code during run time, and by a small part of firmware called PFW (Partition FirmWare) that acts as a glue between the kernel and the hypervisor. The POWER architecture also allows RMA to be virtualized by the hypervisor in which case this region is called VRMA. The VIOS LPAR [8] enables the sharing of physical I/O resources across the VIOS-client LPARs within the same managed server. ALDR currently relies on the VIOS to enable quick and efficient suspension and restart of shared physical I/O and network resources.

In a typical crash-recovery-resurrection scenario handled by ALDR, a system administrator, first enables the OS instance for ALDR functionality with a user-space command. When an OS crash occurs, the repair image will be automatically loaded to repair the crashed OS instance. Once repaired, the original OS instance and all unaffected applications will resume their normal operations, as though the crash had never occurred.

ALDR has been designed with the following goals in mind: 1) being able to recover a crashed OS instance without requiring any changes to the applications, 2) having a small memory footprint and zero overhead to the normal running of the OS, and 3) being able to recover in few seconds, so that the impact on network-based applications is minimized. We also assume in the current implementation that it is acceptable to quarantine the offending process[es] for the sake of resuming the operation of the OS and all unaffected applications after an OS crash.

The remaining sub-sections describe in detail how the repair image is created and loaded when a crash occurs. In the remaining discussion, we use the following AIX/PHYP terminologies: In AIX, a Logical CPU is equivalent to a hardware thread. In PHYP, a Virtual Processor is an LPAR abstraction of a physical processor.

A. Repair Image Creation & Registration

When an OS instance is running healthily, the system administrator enables the OS for ALDR diagnosis and recovery by invoking the *aldrEnable* command which performs the following tasks:

1) Create a diagnosis and repair image (ALDR-AIX) from the current OS kernel, current LPAR and software settings, tools and scripts. The ALDR-AIX image created by the *aldrEnable* command is basically a boot image containing a compressed kernel, a device tree, boot initiation code, and a RAM-disk filesystem containing tools for the diagnosis and repair of the failed AIX instance.
2) Create a new IPLCB (Initial Program Load Control Block) tailored for ALDR-AIX's use, based on the IPLCB of the original running AIX instance.

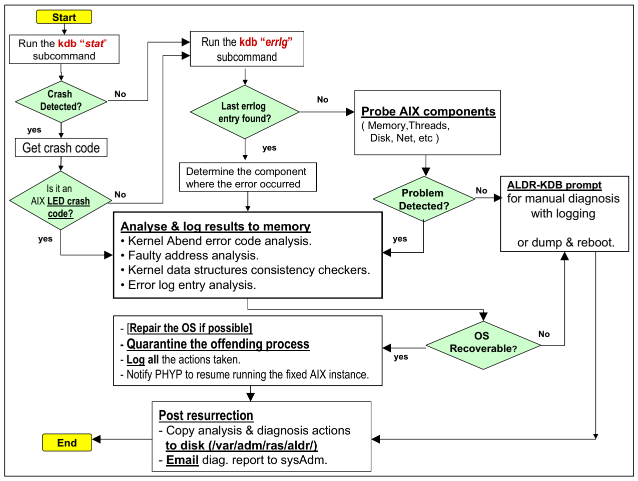
The IPLCB is a data structure that contains all the configuration information necessary for the kernel initialization. The *aldrEnable* command significantly trims the main OS's IPLCB so that the repair image will see only the limited amount of resources that it can use. For example, running on only one hardware thread, accessing a limited amount of memory, and not being able to access any IO devices except the system console.

3) Manufacture the starting registers' values for the hardware thread that ALDR-AIX is going to run on.
4) Create a small data structure, called the ALDR Item Table, to keep track of all the meta-data that is required to boot and run ALDR-AIX. This table is very small and its only purpose is to keep all the crashed-AIX instance-specific data in one place for lookup by ALDR-AIX.
5) Carve out a chunk of memory from the running OS instance to store the ALDR-AIX image, the ALDR Item Table, the modified IPLCB, the starting registers' states for virtual processor 0, and the contents of the RAM disk filesystem, which contains diagnosis tools. This memory region, called *ALDR-MemArea*, is hidden from the main OS to prevent any accidental overwriting.
6) Create a kernel process for each hardware thread that the original OS instance runs on. This kernel process just sleeps waiting for the resurrection event to occur. This kernel process is needed to initialize a few processor hardware resources before the repaired AIX can run again.
7) Finally, register the ALDR-MemArea contents with the hypervisor.

