The Report Committee for Omar Cardona
Certifies that this is the approved version of the following report:

RAS Enhancements for RDMA Communications

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor:

Scott Nettles

William Bard

RAS Enhancements for RDMA Communications

by

Omar Cardona, B.S.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2010

Dedication

I dedicate this report to my wife Lizandra for her help and support during this journey.

Abstract

RAS Enhancements for RDMA Communications

Omar Cardona, M.S.E.

The University of Texas at Austin, 2010

Supervisor: Scott Nettles

Ethernet as the communication medium in the enterprise data center has outlived all competing mediums and resisted the test of time with regards to speed and costs. The future is also poised for growth with 40 and 100Gps speeds just over horizon. The current state of the technology is being enhanced and extended with lossless features to allow for fabric convergence of Storage and Inter Process Communication (IPC) Networks. It is under this medium that an increase in the adoption of Remote Direct Memory Access (RDMA) over Ethernet using offloaded TCP/IP (iWARP) and Infiniband over Ethernet (RoCE) communication stacks to RDMA capable NIC adapters (RNIC) is observed.

RDMA enables direct application to application communication over the network resulting in numerous and significant benefits such as reduced CPU utilization, lower latency communications, increased energy efficiency, and reduced overall system

requirements. However, with said benefits also comes increased software complexity in how RDMA interface users communicate. The RDMA communication semantics, which originate from the High Performance Computing (HPC) domain, are heavily biased towards Low-Latency and High-Bandwidth communications rather than Reliability, Availability, and Serviceability (RAS). As adoption increases, and enterprise data centers begin to leverage RDMA over Ethernet, enhancements to the OS stack software architecture and design of the components involved is required to address these deficiencies. Operating system interfaces, device drivers, adapter hardware design, and embedded firmware features must be viewed from a high-availability and maintainability point of view.

RAS enhancements for RDMA communications proposes the software architectural tradeoffs for enhancing the iWARP and RoCE RDMA implementations for communications in the enterprise data center, with new and traditional RAS features for existing communications stacks and devices. The architecture leverages software enhancements in traceability, availability, maintainability, serviceability, fault-isolation and resource management; such that in the advent of errors, the probability that the forensics data points to identify root cause are immediately and automatically available is increased.

Table of Contents

List of Tables

List of Figures

1.0 INTRODUCTION

Ethernet is the de-facto networks interconnect for the enterprise data center and end users. It has resisted the test of time with the consistent increases in bandwidth and decreases in cost relative to competitors such as ATM, FDDI, and others. The most recent challenge to this position was provided by the Infiniband (IB) physical layers and stacks. Again, due to low cost, high-speed, abundant availability of management skills, and richness in products and markets, the Infiniband concepts were translated over to Ethernet in an attempt to exploit commodity 40 and 100Gbps physical links. The attempt became official with the recent release of the Remote Direct Memory Access (RDMA) over Converged Ethernet (RoCE) Annex to the IB specification [1]. Prior to the Annex, a variant of RDMA over Ethernet was available via the Internet Wide Area RDMA Protocol (iWARP) [2]. The major differences between the two, being that the Network (L3) and Transport (L4) layers under iWARP are based on hardware offloaded TCP/IP, whereas the equivalent services on RoCE are based on IB Network and Transport layers.

The RDMA concepts were developed primarily for Low Latency Inter-Process Communications (IPC) in Clusters and High-Performance Computing (HPC) environments. In said deployments, performance is king; and everything else is a distant second. Therefore, it is of no surprise that the architectural design of the communication mechanisms, adapters, and protocols would result in tradeoffs that favor performance over reliability, availability, and serviceability (RAS). In HPC environments, applications, systems software, and hardware resources are typically devoted to executing large complex problems for long periods of time. Under these types of operating environment, the threshold for hardware and software errors tends to be higher than for enterprise data centers and end users. In contrast, the typical modern enterprise data

center is dynamic, autonomous, highly virtualized, and demands high availability. Compute resources and systems exploit virtualization and mobility mechanisms for fault tolerance and Disaster Recover (DR), therefore RAS is king above all else; obviously in contrast to HPC environments. The enterprise data center customer does not typically have the luxury of restarting a compute job in the advent of errors, as delays in work completion immediately translate to increased operation and business costs.

From an application point of view, the traditional network communication occurs via the Operating System (OS), where said entity owns and mediates access to the protocol stacks (TCP/IP, FC, iSCSI, etc.) and Network Interconnect Cards (NIC). The OS provides a simplified communications API, such as Sockets that abstracts and performs the data movement operations on behalf of the application. The OS is the mediator of the data exchange operations between two applications over the network, hence the visible, traceable, and verifiable RAS mechanisms are readily available via OS kernel state; three key traits necessary for enterprise RAS. From a serviceability point of view, it is common to autonomously obtain forensics data, traces, dumps, logs etc., after system errors to perform root cause analysis.

RDMA differs from traditional OS communications in that its core design point it to provide the most direct and efficient network communication mechanism possible. Therefore it explicitly bypasses the OS during application communication so as to avoid the costs of intermediate buffer copies, protocol processing, and context switching. The application communicates directly with the adapter via a channel (Endpoint) and set of application user specific resources mapped to the underlying NIC. The protocol processing for OSI layers 3-4 and equivalent are executed within the NIC, effectively providing a stateful hardware offload acceleration mechanism and circumventing protocol processing by the host CPU. With both a direct channel to the adapter and

protocol processing in hardware, the elimination of the Userspace to Kernelspace context switch is necessarily eliminated. Thus, in a nutshell, RDMA allows for direct application-to-application data exchange over the network and eliminates the processing and latency overhead inherent in the OS mediated network access design. Clearly, the traditional role of the OS from a RAS point of view has been eliminated.

In both the iWARP and RoCE standards specification, software RAS capabilities are undefined and left as an exercise to the implementer, as is typical in most standards specifications. However, the HPC community and US Department of Defense (DoD) have recognized that performance above all else has an inherent cost. The High Productivity Computing Systems (HPCS) initiative [3] is an attempt to address this need via research into productivity, programmability, and robustness of both software and hardware. This initiative recognizes that the architecture, tools, and components should leverage human productivity over the pursuit of unrealistic performance benchmarks [4]. Though the focus of said research is within the Userspace space domain, the subset of requirements necessarily translates to RAS from an OS point of view when adopting RDMA in an enterprise data center. Given the aforementioned design points, the core problem present for RDMA implementations is the lack of standardized mechanisms or best practice guidelines for providing enterprise level RAS capabilities to RDMA enabled applications. The traditional OS stack communication mechanism, albeit not as efficient as RDMA, allows for system-wide view of operations, data, and resources. A new approach is therefore needed to address the expected enterprise RAS requirements. Furthermore, the RAS architecture should work equally well for both iWARP and RoCE given the amount of commonality in their interfaces and functionality.

A contrast of the RAS deficiencies behind a typical RDMA implementation is necessary; along with the architectural definition and best practices guidelines to balance

the performance benefits of current implementations of RDMA, with the enterprise level RAS features necessary to achieve efficiency in serviceability and maintainability. Research into the existing body of work in the industry and academia is used to leverage software and hardware RAS tools and techniques for applicability to RDMA type implementations.

The remainder of this report is structured as follows. Section 2 provides a technical overview, discussion and comparison of existing RDMA implementation under iWARP and RoCE. Section 3 details the motivation for implementation of the RAS architecture under a typical enterprise data center deployment. The related work in section 4 details the existing applicable RAS approaches currently in use in the industry and academia for advantages and deficiencies. Section 5 encompasses the general proposed architecture and supporting items. Lastly, section 6 concludes with the summarized benefits and results of adopting the architecture.

2.0 RDMA TECHNICAL OVERVIEW

The concept of direct application-to-application communication over the network originates from the Virtual Interface Architecture (VIA) [5]. The goal was to provide a high performance communications interface with direct access to the NIC from Userspace, and equivalent memory protection and isolation mechanisms as provided by the traditional communications stack within the OS model. The VIA design concepts did not fully define a hardware architecture for stateful offload of protocols, thus it transitioned through several iterations and standards bodies, and is currently integrated into both the Infiniband Architecture Specification [6] and the iWARP IETF specifications [2].

In order to frame the discussion on RDMA specifics, it is necessary to discuss a subset of the differences and commonalities of IB and Ethernet. The IB Architecture defines a different set of L1-2 network standards, cables, connectors, and switches that leverage multi-path and subnet management capabilities tailored for communication between servers rather than clients. These capabilities are not available in standard Ethernet, though recent standards work under Converged Ethernet [7] [8] [30] attempts to address these deficiencies. Regardless, the differences in L1-2 necessarily drive the need to preserve existing Ethernet L1-2 fabrics in the enterprise, since IB cannot fully replace it. iWARP eliminates the need for separate fabrics, since it utilizes standard Ethernet and familiar L3-4 management. However, this comes at a cost; lower market adoption versus IB, fewer enterprise and HPC deployments, and lack of equivalence in connection types and RDMA operations which are detailed in later sections. In short, the majority of existing RDMA applications are IB based rather than iWARP based. Recently, the IB Architecture was extended with the RoCE Annex to define and standardize IB L3-4

operation over standard Ethernet, such as to allow for exploitation of existing IB RDMA enabled applications while preserving the familiarity of Ethernet L1-2 in the enterprise.

RDMA operations are based on the concept direct application-to-application communication via OS bypass. In order to achieve said functionality, it is necessary to understand in detail how it is achieved. The discussion is based on the premise of unsustainable processing scalability for network IO as Ethernet speeds increase; as depicted, (Table 1) by the approximate cost of Sockets based communication at 10Gbps. It is clear that the host CPU under the traditional model is tasked with the bulk of the data movement operations.

Traditional Stack	Approximate CPU Networking Cost	RDMA Stack
TCP/IP Processing	~40%	L3/L4 Offload
Intermediate Buffers	~20%	RDMA
Context Switch	~40%	OS Bypass

Table 1: Approximate 10Gbps processing costs for single sessions TCP transfer.