The hypervisor, upon receiving the ALDR registration request from AIX, saves the location and size of each item into PHYP's internal data-structures. It will then internally mark the LPAR as ALDR-enabled, so that a future OS crash detected will automatically load and trigger ALDR-AIX into action. The hypervisor saves the address locations of ALDR-AIX's boot image, initial register values, IPCLB, and ALDR item table. When the OS crashes, PHYP must copy the ALDR-AIX image into the VRMA. The reason for this copying is that ALDR-AIX has to access memory in translation-off mode during its boot time, whereas the higher memory location where ALDR-MemArea resides cannot be accessed in translation-off mode.

B. Repair-Image Invocation

When AIX instance crashes, an OS termination event is sent from the OS to the hypervisor. Upon receiving this event, the hypervisor saves into the ALDR-MemArea the status of the crashed OS instance that may be erased/overwritten by ALDR-AIX while performing its work. This includes the contents of the VRMA, the registers' states of virtual processor 0 (VP0), and the first 8MB of the Hardware Page Table (HPT) entries. The hypervisor then prepares the LPAR for use by ALDR-AIX by copying the repair image and its data structures from the ALDR-MemArea into the VRMA area, loading the pre-manufactured registers's state into VP0, and clearing the first 8MB the HPT entries for use by ALDR-AIX.


Start
Run the kdb "stat" subcommand
Crash Detected?
No
yes
Get crash code
Is it an AIX LED crash code?
No
yes
Run the kdb "errlg" subcommand
Last errlog entry found?
No
yes
Probe AIX components
(Memory,Threads, Disk, Net, etc)
Determine the component where the error occurred
Problem Detected?
No
yes
ALDR-KDB prompt
for manual diagnosis with logging
or dump & reboot.
Analyse & log results to memory
• Kernel Abend error code analysis.
• Faulty address analysis.
• Kernel data structures consistency checkers.
• Error log entry analysis.
OS Recoverable?
No
yes
- [Repair the OS if possible]
- Quarantine the offending process
- Log all the actions taken.
- Notify PHYP to resume running the fixed AIX instance.
Post resurrection
- Copy analysis & diagnosis actions to disk (/var/adm/ras/aldr/)
- Email diag. report to sysAdm.
End
Figure 2: **ALDR-AIX's Diagnosis & Repair Flow Chart**

The HPT contents for the crashed OS must be kept unchanged so that diagnosis can be made. To preserve these HPT contents, the hypervisor saves the first 8MB of HPT entries into ALDR-memArea, and limit the size of the HPT to 8MB before activating ALDR-AIX; i.e. we restrict ALDR-AIX to run with only the first 8MB of HPT entries. However, to enable ALDR-AIX's kernel debugger to access all the memory locations used in the original OS (so as to perform diagnosis), we created a new hypervisor call for ALDR-AIX's kernel debugger to access those HPT entries that are beyond the first 8MB. Additionally, any access to virtual I/O devices is disabled during the operations of the repair image to prevent virtual devices from changing any state. This is accomplished through a new hypervisor call that is issued right before invoking ALDR-AIX.

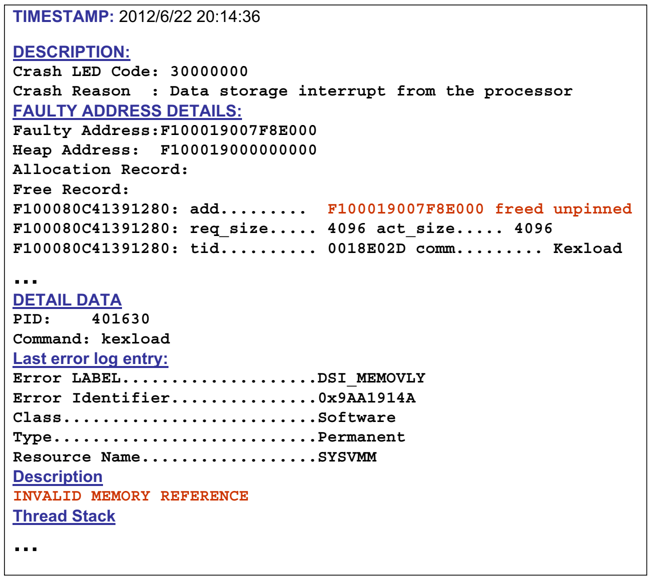
ALDR-AIX is not booted with the normal re-IPL code, as a normal re-IPL would initialize memory and the HPT contents, destroying all the data needed for performing diagnosis of the crashed OS. For this, we defined a new boot type in PHYP which allows us to re-use and customize LPAR initialization routines to activate ALDR-AIX without a re-IPL. Finally, PHYP loads the initial registers' values manufactured for the single-threaded ALDR-AIX into the primary thread of VP0. All other hardware threads are kept in a state where they do not execute any instruction from the failed LPAR.