The CPU overhead related to networking is generally estimated at 1 Mhz of CPU required for each Mbps of un-accelerated network throughput. As an individual connection scales to 10Gbps, a 20 Ghz CPU is required to drive a single session of bi-directional link communication [9]. An emphasis is placed on un-accelerated in the example due to the fact that there are several stateless offload mechanisms that are typical in server L2-4 communication (Table 2). Stateless acceleration refers to hardware mechanisms whereby the state of the connection and protocol processing is not transferred to an external hardware entity for autonomous management on behalf of the software. Though stateless mechanisms can alleviate the CPU processing costs and measurably reduce latency, they preserve the existing limitations of traditional OS

Kernelspace stack processing previously highlighted (Table 1), and thus will not efficiently scale in the presence of speeds greater than 10Gbps.

Mechanism	Description and Purpose
Checksum Offload	Delegate checksum generation and decode of IP, TCP, and UDP packets to underlying NIC adapter ASIC such that the OS is relieved of performing the computations.
Large Send	Allows for the OS stack to pass up to 64K of data to the adapter as a single transmission. The adapter hardware will segment the data into MTU size frames based on the negotiated MSS value. Additionally, it will perform IP and TCP checksum generation for each packet. This allows for a single large send operation versus ~43 individual operations at MTU 1500, thus reducing CPU utilization
Large Receive	Coalesce and build multiple receive packets from the same TCP session (up to 64KB) in the hardware or software L2 processing such that a single data transfer operation is passed to the receiving Socket rather than ~43 individual operations. Similar benefits to Large Send.
Receive Side Scaling	Allows for parallelism on receive packet processing by utilizing multiple interrupts sources and hashing the 5-Tuple connection information to separate CPUs. Thus distributing the load for receive processing among multiple CPUs in the system. Allows for utilization of available bandwidth with multiple connections. Provides lower latency but does not accelerate single session communication.
Jumbo Frames	Non-standardized MTU size (Up to 9KB) to reduce the data movement operations from OS to the network. Similar benefit as Large Send and Large Receive.

Table 2: Ethernet stateless offload techniques.

2.1 CONTEXT SWITCHING

A contrast of the Sockets model (Fig 1.1) versus the RDMA model (Fig. 1.2) serves to show the location of performance bottlenecks. The context switch cost (1.A) consists of when a CPU must switch from Userspace processing to Kernelspace privileged processing and vice versa. During this operation the CPU must save off the Userspace application states, registers and pointers for restoration after the Kernelspace operations have completed. This cost is a necessity due to the presence of protocol processing in Kernelspace. Later sections demonstrate how the elimination of context

switching (1.A) is a consequence of the eliminations of intermediate buffer copies (1.B) and protocol processing (1.C).

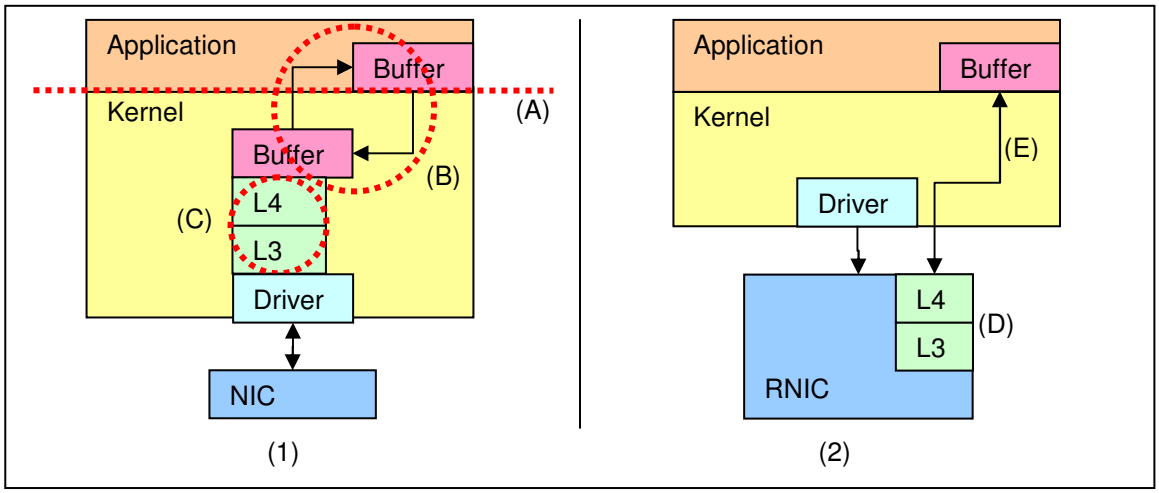

Figure 1: Sockets model vs. RDMA model.

2.2 INTERMEDIATE COPIES

In the traditional Sockets model, an application writes data into a Socket buffer that is copied (1.B) into Kernelspace for protocol processing. After protocol processing is complete, the kernel passes the application data to the underlying adapter for packet transmission via DMA. Clearly the reverse operation occurs on packet reception with similar costs in data copies. A major bottleneck associated with data copy operations is the fact that aside from the CPU cost, a memory write requires two memory bus operations, the first operation performs a read, and the second performs the write. The result is that for N bytes of data to be copied, it consumes N*2 the memory bandwidth. The performance impact is obviously relative to copy frequency and size, regardless, as line speed increases, the copy operations pose a limitation since the memory bus may become overloaded and Userspace applications are now at a disadvantage relative to

prioritized Kernelspace operations. Under the RDMA model, a mechanism is provided whereby an application can register its memory directly with the underlying NIC such that it is visible to both entities (1.E). A Userspace application can now initiate control and data movement operations between itself and the underlying NIC directly, without the mediation of the OS. From the NIC point of view, it has DMA read and write capability directly to the Userspace data, hence data transfers are not subject to intermediate copies by the OS. This practice is commonly referred to as Zero-Copy.

2.3 PROTOCOL OFFLOAD

The movement of protocol processing to the NIC (1.D) effectively means that the majority L3-4 processing is now resident on the adapter hardware via a combination of vendor specific implementation of hardware and firmware. Under the iWARP model, it is referred to as TCP Offload Engine (TOE). Similarly, under the RoCE model, the standard IB Transport and Network (L3-4) hardware offload mechanisms are carried forward. The protocol processing offload addresses connection timers, sequence number generation and tracking, reception acknowledgements (ACKs), and retransmissions. Hence the host CPU now has a reduction in preemptions from said operations, resulting in more free cycles for application usage.

The OS is circumvented as an active participant during data transfers but it is still the mediator for the creation and destruction of RDMA resources, analogous to Socket resource management. Similarly, the OS also performs the address resolution service, thus the acquisition of the destination MAC address is resolved through ARP for both cases since the RDMA functionality is effectively an extension and superset of the basic NIC functionality.

2.4 RESOURCES

Operations executed by an RDMA capable application are based on the creation of a direct and unique communication channel (Endpoint) with the underlying network adapter. The adapter may be referred to as a Host Channel Adapter (HCA), however with the convergence of both iWARP and RoCE on Ethernet the standard naming convention in the industry is referred to as RDMA capable NIC (RNIC). The communication channel established with the adapter is based on a set of standardized Verbs and Resources (Fig. 2) within the IB Architecture, and is also reused within the iWARP specification. Neither of the former RDMA implementations uses a defined API, but rather a set of operations which are encompassed by the Verbs representing an abstracted function that may be implemented as any combination of hardware, software or firmware [10]. The application, though free to do so, is unlikely to use the Verbs directly for communication across the network, as the intent of the Verbs is to provide a rich set of functionality over which a formal simplified API can be defined for application usage. Examples of such APIs are Sockets Direct Protocol (SDP) and Direct Access Programming Library (DAPL), neither of which exposes the Verbs interface to the application.

A key ingredient of any RDMA deployment is a rich set of APIs and interfaces to facilitate adoption and interoperability with legacy interfaces. In an effort to avoid proprietary and incompatible APIs, the creation of an Open Source and relatively portable RDMA API set was created by the Open Fabrics Alliance (OFA) [11] in the Open Fabrics Enterprise Distribution (OFED) package. OFED is the most widely deployed RDMA communication stack in both HPC and enterprise data centers. Currently, over 80% of the Top500 supercomputers in the world run the OFED stack.

Given its lead in market adoption, the majority of the RAS discussion on RDMA is focused on the OFED implementation of RDMA.

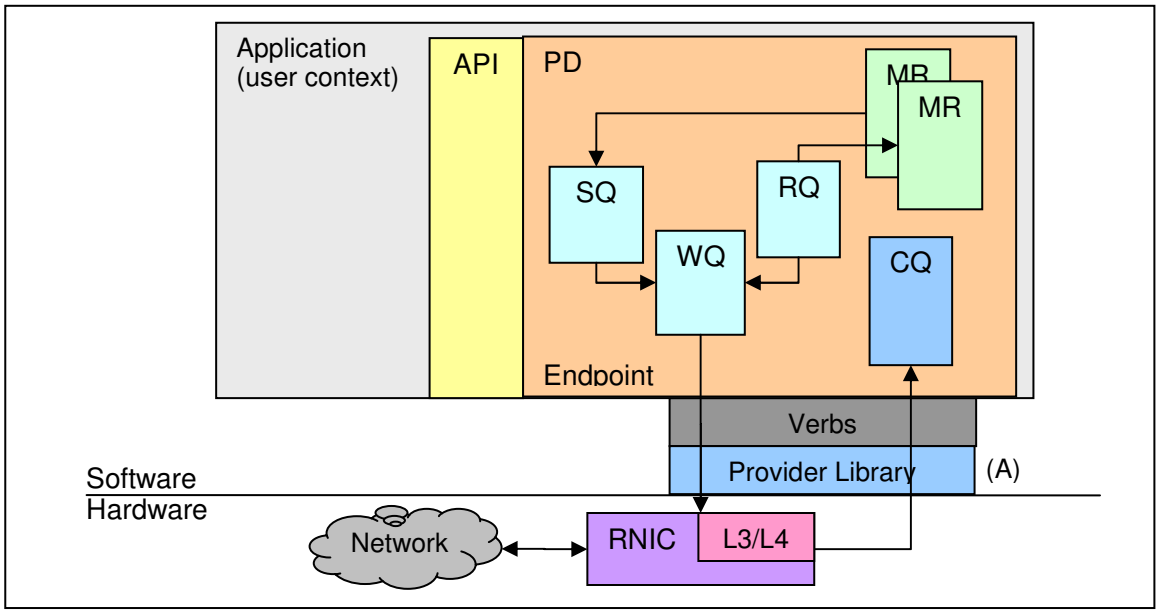

Figure 2: RDMA Endpoint resources

Endpoint resources are created via application communication with a Userspace IB capable Provider Library tied to the application instance (Fig 2 A). In order to create the resources, a series of Verb operations are executed by the specific API in use on behalf of the application. For any given Endpoint, a Protection Domain (PD) is used for security and isolation from other like resources. All of the user specific RDMA resources fall into this domain. The primary component of an Endpoint is a Queue Pair (QP) that is composed of a Send Queue (SQ), Receive Queue (RQ), and a Work Queue (WQ). The SQ is used to execute RDMA Read, RDMA Write, and RDMA Send operations into the WQ, which the RNIC uses to obtain the command and control operations. An RQ is used post buffers visible to the adapter and inform the adapter where incoming network data from the remote peer should be placed. The buffers posted into the RQ must be pre-

registered with the RNIC as a Memory Regions (MR) object. An MR is the representation of a contiguous memory area to be used by the application and DMA accessible by the RNIC. A Completion Queue (CQ) provides an asynchronous notification and completion mechanism to allow software to track when a specific event or error has occurred. The CQ notification mechanism is typically used to notify the application that an RDMA Write, RDMA Read or RDMA Send operation has completed.