IV. OS-Crash Diagnosis and Repair

When the repair image is up, it automatically runs the diagnosis and repair scripts provided in the RAM disk filesystem. Note that in the current implementation, all the diagnosis and repair are done using Perl scripts invoking the standard AIX kernel debugger program kdb [15]. The kdb kernel debugger has an extensive set of subcommands and tools that are used by developers to debug device drivers, kernel extensions and the kernel itself. kdb can run on a running (or frozen) AIX kernel or on a system dump file produced by a previously crashed OS. We modified kdb to add functions needed by ALDR-AIX for diagnosis and repair. These functions act as building blocks for the diagnosis and repair Perl scripts, and significantly reduce the complexity of these Perl scripts. We refer to the modified version of kdb as ALDR-kdb.

In the diagnosis component of ALDR, we rely heavily on the rich functionality provided by kdb to identify the type of crash that occurred, analyze its cause, and change the necessary kernel data structures to do any repair. Figure 2 shows the general flow of the steps used during diagnosis and recovery. Initially, the kdb subcommand *stat* is used to display the status of the processor where the crash occurred. The *stat* subcommand also displays crash-related information such as the crash type and the code execution stack that triggered the crash. The crash code is decoded to determine if it is one of the standard AIX crash codes. If so, we proceed to the repair phase. If we cannot detect what type of crash happened, we use the *errlg* kdb subcommand to examine AIX error logging data structures to check if any error has been logged prior to the crash. The last logged error entry can give useful insights into what component caused the crash, since many AIX kernel components use the error logging subsystem to log various errors encountered. Based on the result of the *errlg* subcommand, we perform further analysis in the suspected AIX component. For example, if the problem was in the main memory subsystem, we check for memory leaks, insufficient paging space, etc. If the problem was detected in the storage subsystem, we check for full filesystems, metadata corruption, etc. If the problem was detected to be in the process subsystem, we check if it was a deadlock, or if there were too many threads per CPU.

As part of ALDR crash diagnosis, we use the RTEC (RunTime Error Checking) features in AIX to check if there has been any error checking active prior to the crash. For instance, the memory allocator component of AIX provides a service called Xmalloc debug (XMDBG) that provides significant error checking for the *xmalloc()* and *xmfree()* kernel services. Typical errors include freeing memory that has not been allocated, memory leaks, and referencing uninitialized memory or free storage [19]. Figure 3 shows snippets of an ALDR diagnosis report that resulted from referencing a recently freed memory location. The report shows that the



```
TIMESTAMP: 2012/6/22 20:14:36

DESCRIPTION:
Crash LED Code: 30000000
Crash Reason  : Data storage interrupt from the processor
FAULTY ADDRESS DETAILS:
Faulty Address:F100019007F8E000
Heap Address:  F100019000000000
Allocation Record:
Free Record:
F100080C41391280: add.........  F100019007F8E000 freed unpinned
F100080C41391280: req_size..... 4096 act_size..... 4096
F100080C41391280: tid.......... 0018E02D comm......... Kexload

...
DETAIL DATA
PID:    401630
Command: kexload
Last error log entry:
Error LABEL...................DSI_MEMOVLY
Error Identifier..............0x9AA1914A
Class.........................Software
Type..........................Permanent
Resource Name.................SYSVMM
Description
INVALID MEMORY REFERENCE
Thread Stack

...
```

Figure 3: **Diagnosis Report of a DSI Crash (Snipets)**

faulty address that caused the crash has already been freed. Another use of RTEC is the use of kernel data structure consistency checkers. Our diagnosis runs several consistency checkers to check that key kernel data structures have not been corrupted. Examples structures are: thread, proc, ppda (Per Processor Data Area), MST (Machine STate save area), etc.

One way of recovering from an OS crash is to quarrantine the offending thread(s). Once the offending thread causing the crash is identified, ALDR-AIX quarantines the offending thread by removing it from the ready-to-run queue (see Figure 4). The state of this offending AIX thread is changed to ”sleep” mode, and its owning process’ state is changed to ”stop” mode. The previously running threads in all other Logical CPUs are placed into the ready-to-run queue, so that they can be re-dispatched by the OS once the recovery kernel process finishes its execution. ALDR-AIX sets up the registers’ values of all the Logical CPUs in the LPAR such that the special kernel process created by the *aldrEnable* command for each Logical CPU will be the first one to run on each Logical CPU when the hypervisor resumes the original OS.

Whether or not ALDR-AIX succeeds in repairing the crashed-AIX instance, a diagnosis report will be produced, which will be copied to disk when the fixed original-AIX resumes running, or when the original AIX re-IPLs if ALDR-AIX is not able to fix the problem. Additionally, email notifications can be automatically sent to a user-specified list of email-IDs in all situations.

V. OS Resurrection Phase

After the crashed OS is diagnosed and repaired, ALDR-AIX informs PHYP to reload the fixed OS instance. The hypervisor then restores the saved VRMA memory contents

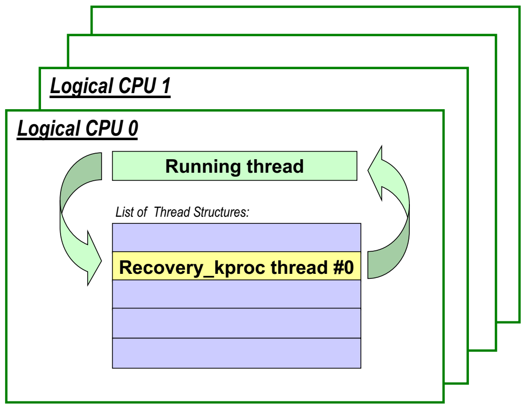


Figure 4: **Quarantine the offending thread**

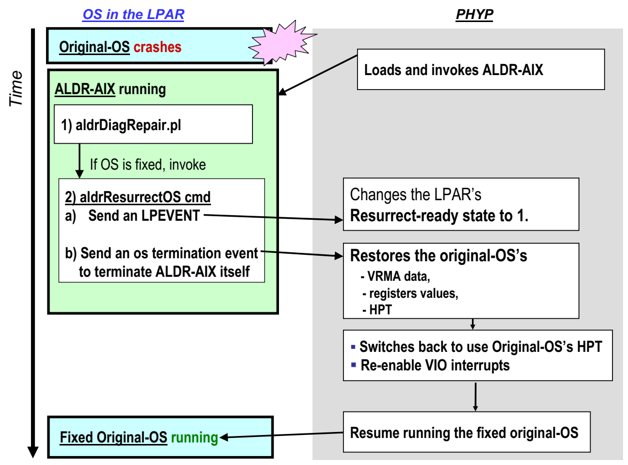

OS in the LPAR
PHYP
Time
Original-OS crashes
Loads and invokes ALDR-AIX
ALDR-AIX running
1) aldrDiagRepair.pl
If OS is fixed, invoke
2) aldrResurrectOS cmd
a) Send an LPEVENT
b) Send an os termination event to terminate ALDR-AIX itself
Changes the LPAR's Resurrect-ready state to 1.
Restores the original-OS's
- VRMA data,
- registers values,
- HPT
Switches back to use Original-OS's HPT
Re-enable VIO interrupts
Resume running the fixed original-OS
Fixed Original-OS running

Figure 5: **ALDR resurrection steps**

and register values of the original OS instance from the ALDR-memArea, restores the first 8MB of HPT entries from the ALDR-memArea, and switches back to use the HPT of the original OS instance. Figure 5 shows the resurrection steps in ALDR.