The Verbs to create and destroy resources are generally synchronous in that the execution thread from the application traverses into Kernelspace and back. However the operations for data transfer such as RDMA Read, RDMA Write, RDMA Send and Receive are asynchronous in nature. This stands in contrast to the Sockets model, where an application typically has a thread listening for receive data on the Socket, hence the listening thread is blocked. In the RDMA model, data arrival or operation completion occurs as an event to be handled by the application via a purely asynchronous mechanism, similar to an interrupt handler.

2.5 OPERATIONS

RDMA supported connection types and operations differ between iWARP and RoCE (Table 3) [6]. The instantiation of a QP within the Endpoint determines the connection type and the desired reliability level. Four basic IB services compose RDMA transport functions. The Reliable Connection (RC) service is the standard connection establishment between Endpoints that, as the name implies, provides reliable communication. The Unreliable Datagram (UD) service is analogous to UPD where a message can be sent to an Endpoint without connection establishment. The Unreliable Connection (UC) is similar to RC in terms of connection but does not provide reliable message delivery. Lastly, the Reliable Datagram (RD) service allows for reliable

message delivery to an Endpoint without the connection establishment mechanism via the use of IB specific L1-2 network features. There is clearly a difference in the scope of RDMA support between iWARP and RoCE. The IB RDMA implementations are much richer in features, whereas iWARP only supports Reliable Connection (RC). This is to be expected as the underlying transport service behind iWARP is TCP, which is natively a connection based reliable transport.

Transport Function	Reliable Connection	Unreliable Connection	Reliable Datagram	Unreliable Datagram	iWARP Support
RDMA Send	Yes	Yes	Yes	Yes	Yes
RDMA Write	Yes	Yes	Yes	Yes	Yes
RDMA Read	Yes	Yes	Yes	Yes	Yes
Atomic Op	Optional	No	Optional	No	No
iWARP	Yes	No	No	No	

Table 3: Transport Function Supported for Specific Services

Of the supported RDMA operations, all except Atomic Operations are common with iWARP. Each of the operations, excluding atomics, allows for the transmission of up to 2^31 bytes of data prior to CQ notification. The Receive operation is only used to post buffers which allow the Endpoint to be the destination of RDMA Send operations, thus for practical purposes is it not considered a true RDMA operation.

The RDMA Send operation allows for sending data to the remote Endpoint without the need to exchange or negotiate the memory destination of the message. The data will simply be placed on the next available free buffer on the remote Endpoint. This operation is typically used to exchange memory information and control messages in preparation for the RDMA Read and RDMA Write operations.

RDMA Write operations allow for an Endpoint to write directly into the pre-negotiated memory destination of the remote Endpoint. As previously mentioned the memory control information is exchanged via RDMA Send operations between the two

Endpoints. Afterwards, the writing Endpoint will perform a contiguous write operation into the destination Endpoint memory. Upon completion, it is up to the sending Endpoint to determine if a separate RDMA Send operation is required to identify when the CQ notification is provided to the sending Endpoint, indicating the data transfer process has completed. The same notification mechanism is required for RDMA Read operation completion.

The RDMA Read operation is similar to the RDMA Write, but differs by the fact that when the memory destination parameter exchange between the Endpoints occurs, the initiating Endpoint memory is the target of the RDMA Write operation. Hence the remote Endpoint is actually performing an RDMA Write into the memory specified by the Endpoint initiating the RDMA Read operation.

2.6 RDMA COMPARISONS

In discussion of the differences and commonalities between iWARP and RoCE, several high level design points must be noted when describing RDMA and how each implementation fits into the enterprise and HPC environments.

In a nutshell, iWARP is a best fit for interoperability with existing switch infrastructure and WAN communications due to the fact that it leverages TCP/IP as the protocol offload mechanism. This becomes important with the rise in demand for large data transfers via streaming data such as video, music, and torrents. Though the aforementioned can run over UDP, current implementations in the market stream this data over TCP. With the combination of Client based software iWARP [12] and Server based hardware iWARP, it is possible to create a Server efficient mechanism for high speed media dissemination over the internet [13] which cannot be accomplished with RoCE due to the aforementioned routing and interoperability issues with IB.

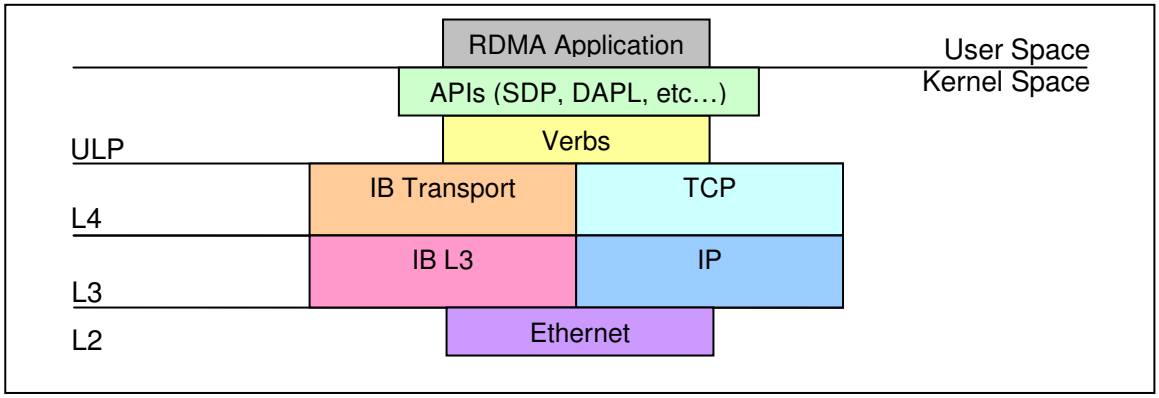

Figure 3: and iWARP over Ethernet

iWARP packets are virtually indistinguishable from standard TCP/IP packets on the network (Fig. 3) [15] unless deep packet inspection beyond L4 is performed. This interoperability however comes at a cost in terms of complexity. The iWARP protocols require special standards workarounds to convert a Streams based communication mechanism into a Message based mechanism. Thus the application of the Marker PDU Alignment (MPA) [14] protocol is required. Additionally, the enablement of TCP no-delay options avoids the coalescing of data for transmissions typical of Streams protocols. Effectively forcing the protocol offload mechanisms to encompass a Streams to Message conversion in hardware which is of more complexity than a native Message based protocol such as the IB Transport. It is noted that iWARP can also leverage SCTP which is also Message based, however, to date there is no wide market adoption of iWARP SCTP, which is expected as there are few major adoptions of standard SCTP, therefore SCTP is omitted as a viable alternative to RoCE.

From a RoCE point of view, the fact that it utilizes a native Message based transport specifically engineered for efficiency and HPC, translates to higher bandwidth processing capabilities and lower latency by design; RoCE therefore outperforms iWARP

in cluster based IPC, and distributed systems. With its adoption over Ethernet, RoCE addresses the needs of most enterprise data centers since the routing aspects can in theory be resolved with Ethernet to IB bridges. The market for such devices is quite niche, thus there is currently no wide adoption for the approach.

Lastly, the aforementioned discussion touches upon the key points and concepts relevant as background information for implementing RAS on RDMA implementations. An exhausting review of RDMA technical details and comparisons falls outside of the intended scope, thus the reader is encouraged to review the references further technical information.

3.0 MOTIVATION

With the adoption of RDMA over Ethernet, the traditional NIC software functionality and requirements has been extended, hence RDMA support becomes a superset of the basic server NIC functionality. In order to understand the motivation for RAS application over RDMA enabled NICs, it is best to describe the underlying software NIC framework over which said functionality will be deployed.

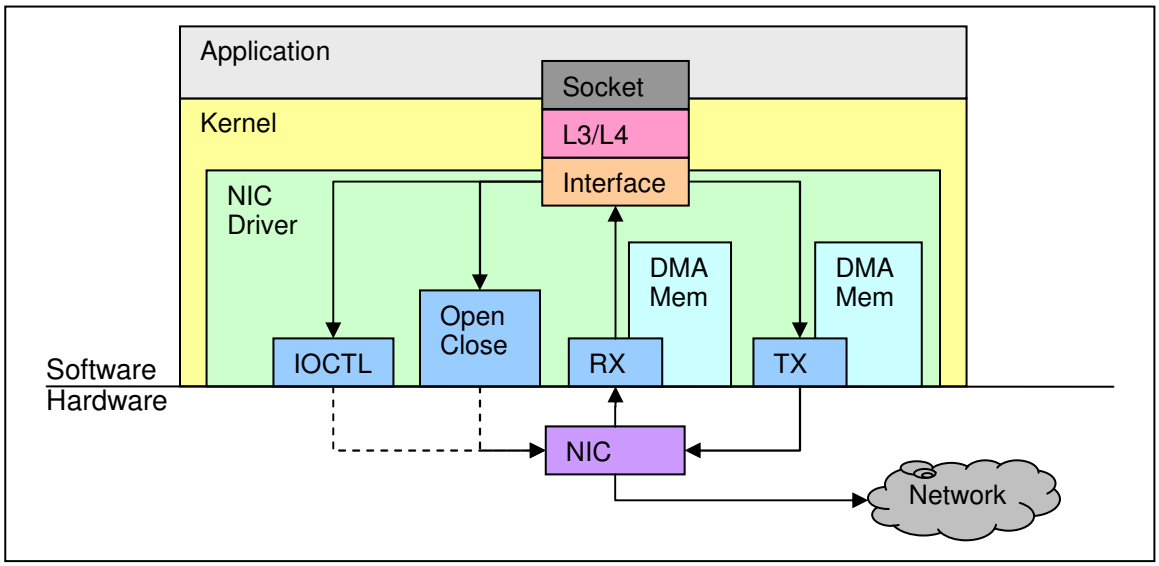

Figure 4: Generic NIC driver software Architecture

Typical software architectures for an L2 NICs (Fig. 4) are composed of 5 driver entry points, excluding configuration. Execution of the NIC entry points is simplistic in nature since the Open and Close operations occur only at setup or teardown of the interface. The IO Control (IOCTL) entry point addresses special and rare operation cases such as multicast address registrations and statistics gathering; it is not considered a hotpath. A hotpath is defined as an execution path that is sensitive to performance, hence requiring additional design considerations during implementation. The remaining

Transmit (TX) and Receive (RX) entry points are the hotpaths. In a multiprocessor system there may be multiple instances of Transmit and Receive pairs to leverage parallelism, also known as Receive Size Scaling (RSS) [16].

3.1 RNIC EXTENSIONS

With RDMA, the standard NIC driver is extended in functionality (Fig. 5) via the addition of a Provider driver interface (Fig. 5.B), which extends the existing 5 NIC entry points with more than 40 new entry points. The Provider driver interface contains the bulk of the software logic to handle RDMA resource create, destroy, and management operations. RDMA entry points differ from the standard NIC entry points in three ways.

Firstly, the generic NIC interface allocates and destroys all resources at Open, Close, and Error time only, whereas a Provider interface uses dynamic resource management. RDMA Endpoint resources are granular and dynamic by design, which is to be expected as they are representative of unique connections. Endpoints are analogous to the dynamic and granular nature of TCP connection establishment, hence a Provider driver interface must allow for concurrency and scaling, not typical of a generic NIC driver, during Endpoint allocation and de-allocation.

Secondly, RDMA resources allow for unprecedented scale in the number of resources. A typical iWARP RNIC allows for roughly 64K CQs, QPs, PDs, and MRs resources. Hence, software RNIC architectures must scale from the typical 5 NIC resources up to 256K Endpoint component resource instances. Similarly, a larger resources scaling requirement is necessary for RoCE, since the maximum supported resources are 1.2M for each of the previously mentioned types.

Thirdly, the Provider interface also contains a Userspace Provider Library component (Fig 5.A) which is dynamically loaded by the API or user application

instantiation. The Provider Library allows for the execution of RDMA Verbs from Userspace, and transitions to Kernelspace on the executing thread for resource allocate and destroy operations. After the Endpoint resources are created, and during data exchange between Endpoints over said connection, the Verbs interface allows for operation between the API (application) and RNIC directly through the Verbs interface and bypassing the OS.

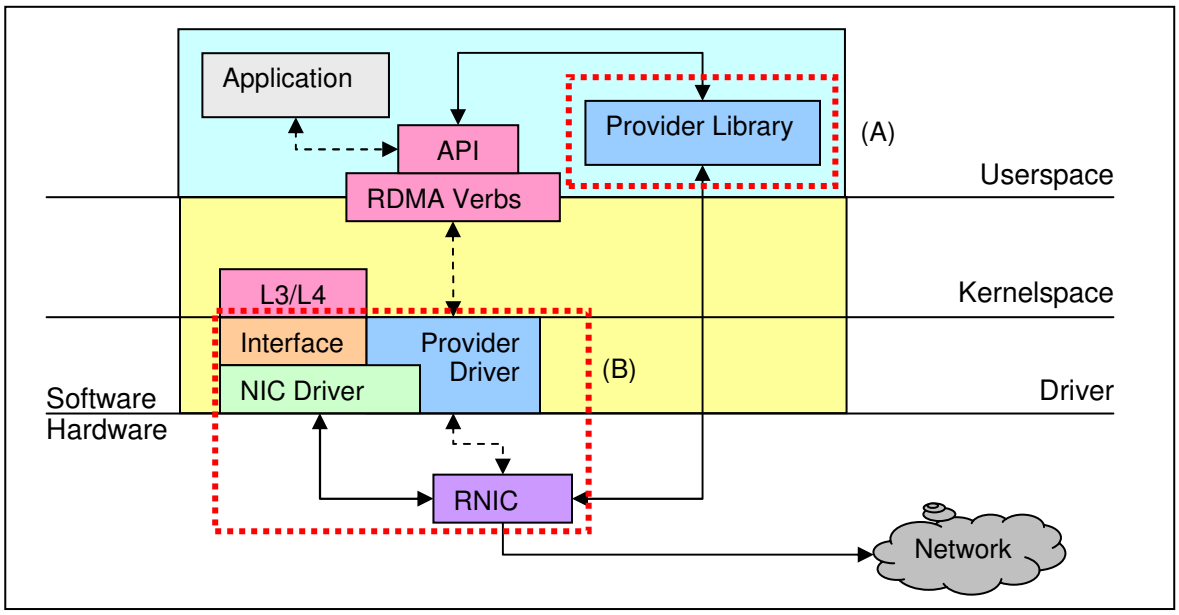

Figure 5: Generic RNIC driver software Architecture with RDMA Extensions

Lastly, as is to be expected, the Userspace Provider Library is vendor specific, just as the NIC and Provider drivers; hence any two separate adapter types will necessarily contain unique software versions of their respective Provider Library.

3.2 RAS LIMITATIONS

Given the comparison between NIC and RNIC software architectures, a detailing of the challenges from an enterprise RAS point of view is necessary. The OFED implementation of RDMA and the associated certified vendor specific drivers that ship with the installation image provide a measured level of reliability via certification testing. Though important and essential for deployment, it provides differing serviceability approaches within the separate modules, similar to what is provided via any Open Source software. The inconsistency is most evident in the vendor specific RNIC drivers where any combination of debug levels, Syslogs, and error messages are used as the primary vehicle. Obviously, the extent to which each is implemented is specific to the OS and vendor implementation. As an example, Linux RAS is mostly based on Syslog and printf as the sole kernel vehicles. Regardless, this approach tends to foster trial and error types of root cause determination that is unacceptable in an enterprise data center.

Another challenge inherent with RDMA is the fact that the protocol processing has been offloaded to the underlying RNIC. The RNIC now contains the processing engines and states for L3-4, which are not accessible by the OS. Traditionally, these layers are processed in Kernelspace where the data is globally visible, such that when an unexpected operation occurs, all modules can be inspected for traces, states, and errors to determine root cause. In having the L3-4 processing in hardware, its error data is no longer readily available.

To further complicate this matter, the L3-4 implementation of protocol processing in hardware is vendor specific and will differ across RNIC adapters from different vendors. In a stateless offload environment, a single software based L3-4 stack is used across multiple NICs, hence simplicity via singularity is achieved. As the L3-4 offload processing moves into vendor specific RNICs, it becomes the equivalent of having both

multiple instances of L3-4 stacks, and differing implementations of each. Resulting in the possibility that a protocol problem addressed by one adapter will not necessarily be addresses in another. Just as an OS is typically subject to errata and updates to address L3-4 protocol processing issues, the underlying RNIC must now be managed in the same manner. The requirement poses a management problem since RNIC vendors can have differing approaches and rates of errata application, thus the role of systems management has been complicated.

Concurrency in an RDMA environment must be revised since the simplistic serialization mechanisms used for a NIC cannot scale to meet the needs of dynamic resources. The parallelism and serialization of Provider entry points falls under the scope of the device driver rather than the interface to the driver. Critical Paths must be tracked and monitored in more detail such as to avoid conditions where an RNIC Fatal Error or Close operation causes invalid operations or memory accesses.

3.3 SOFTWARE DEVELOPMENT AND SUPPORT

Software maintenance is widely known to be the single largest cost of software development expense. Deployments of RDMA in data center environments must be enhanced to allow for ease of Serviceability such that a support team can obtain forensics data in the face of error, with minimal trial and error cycles. Enterprise RAS features such as granular level tracing, system level dump, software component level dumps, resource operation history, and detailed error logging can be implemented in RDMA environments. RAS features may carry a cost that can result a measurable performance difference between a non-RAS RDMA implementation and a RAS enabled implementation. Such architectural tradeoffs are the focus of the discussion at hand. The problem has been recognized by Commercial OS vendors and the US Department of

Defense with concerns about the cost of performance at the expense of human effort [4]. The RAS enhancements for RDMA communications will leverage key architectural improvements from both Academia and enterprise OS vendors to enable granular and dynamic levels of RAS.

4.0 RELATED WORK

A software product may work correctly without a robust RAS architecture; however as it is extended, additional testing and verification must take place. This is commonly accomplished via static and dynamic analysis source code analysis, Unit Testing (UT), Functional Verification Testing (FVT), and Integration Systems Testing (IST), of the source code in place of RAS. While analysis certainly serves to catch coding errors, it does not allow for catching architectural errors. UT, FVT, and IST are attempts to cover all hardware and software interactions; however it can rarely address the myriad of external error conditions triggering incorrect software or hardware operation. For these reasons a robust RAS architecture serves as an essential mechanism to provide recovery and data acquisition in the face of unexpected operating error and conditions.

A review of the main RAS topics with existing examples is performed, along with a highlight of the benefits and deficiencies of each in order to establish the groundwork for the RDMA RAS architecture. Recognizing the OFED implementation of RDMA is not the sole implementation in the market, it is noted that some or similar features discussed may be present in proprietary RDMA implementations from vendors such as IBM, HP, Sun, Microsoft, etc. The fact that said implementations are closed source translates into an inability to review in detail their supported RAS capabilities, thus the general approach is to provide a generic RAS architecture and development guidelines to explicitly address deficiencies which can be implemented or ported to any of the aforementioned OSes.