Similar to the ALDR-AIX loading steps, we created a special boot type in PHYP, under which we use customized LPAR initialization routines to resurrect the repaired OS without a re-IPL. Hardware threads are then put back into the PHYP online state, and virtual processors are put into PHYP ready-to-run state. PHYP then invokes the new hypervisor call to enable VIO interrupts, which in turn re-starts virtual I/O devices such as virtual disks and network. Since the virtual interrupts for the system console were never disabled, the system console continuously works throughout the whole process from system crash to resurrection.

When the repaired AIX instance gets control from PHYP, the first code to be executed on each of the virtual processors of the LPAR is the code of the kernel process created by the *aldrEnable* command. These kernel processes will perform hardware initialization such as clearing the translation buffers and loading some pinned entries into the translation buffers in each processor. As mentioned previously, the resurrected AIX instance then automatically re-enables the LPAR for ALDR to be ready to handle a future OS crash.

VI. Evaluation

We developed various crash-injection kernel extensions to cause an OS crash. Using the IBM Internal Post-Sale Database (PSdb), a database where customers report to IBM the technical problems they encounter with IBM products, we surveyed the most frequent kernel crashes that occurred at customers sites. Based on this survey, we developed these kernel extensions to inject the most frequently occurring kernel crash types:(1) a DSI (Data Storage Interrupt) which occurs when access to data in memory is illegal, (2) an ISI (Instruction Storage Interrupt) which occurs when an instruction fetch cannot be performed, (3) a PGM (ProGraM exception) such as an exception caused by an illegal instruction or a privileged instructions, and (4) a FLP (FLoating-Point exception) which occurs when the floating-point available bit `MSR[FP]` in the Machine Status Register is cleared and a floating point instruction is executed. All evaluation experiments were conducted in a VIOS client LPAR running the IBM AIX v6 OS with POWER7 processor clock speed of 3864 MHz. The LPAR has 16GB of RAM, 2 physical CPUs running with 4 hardware threads, and 25GB of disk storage. We evaluated the capability and speed of ALDR resurrection a variety of applications. For each application, 4 types of crashes were injected while the application was running. Each experiment was repeated at least 3 times to ensure the consistency of the results. After each ALDR recovery, we examined the state of the application to see how well ALDR recovered the application and its data. Details about the applications tested are given below:

1) vi: Text editing into vi's buffer was being performed while the crashes were injected.
2) gzip: We injected the crashes while compressing a large file (384 MB).
3) Stream TCP sockets: A client/server program that uses TCP sockets within the same LPAR. A large number of messages were being sent from the client to the server.
4) Datagram UDP sockets: Same as the TCP sockets with the difference of using UDP sockets.
5) ftp: Crash injected while transferring a 1GB between a local LPAR and a remote one.
6) vnc: A VNC (Virtual Networking Computing) server was started in an LPAR. A VNC client was invoked from a Windows machine to connect to the VNC server. Several terminal windows were opened from the VNC client, and an OS crash was injected.
7) DB2: The DB2 database test consisted of driving DB2 with a set of queries of the DayTrader benchmark.

DayTrader [1] is a Websphere benchmark application that emulates an online stock-trading system. The application allows users to perform trading operations. In our experiments the DB2 instance ran on the LPAR where the crashes were injected, while the WebSphere Application Server was running on a different LPAR. A workload simulator was used to simulate 500 concurrent trader clients stressing the DB2 database.

Table I summarizes the experimental results, which show that in all cases ALDR was able to successfully resurrect the running applications, and without much overhead. On the average, it takes less than 18 seconds to enable ALDR, which is a one-time cost. Our results also show that the recovery time is independent of the type of application being run. On the average, ALDR resurrection took only less than 3 seconds. This is 2 orders of magnitude faster than the service downtime that Otherworld [7] have achieved for a similar class of applications. For the vi test, the text editor was able to recover the session as well as the buffer used for copying/yanking data. The gzip compression test was also successful, and gzip continued its operation after ALDR recovery. We also tested the correctness of the compression by de-compressing the resultant compressed file and comparing it with the original file. The sockets applications also continued their operations after ALDR resurrections. For the VNC test, from a Windows VNC-clients's viewpoint, because the recovery time was very short, it seemed to the user that no crash had ever happened. For the DayTrader test, we also verified correctness by comparing the DB2 client logs generated with and without a crash, and they were identical, which shows again that ALDR was successful in resurrecting the DB2 connectivity with the DayTrader client.