4.1 RELIABILITY

Software reliability is based on the likelihood of operation without failures for a predetermined amount of time and in a predefined environment. In the case of RDMA, both software and hardware reliability must be considered due to the stateful offload capabilities. Software components should make all possible attempts to overcome and continue operation in the advent of both software and hardware failures. Hence this is effectively the implementation of fault tolerance in a combination of both software and hardware. It is clear that fault tolerance can be achieved via distributed computing and clustering at the systems level, however these are coarse solutions and have larger scope than simply recovering from an RNIC error. A focus is placed on aggregation techniques that allow for the abstraction of the underlying RNICs into a single interface to the application or OS user.

4.1.1 Aggregation

Two well known Ethernet reliability concepts are IEEE 802.1AX [17] and NIC Teamming [18], both of which provide the aggregation of multiple physical links as a single interface to the OS, thus allowing for uninterrupted and fault tolerant communications in the event of NIC errors or link loss. The approach leverages the fact that the L3-4 processing is contained within the OS (Fig. 6.A), such that the state of a given TCP connection on NIC(x), can be easily transitioned to NIC(y) with minimal loss or retransmit operations. The solution however, does not meet the needs of RDMA enabled communication. As mentioned previously, the offloaded L3-4 processing is tied to a specific RNIC (Fig 6.B). Hence the Endpoint connection context and state are tied to a specific RNIC, and cannot be shared among RNICs. To date there are no known public

or proprietary adapter connection state extract and insert mechanisms for stateful offload that would allow for such action.

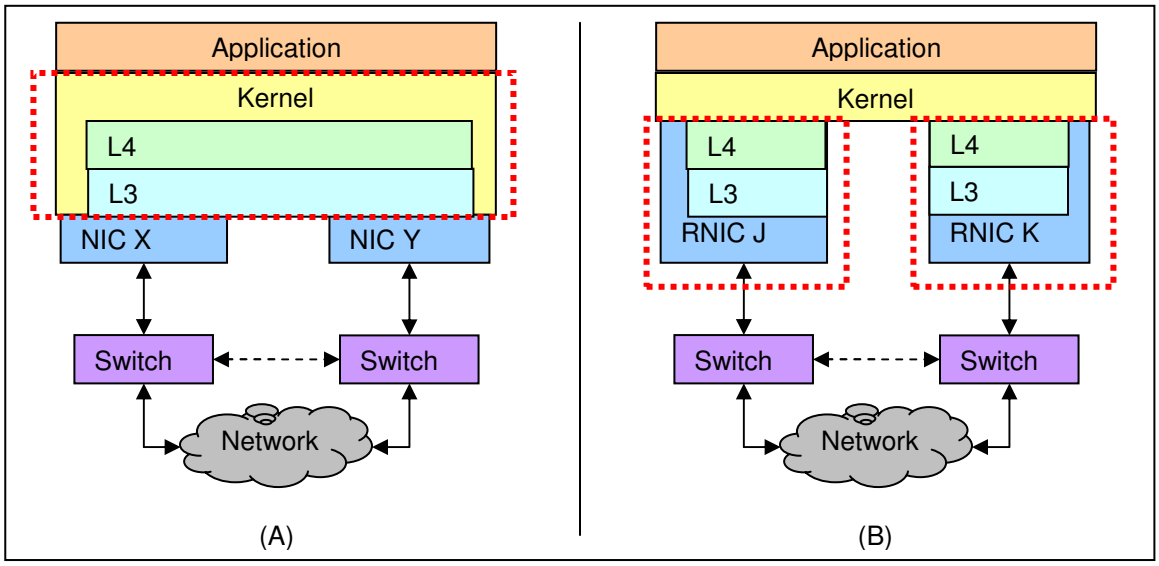

Figure 6: Native OS Stack and aggregation challenges due to L3-4 Protocol Offload.

The IB architecture specification defines a feature known as Automatic Path Migration [6], which allows for the specification of primary and alternate physical network paths for a connection. Path Migration allows for a given connection to be registered with multiple ports on the same adapter and subsequently separate network paths. The ports share the same adapter L3-4 contexts. At Endpoint creation time, primary and secondary paths contexts are specified. If the primary path and underlying network port loses its link, or connectivity due to downstream switch outage, then a transition to the secondary path is automatically triggered. Unfortunately, this feature is not supported under the RoCE implementation as it leverages IB L2 features not available in Ethernet. Furthermore, it requires the registration of the Endpoint with both

ports on the same RNIC, therefore it does not solve the Single Point of Failure (SPoF) scenario from the adapter point of view as is addressed by NIC aggregation.

Application or user level aggregation is possible; however it places the burden of resource management and failure on the Verbs user. Though the approach achieves the desired elimination of SPoF for RDMA, it necessarily means that each Verb user will need to implement a proprietary mechanism. Given the OFED stack provides a wealth of existing APIs to abstract RDMA functionality, they can be extended to also abstract aggregation functionality, such that the application does not have to account for it. Thus it provides a more desirable approach to avoiding both duplicity and complexity.

4.2 AVAILABILITY

Availability refers to the capability of a system to operate in a normal or degraded state in the event of unexpected operations; commonly referred to as Uptime. Enterprise class systems designs strive to achieve the ideal 99.999% uptime via any combination of RAS features. A discuss of existing mechanisms to prevent or circumvent system outages is provided.

4.2.1 Device Recovery

The typical RNIC adapter is PCI attached and subject to hardware errors uncorrectable by software. Depending on the error type and timing, it is possible for bad data to be placed on the PCI bus and a resulting machine check interrupt occurs. This event typically results in a system crash and requires a system restart to recover. In large enterprise class servers a common feature known as PCI Enhanced Error Recovery (EEH) [19] is available to reduce the likelihood of system wide outages. EEH provides a

means where each PCI slot is dedicated PCI switch, therefore a PCI bus error can be isolated to slot domain that is not possible under s multi-PCI slot to switch scenario. When the PCI error is detected, the slot is frozen resulting in no further IO operations. Software is responsible for checking for the freeze condition. If the software driver detects such condition, then a graceful software error exit occurs. The adapter is then reset and reinitialized to continue normal operations. Said actions effectively recover the adapter and circumventing a permanent PCI error which would have resulted in system crash or reboot. The entire process typically completes in a few seconds.

The major caveat of this approach is that all RDMA connections must be closed and restarted when the RNIC is reinitialized. A solution such as Path Migration does not resolve this problem since both primary and backup paths would be on the same RNIC and subject to the same freeze condition. A further complication to RDMA is the fact that Endpoint resources in Userspace have a direct channel to the RNIC via the Endpoint resources, and thus their software cleanup operations must complete prior to allowing for the RNIC hardware re-initialization operation to start. This allows for a condition where if an application does not perform proper Endpoint cleanup, it blocks the recovery of the RNIC for an extended period of time, resulting in extended network outages. A mechanism to address this issue by tracking the Endpoint process IDs is provided within the RDMA RAS architecture, furthermore it allows for automatic or manual removal of the Endpoint resources.

4.2.2 Memory

Memory is considered a crucial component of systems availability and the subject of years of RAS research. Clearly an exhaustive discussion of all existing mechanism which apply to RDMA communications is not feasible. For the purpose of RDMA,

emphasis is placed on existing techniques that allow for ease of implementation when extending the functionality from the NIC software architecture to the RNIC software architecture. A focus is placed on memory and resource leaks, memory corruption, and memory serviceability.

Software memory management, when applied to RDMA, has the peculiarity of allowing for instrumentation with minimal performance impact when compared to Sockets communication. This is due to the fact that when an Endpoint is created under the RDMA model, the operations require the creation of multiple resources (PD, QP, CQ, and MRs), and registrations of each between the OS and the underlying RNIC. A typical connection setup and first time to byte comparison shows that TCP takes roughly 0.1ms whereas RDMA takes roughly 202ms [20]. hence RDMA tends to favor persistent long lived connections. Given these characteristics, memory instrumentation is unlikely to add significant overhead to RDMA performance.

4.2.2 Memory Leaks

Most modern operating systems have low focus on RAS features to pinpoint the source of a memory leak. It is normally left up to the application developer to perform graceful cleanup. In most cases the OS will prevent memory leaks for Userspace, however in Kernelspace the operation is somewhat more complex. Garbage collection is not typical, hence kernel operations such as the low level RNIC driver and Provider are forced to track each resource allocated and exit gracefully during Close or Error. The number of resources can scale into the millions, thus the resource tracking mechanisms can become complex. Given the peculiarity of RDMA resources in allowing for the instrumentation of memory allocations, a mechanism is required to track the memory allocated and additional details such as resource, user, and source code data. This differs

from existing efforts in that it occurs at runtime, is generic, and avoids complex algorithms [21] [22] which do not provide the descriptive data to clearly show the root cause.

4.2.2 Memory Corruption

Applications residing within the same OS instance are typically not subject to overwriting each others memory, however the OS kernel is a privileged entity and able to perform incorrect write operations to both Userspace and Kernelspace. A technique or mechanism is necessary for the application to detect and recover from said memory corruption on its critical data structures. A proprietary hardware mechanism to detect such a condition exists via Storage Keys [23] which allow for the hardware protection of memory associated with a process. The Keys are a set of shared resources and requires the multiplexing of multiple processes into the same keys. Furthermore is does not address the case of an application corrupting its own memory. Typically, software type corruption is characterized by byte alignment since the smallest granularity is a word, whereas hardware corruption tends to be bit based and associated to DMA mapped memory only. A technique to probabilistically detect and prevent the use of corrupted data within Userspace and Kernelspace software is necessary such that the RNIC driver and Provider Library are protected from external memory overlay errors.

4.3 SERVICEABILITY

The serviceability aspects of RDMA revolve around the expectation that the operations, software, and hardware resources composing the Endpoints and network communications are accessible, verifiable, and traceable such as to facilitate diagnosis of

problems or issues resulting from incorrect operation or errors. In the advent of said events, a system is typically taken out of the production environment (offline) when possible, to diagnose and or recreate the problem. This action has both a monetary and business cost for both the customer and the service Provider. The shorter the Mean-Time-To-Repair (MTTR), the sooner the system can continue operation at the expected Reliability and Availability levels. From the service provider point of view, the sooner the diagnosis and repair are completed the lower the software service cost incurred.

4.3.1 Trace

Tracing is the primary debug vehicle for both Userspace and Kernelspace software development. A System Log (syslog) also allows for providing equivalent functionality. Taking the Linux OFED stack and RDMA as examples, it is clear that the tracing capability is quite coarse. The driver, OFED stack and Provider components do not have a defined mechanism to implement dynamically tunable traces and levels without requiring a recompile of the source code. Clearly this approach is tedious and requires manual steps to problem recreation. A dynamic and tunable tracing mechanism for both Userspace and Kernelspace based on the Log4J concept [24] is preferred. Granular tracing allows the user to tune the level of tracing on a given path and balance the performance impact, thus conditions where tracing or debug can mask off timing and performance sensitive issues can be addressed in a more dynamic fashion.

4.3.2 Debugger

Another commonly used debug tool is the system debugger. During a system crash or investigation into incorrect behavior, the debugger provides access to structures

in memory and the historical trace data. From the Kernelspace point of view, the debugger can be used to obtain a global view of all of the memory and variables, in host memory at the time of crash, associated with Endpoints and any software based RDMA resource. Furthermore, the values in memory can also be modified such that execution is instrumented in a live system. The major caveat of the debugger is the cognitive complexity inherent in its use (Fig. 7). The developer investigating an issue has to perform a series of mathematical steps to acquire and translate memory locations into a human readable format. A mechanism to automate and reduce this process is essential in achieving a low MTTR. Clearly the use of ASCII markers and decoders can address this issue and reduce the cognitive complexity inherent in raw debugger commands.

Address:	64bit word	64bit word	ASCII Decode
F1000A0018DE0060:	00000000FFE7E000	0000000080000000	.....
F1000A0018DE0070:	00000000FFE7F000	0000000090000382	.....
F1000A0018DE0080:	0000000090000382	0000000000008C03	.....
F1000A0018DE0090:	0000000100000001	0000000100000118	.....
F1000A0018DE00A0:	0000022500000226	0000022700000228	...%...&...'...(
F1000A0018DE00B0:	000002290000022A	0000022B0000022C	...)...*...+...,
F1000A0018DE00C0:	0000000800000002	0000000000000000	.....

Figure 7: Typical debugger output

4.3.3 Error Log

An OS error log [25] provides a means whereby critical events and informational messages about the system state can be clearly communicated to the user. This facility is commonly used to perform automated service callout, notify the system administrator, and optionally communicate to the system vendor both software and hardware errors which need immediate attention. The robustness and capacity of the logging facility is dependent on the specific OS implementation. Implementations such as the AIX OS

error log provides facilities to capture and display to the system administrator sufficient data to allow for corrective actions or potential workarounds to non-automatable recovery solutions. This concept is leveraged within the RDMA RAS architecture for cases where the RNIC device cannot be, closed, reset, or recovered.

4.3.3 Statistics

During Endpoint connectivity issues, performance degradations, or general system problems, a robust and granular statistics framework serves as a means to answer many of the questions and concerns that would traditionally be the subject of debugger analysis. A basic statistics framework will obtain and display data about the connection and states, but usually limiting the diagnostic functionality. In an RDMA RAS architecture, the traditional statistics framework must account for the myriad of rare errors and conditions for ease of serviceability purposes.

A traditional TCP stack in kernel statistics approach does not typically account each socket resource nor for L2-4 as a single entity. It is therefore typical for statistics programs to be bounded to a specific network layer such as L3 or L4. In the RDMA case, since the L3-4 processing has been offloaded to the adapter, any RDMA specific statistics framework necessarily has to encompass L1-4. In accounting for said layer information, the remaining issue is the identification and association of the (PD, MR, CQ, QP) resources that compose individual Endpoints. Hence statistics in the RDMA case are significantly more complex than both the traditional NIC and the traditional communication stack in Kernelspace approach.

4.3.4 System Dump

System dump is the most disruptive and typically the serviceability action of last result. A system dump [26] is configured by the system administrator such that when a fatal error event occurs within the OS, all memory contents are written to either a local disk or remote network dump device. This allows for the post mortem analysis of all system memory and states. For a large enterprise server, the dump will necessarily be of the same size as the system memory in use for the OS which initiated the dump, hence it is not uncommon to perform post mortem analysis on >10GB of memory to find root cause. This technique requires the application of the debugger to interpret the memory contents. A system dump debugging approach is coarse and causes excessively long system outages. It is therefore necessary to provide a mechanism whereby a very granular component dump mechanism [27] can be applied which obtains only the memory snapshot of interest, thus avoids the full system outage time resulting from a normal system dump.

5.0 ARCHITECTURE

Given the aforementioned limitations and inefficiencies in the existing combined approaches to RAS application in the face of RDMA, the individual components within the architecture are order and discussed by groups. Several of the proposed RAS components are interrelated (Fig. 8) and build upon each other. Though some components within the architecture are not specific to RDMA operations, they are currently unimplemented in current existing version of OFED or Linux RDMA drivers; hence their adoption would serve to improve the RDMA RAS capabilities.

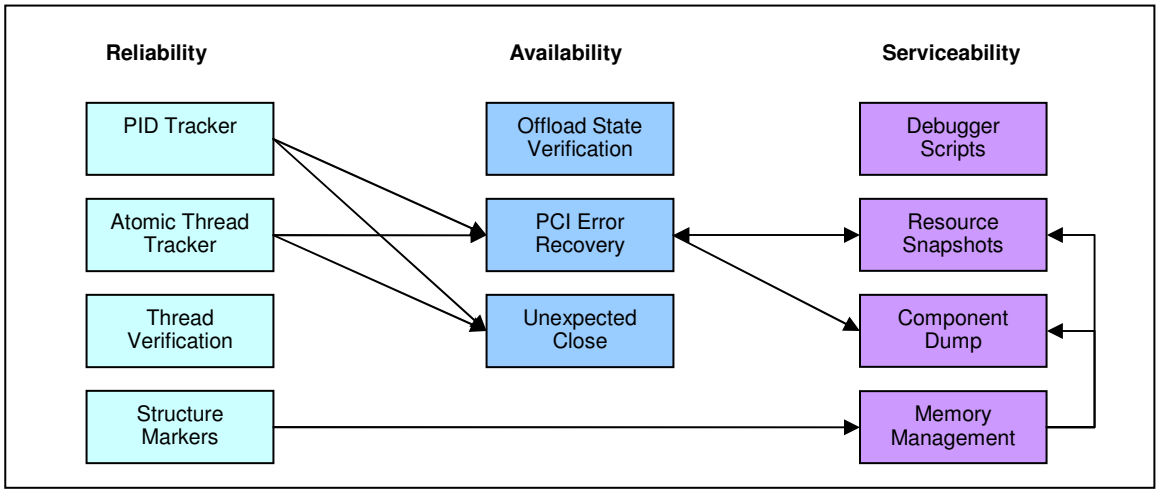

Figure 8: RAS component relationship

5.1 RELIABILITY

In order to enhance reliability, a set of guidelines and concepts intended to improve self-checking and self-recovery in the presence of internal en external errors is necessary. Three key issues must be addressed regarding reliability within the architecture. Firstly, the act of tracking user process IDs such that a rouge user does not cause an RNIC removal or recovery hang by omitting the de-allocation of Endpoint

resources registered with the adapter. Secondly, a critical path protection mechanism to atomically detect the presence of executing threads. Thirdly, the probabilistic detection of host memory corruption via the use of structure markers. Lastly, a simplistic mechanism to detect the API thread level violations which lead to system assert.

5.1.1 Process ID Tracking

Within the OS every process has a unique identifier known as the PID, this allows for the system to perform the management, prioritization, security, and resource association of all threads executing within the system. Under a traditional stack design, the underlying L2 NIC driver is oblivious to the upper layer users. The L2 RNIC software driver receives a single Open call and does not need to track each individual user since the stack and L3-4 protocol processing occurs above and abstracts the individual users.

In the case of an RDMA connection, the L2 driver has to be user-aware, since each user with an established connection or Endpoint to the adapter effectively has resources (PD, MR, CQ, and QP) associated with the adapter. Failure by a user to de-allocate RNIC registered Endpoint resources (Fig. 9) will result in blocking the recovery from a PCI bus Error or the removal of the device during a Close operation.

The failure to remove resources condition is typical in the early development phases, bring-up, and integration phases of heavily multithreaded applications. Since each resource is now actively tracked by both the Userspace Provider Library and the Kernelspace Provider Driver with a PID association, the driver can extract this information from the thread and build a table or list of active PIDs. Additionally, the driver can extract the human readable thread name that identifies the parent process. With the PID at hand, a verbose and detailed error log entry can be generated which

contains PID of the rogue user, and the human readable application name owning the process.

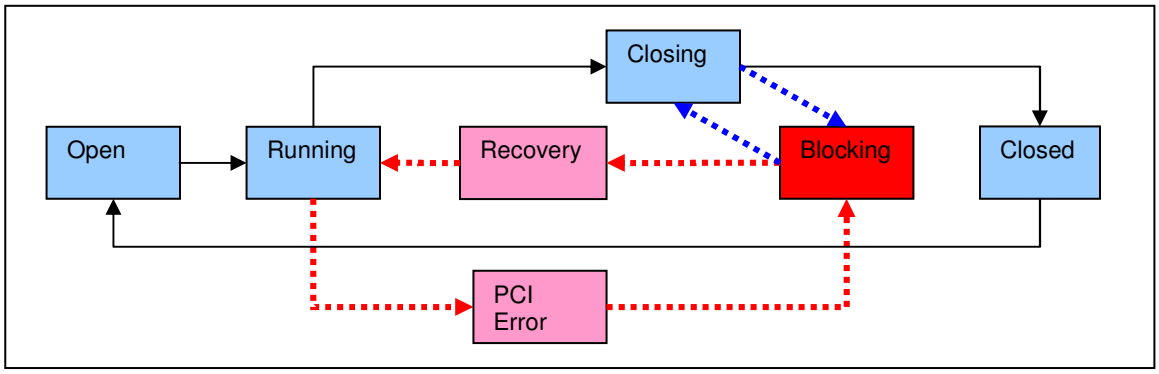

Figure 9: Driver paths blocked by non-deallocated resources

By tracking and reporting the PIDs by number and application name, it is possible to provide a clear and concise mechanism of low cognitive complexity to allow a user or customer support representative to detect and correct the misbehaving application by manually or automatically killing the process. Thus providing a means to ensure the interface works as expected in the presence of the inability to remove the device due to outstanding Endpoint resource references. This feature circumvents system restart actions and avoids extended debug cycles in identifying the source of a hung RNIC.