Applications	Average aldrEnable Time(sec)	Resurrection Time(sec.)			
		DSI	ISI	PGM	FLP
vi	17.88	2	3	2	3
$gzip$	16.45	3	2	2	2
$StreamSockets$	15.99	2	3	2	3
$DatagramSockets$	18.02	2	3	3	2
ftp	18.07	3	3	3	3
vnc	18.77	2	3	2	3
$DB2$	17.54	2	3	3	3
Average	**17.5**	**2.3**	**2.9**	**2.4**	**2.7**

Table I: **ALDR diagnosis-and-resurrection time and aldrEnable time for applications.**

VII. Benefits, Conclusions, and Future Work

High availability is becoming increasingly critical to today's enterprise data centers; hence it is essential for commercial server-OS and hypervisors to provide the necessary flexible tools to reduce or eliminate downtimes. Providing complete automation of OS-crash detection, repair and recovery is extremely useful and attractive, and can greatly reduce the downtime and administrative costs of enterprise data centers. The ALDR infrastructure is a first attempt in this direction, and its benefits and potentials are numerous:

1) ALDR's automated diagnosis and recovery from OS crashes/failures saves much more time than manual analysis of dumps.
2) Diagnosis report is automatically generated and emailed to user-specified system administrators.
3) No changes to applications is needed for ALDR to function.
4) Almost zero path-length has been added to the main-line code of OS and hypervisor.
5) ALDR is particularly useful in OS instances with huge memory foot-prints, because it eliminates the need to send very large dumps (which may not be possible to create if dump space is limited), resulting in much reduced time and costs for problem resolution.
6) Having access to almost all the memory of the crashed OS by the repair image enables more accurate diagnosis, compared to manual diagnosis of emailed dumps, which are often very restricted in scope.
7) ALDR will be very useful for Lights-Out data centers.
8) ALDR's infrastructure can potentially be used for software hotpatch or update.
9) ALDR's concept can be extended to other OS-Hypervisor-Architecture types e.g. Linux-PHYP-POWER, etc.

We have presented in this paper the design and implementation of a novel technology for automated and very fast OS-crash diagnosis and recovery. Our evaluations have shown that ALDR is capable of successfully resurrecting all the tested applications without having to reboot the OS, and without the need to change any application. In all our evaluation experiments, we found that the whole ALDR cycle of diagnosis, repair and recovery takes less than 3 seconds. Hence ALDR provides a very powerful and useful technology/infrastructure for automated OS-crash analysis and recovery.

Future work to enhance ALDR include increasing the scope and types of crashes that ALDR can handle, providing additional recovery techniques, attempting to fix the offending process instead of quarantining it, and testing ALDR in much larger commercial application scenarios.

References

[1] Apache DayTrader benchmark. https://cwiki.apache.org/GMOxDOC20/daytrader.html.

[2] VMware fault tolerance, http://www.vmware.com/products/fault-tolerance/overview.html.

[3] Arnold, A. Assessing the financial impact of downtime, http://www.it-director.com/business/costs/content.php?cid=12043. Tech. rep., Vision Solutions, 2010.

[4] Arthur, S. Fault resilient drivers for longhorn server, winhec 2004 presentation, available at www.muglin.ru/DW04012_WINHEC2004.ppt, 2004.

[5] Baker, M., and Sullivan, M. The recovery box: Using fast recovery to provide high availability in the unix environment. In *In Proceedings USENIX Summer Conference* (1992), pp. 31–43.

[6] David, F. M., Chan, E., Carlyle, J. C., and Campbell, R. H. Curios: Improving reliability through operating system structure. In *8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008, December 8-10, 2008, San Diego, California, USA, Proceedings* (2008), R. Draves and R. van Renesse, Eds., USENIX Association, pp. 59–72.

[7] Depoutovitch, A., and Stumm, M. Otherworld: giving applications a chance to survive OS kernel crashes. In *EuroSys* (2010), C. Morin and G. Muller, Eds., ACM, pp. 181–194.

[8] Devenish, S., Dimmer, I., Folco, R., Roy, M., Saleur, S., Stadler, O., Takizawa, N., and Vetter, S. *IBM PowerVM Virtualization Introduction and Configuration, available at http://www.redbooks.ibm.com/abstracts/SG247940.html*. IBM Corporation, 2011.