5.1.2 Atomic Thread Tracking

Given the number of driver entry points and the level of thread concurrency on said entry points, a non-blocking mechanism to detect the presence of threads executing in the critical path is required (Fig. 10). Under a traditional NIC driver the control (non-performance) paths are protected by locks and the data (performance) paths are protected by atomic flags to allow for maximum concurrency. As the NIC functionality is

extended to support RDMA, the level of concurrency from the theoretical 64K+ users necessarily forces all entry point paths to be atomic.

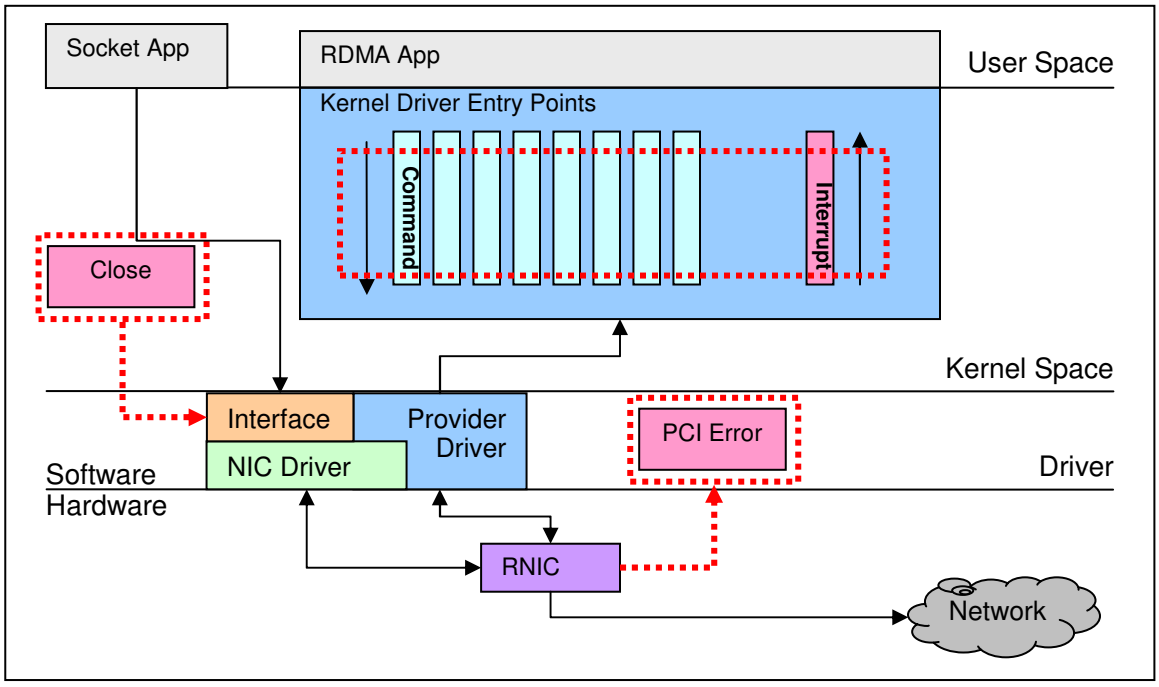

Figure 10: Non-Blocking kernel critical paths and dependent operations

The primary goal is to ensure that an unexpected Close or Error event that triggers a software or hardware recovery action yields until all users have successfully cleaned up their resources. A secondary effect of this approach is that it allows for validation of the paths, during software design and development, to ensure there are no timing holes. The Close or PCI Error recovery operations must yield until the currently running threads have exited the critical section and have quieced, such that the L2 driver can perform a graceful software and hardware Error recovery or Close.

Clearly this feature has interactions with the aforementioned PID tracking feature. To clarify, the PID tracker provides the identification of the thread to kill for rogue users,

whereas the Atomic Thread Tracker provides a means to ensure the critical paths are protected.

5.1.3 Thread Level Verification

Userspace threads within an OS run at process level from a kernel point of view. Thought there may be manual prioritizations via the nice command in Unix based systems, from the Kernelspace point of view it is categorized as process level. Within the kernel processes can execute at either process level or interrupt level. There are several kernel services which cannot execute at interrupt level, namely operations such as thread sleep, memory translation, pinning services, and coarse timers are examples. Linux based OSes contain just these two kernel thread levels, whereas advanced Unix based systems such as Solaris, HP-UX, and AIX contains more granular kernel thread levels.

Given the brief background on process levels, the RDMA Verbs and APIs which exploit them will have clearly defined input parameters which can be subjected to input parameter sanity checks. In HPC RDMA applications, this is typically not carried out for the sake of performance and the quest for the absolute lowest possible communication latency. The standard practice is to trust the input parameters and flush out problems via testing; clearly at odds with data center RAS practices. Hence input parameter checking typically does not check for incorrect thread level. API violations due to thread level are typically undetected until a kernel service which is not allowed to execute at interrupt level fails; usually an unintended consequence of improper design. Upon said event, a debug and analysis process of the crash scenario is required to identify root cause.

A simplistic guideline with minimal overhead is proposed which can be used to obtain the thread level regardless of the OS in use (Fig. 11). The mechanism is most applicable to kernel level RDMA Verbs access where varying priority levels are present.

When this condition occurs, one can deterministically detect this violation and perform a graceful exit, thus preventing system crash. In the absence of a native OS service to obtain the thread level, the routine can as a software service to the binary of interest.

```
inline int32_t
thread_level_check() {
    uint32_t priority;

    /* Transition to highest thread level, saving off the original level, followed
     * by an immediate restore of the original thread level */
    priority = disable_interrupts();
    enable_interrupts(priority);
    if (EXPECTED_THREAD_LEVEL != priority) {
        /* Unexpected thread level, assuming 0 is not a valid priority */
        execute_additional_debug_actions_here();
        return priority;
    }

    return 0;
}
```

Figure 11: Thread level sanity checker

5.1.4 Structure Markers

Within the kernel, memory is globally accessible. Kernel software has the capability to modify the memory contents of other kernel components, which opens the door to programming errors causing data integrity problems. A kernel driver can also access Userspace memory by performing a cross memory map operations, hence incorrect kernel references to Userspace can also lead similar data integrity problems in Userspace. The condition is not unlikely since these types of errors are difficult to recreate and require significant time and manual effort in memory debug. Advanced server systems such as IBM POWER contain a hardware based memory overlay protection mechanism to detect this condition via hardware Storage Protection Keys [28]. Storage Keys operate by association of a key value to a subset of the system memory. An

execution threads must provide the key at the software module entry points to authenticate access to the memory. If an incorrect key is provided then loads and stores to memory result in a program assert.

As with any hardware based mechanism, the quantity of physical resources become the bottleneck when scaling is desired. Though the storage keys mechanism provide a means to protect the system, it cannot scale linearly with the number of RDMA resources, and thus results in key multiplexing which leaves a software module exposed to memory overlay errors by other modules within the same storage key group. An alternative mechanism to provide similar memory overlay problems is the use of defensive programming techniques such as structure markers (Fig. 12). Markers allow for rudimentary sanity checking of software structures prior to use, thus avoiding operations on invalid data.

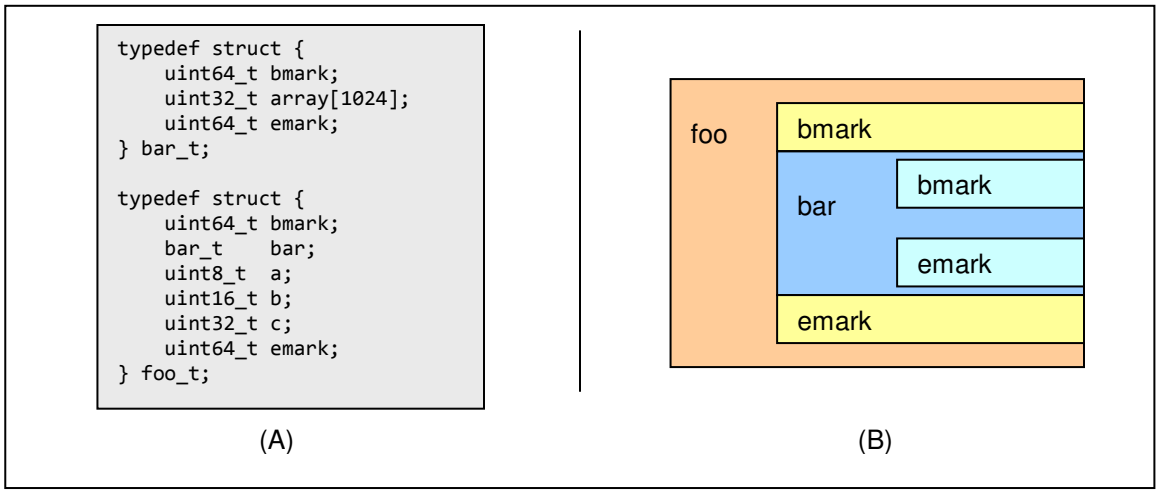

Figure 12: Structure marker definition and conceptual memory layout

At structure allocation or initialization time, a unique and well known value is inserted in the start and end of the structure. Upon operations on said structure, the driver simply validates the markers with the expected values. If the markers are not valid then it

is safe to assume that a data integrity problem has been detected. The thread may then circumvent both the accesses to compromised memory and pointer dereferences.

Clearly this method is most effective in detecting contiguous unintended overwrite operations in memory, thus its effectiveness is probabilistic and relative to the size between the start and end markers within the structure. The addition of markers will increase the memory footprint of the driver, however given that RDMA resources are relatively large in terms of memory usage when compared to TCP/IP communications. It is sage to assume that the size delta in memory footprint is negligible. Increases in footprint are common as in the case when line speed increases and performance is of priority. Structure markers are a basic mechanism exploited by other components within the RAS framework..