[9] Dong, X., Xie, Y., Muralimanohar, N., and Jouppi, N. P. Hybrid checkpointing using emerging nonvolatile memories for future exascale systems. *ACM Trans. Archit. Code Optim. 8* (June 2011), 6:1–6:29.

[10] Erlingsson, Ú., Abadi, M., Vrable, M., Budiu, M., and Necula, G. C. Xfi: Software guards for system address spaces. In *OSDI* (2006), USENIX Association, pp. 75–88.

[11] Fraser, K., H, S., Neugebauer, R., Pratt, I., Warfield, A., and Williamson, M. Safe hardware access with the Xen virtual machine monitor. In *1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS)* (2004).

[12] Giuffrida, C., Cavallaro, L., and Tanenbaum, A. S. We crashed, now what? In *Proceedings of the Sixth international conference on Hot topics in system dependability* (Berkeley, CA, USA, 2010), HotDep'10, USENIX Association, pp. 1–8.

[13] Herder, J. N., Bos, H., Gras, B., Homburg, P., and Tanenbaum, A. S. Construction of a highly dependable operating system. In *Proceedings of the Sixth European Dependable Computing Conference* (Washington, DC, USA, 2006), IEEE Computer Society, pp. 3–12.

[14] Herder, J. N., Bos, H., Gras, B., Homburg, P., and Tanenbaum, A. S. Failure resilience for device drivers. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (Washington, DC, USA, 2007), DSN '07, IEEE Computer Society, pp. 41–50.

[15] IBM. AIX version 7.1: KDB kernel debugger and kdb command, available at http://publib.boulder.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.kdb/doc/kdb/kdb_pdf.pdf. Tech. rep., IBM Corp, 2011.

[16] Jann, J., Lindeman, J., Burugula, R. S., Wu, C.-F., and Maghraoui, K. E. Automated transition to a recovery kernel via firmware-assisted dump flows providing automated operating system diagnosis and repair. *US Patent 8132057*.

[17] King, S. T., Dunlap, G. W., and Chen, P. M. Debugging operating systems with time-traveling virtual machines. *Proc USENIX Annual Technical Conference* (2005), 1–15.

[18] Laadan, O., and Nieh, J. *Transparent Checkpoint-Restart of Multiple Processes on Commodity Operating Systems*. USENIX Association, 2007, pp. 323–336.

[19] Lascu, O., Bodily, S., Harvala, M., Singh, A. K., Song, D., and Berg, F. V. D. *IBM AIX Continuous Availability Features, available at www.redbooks.ibm.com/abstracts/redp4367.html*. IBM Corporation, 2008.

[20] LeVasseur, J., Uhlig, V., Stoess, J., and Gotz, S. Unmodified device driver reuse and improved system dependability via virtual machines. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6* (2004), pp. 17–30.

[21] Martinez, H. How much does downtime really cost? , http://www.informationmanagement.com/infodirect/2009_133/downtime_cost-10015855-1.html. Tech. rep., InfoManagement, 2010.

[22] Sancho, J. C., Petrini, F., Davis, K., Gioiosa, R., and Jiang, S. Current practice and a direction forward in checkpoint/restart implementations for fault tolerance. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 18 - Volume 19* (Washington, DC, USA, 2005), IPDPS '05, IEEE Computer Society, pp. 300.2–.

[23] Srinivasan, S. M., Kandula, S., Andrews, C. R., and Zhou, Y. Flashback: A lightweight extension for rollback and deterministic replay for software debugging. In *USENIX Annual Technical Conference, General Track* (2004), pp. 29–44.

[24] Swift, M. M., Annamalai, M., Bershad, B. N., and Levy, H. M. Recovering device drivers. *ACM Trans. Comput. Syst. 24* (November 2006), 333–360.

[25] Vetter, S., Aleksic, R., Castillo, I. N., Fernandez, R., Rll, A., and Watanabe, N. *IBM AIX Version 6.1 Differences Guide*. IBM Redbooks, 2008.

[26] Wang, M., and Chen, H. The design and implementation of process recovery mechanism based on Xen. In *Proceedings of the 2011 International Conference on Business Computing and Global Informatization* (Washington, DC, USA, 2011), BCGIN '11, IEEE Computer Society, pp. 581–584.