5.2 AVAILABILITY

System availability is the means whereby it can recover from failed components with minimal to no impact to the overall system. In the case of RDMA, the offloaded L3-4 connection contexts are a particularly challenging issue to address. The goal of serviceability for RDMA is to provide a mechanism to ensure the adapter and RNIC driver can perform a graceful Close and Error recovery, along with a mechanism to preserve the offloaded connection state information for future analysis in the face of such error events.

5.2.1 Offloaded State Verification

As mentioned in previous sections, the L2-3 protocol offload into the RNIC domain introduces challenges achieving availability. When the L2-3 protocols are in

software, their state is globally accessible and transportable to any underlying stateless NIC adapter. This allows for simplification of the availability approaches using 802.1AX or NIC teaming. This however is not possible in RNICs since the connection context resides within the individual RNICs. A failure by the adapter is effectively a loss of all context information. Under the TCP in kernel approach, the TCP connections would still be available and thus any losses would be addressed via retransmissions over the backup links.

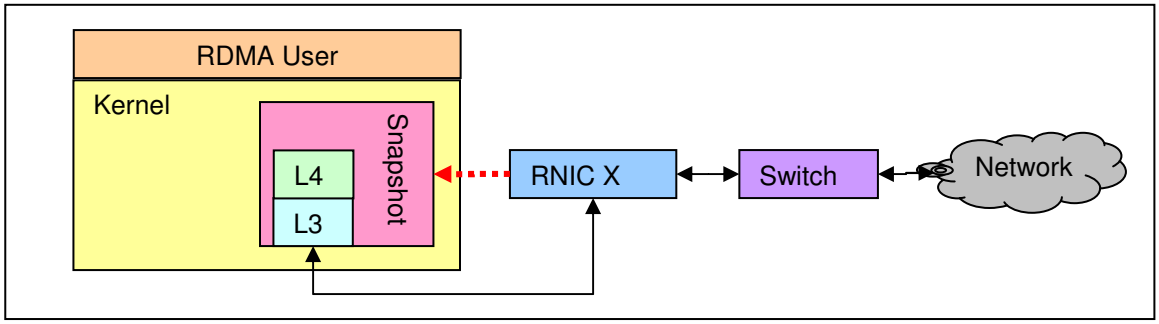

Figure 13: RNIC connections and states extracted into host memory

Though one of the major benefits of RDMA includes high-throughput and low-latency, there are cases where the former are not hard requirements and the interest is skewed towards low-power and low CPU utilization workloads. For these environments it is feasible for the RNIC L3-4 states to be periodically extracted into host memory. A simple design approach is to extract the L3-4 states at a software defined interval (Fig. 13). Given that adapter vendors are likely to contain proprietary implementations of L3-4 protocols, there are no explicit requirements placed upon the RNIC on how the contents are used.

The main advantage of the state extract approach is that in the case of adapter failure, the L3-4 and any additional adapter specific information is available in host

memory as a historical record to aid in root cause determination. During the service process, the snapshot contents can be provided to the RNIC vendor for analysis.

Under the traditional RNIC model, an adapter failure will translate to loss of all L3-4 information and connections from the RDMA application point of view; hence the snapshot approach provides a mechanism to preserve critical connection details which would otherwise have to be obtained via problem recreation and software instrumentation. It is clear that the contents may not reflect the actual state of the device at error time, however this approach provides at minimum some information that can aid in debug whereas the lack of this feature translates to no L3-4 data available for analysis.

5.2.2 Aggregation

In order to overcome the SPoF scenario an RDMA API, such as DAPL or SDP, necessarily has to implement the use of multiple adapters and implement aggregation at the Endpoint level for both RNICs (Figure 14). Abstracting this functionality from the application allows for API handling of any adapter failures.

Since the L3-4 connection contexts are resident in the RNIC memory, the API must perform Endpoint creation across both adapters. This differs from standard aggregation procedures, such as NIC Teaming and 801.1AX, in that the RDMA API must duplicate all connection establishment operations and resource creation actions on both adapters. The secondary adapter must have a connection context to the same remote host as the primary. MR resources must also be registered on both adapters such that in the advent of a primary RNIC failure, the application egress messages can be re-routed by the RDMA API to the secondary adapter without the participation of the application.

Though both Endpoints can be used for transmission, concurrency may introduce complexity in message ordering at the receiver based on network load balancing

mechanisms in use. To overcome this, the sending API would need to hash traffic from the sender to the same adapter always to avoid the out of order reception problem on the remote Endpoint. It is therefore recommend, strictly for RAS simplification purposes, that an active-passive approach be used.

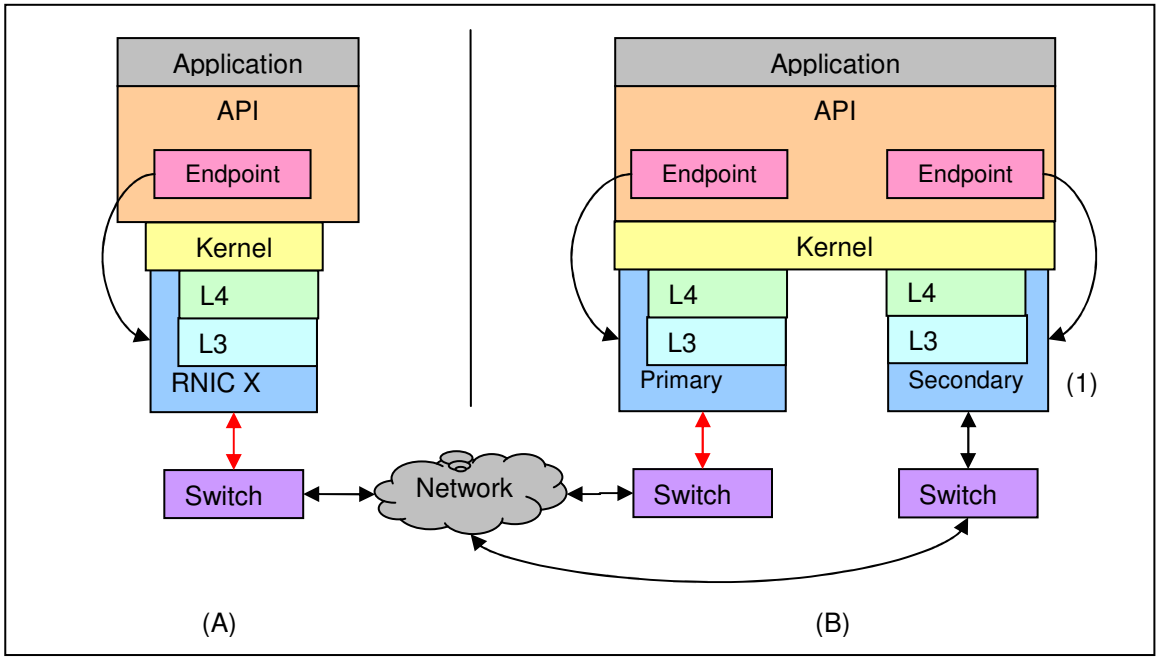

Figure 14: API level aggregation to overcome RNIC SPoF

5.2.3 PCI Error Recovery

PCI bus errors necessarily translate to an adapter failure or freeze event for systems configured with PCI Error recovery functionality. As described in previous architecture components, when this event occurs, all L3-4 connection state information resident on the adapter is lost. All RDMA connections will need to transition to error state indicating to the user, that they must close and de-allocate all resources. A typical PCI error implementation will take this into account and allow for a BUSY return by the L2 driver during error recovery phases.

A viable approach to availability in the context of PCI bus errors revolves around the intent to recover, reset, and restore the adapter to operational state as fast as possible. Clearly rouge users who do not perform proper resource de-allocations will cause the adapter recovery time to extend indefinitely, however the PID tracking mechanism provides a means whereby any rouge processes are identified and brought to the attention of the administrator or service personnel such that they can be manually removed. A manual correction of this state clearly does not meet the availability requirement of fast PCI error recovery; hence an automated mechanism is preferred. It is therefore recommended that the PID tracker information be used by a management program to cleanup the (rogue) processes. The implementation is simplistic in the sense that a background process is executing on the system which the RNIC software registers with. At error time, an asynchronous notification or wakeup is provided to the process that receives or retrieves the PID and application name. If the application name is one of the automatic removal candidates, the process proceeds to kill said application automatically.

The approach allows for implementing a guaranteed time quanta and bounds the RNIC device recovery time. Hence to increase availability, the automated approach is ideal as long as the candidates are defined and appropriate application teardown operations are in place.

5.2.3 Unexpected Close

A typical Ethernet to the system assumes that it can be initialized by the action of setting the IP address via manual or automated mechanisms such as DHCP. All operations over the interface are multiplexed by L4 protocols over the IP address, hence the removal of the IP address while connections are active result in the termination of all connections. This is not the case under the RDMA model where offloaded connections

are in operation alongside traditional non-offloaded connections. Clearly a removal of the IP address on the interface will close non-offloaded connections; however it will not close the offloaded connections. Without the NIC interface in place the RDMA interface cannot perform address resolution via ARP, hence when the NIC interface is removed, the RDMA connections must be terminated.

Before the device can be reconfigured again, the same mechanism discussed for PCI Error recovery must be used, to ensure no rogue users extend the Close time unnecessarily. If an automated solution is required, then the same constraints for addressing any application specific PID destruction process are necessary.

5.3 SERVICEABILITY

Serviceability is defined as the capability to determine why a system failure has occurred, and the associated ease with which the root cause determination can take place. The RDMA RAS architecture attempts to address both hardware and software types of failures along with guidelines and best practices for fast problem resolution. The key focus is on speed in root cause determination to reduce maintenance costs.

5.3.1 Component Tracing

The tracing capabilities implemented in the OFED implantation of RDMA and the standard Linux kernel do not allow for granularity in tracing. Effectively an all or none approach which does not provide a granular tunable for the desired level of tracing. Performance and timing sensitive environments require the ability to dynamically tune the level of tracing during program execution to allow tradeoffs in debug and performance. The cost of a single trace requires an indirection in code execution to save

off trace parameters to a separate reserved location in system memory, or alternately to a log file as is most common in Linux implementations. The latter obviously being more expensive in terms of processing time, hence it is not uncommon for RAS features such as tracing to hide timing sensitive problems or be disabled by default resulting in no historical data at error time.

```
#define TRACER(_level,  _desc, _tag, _d1, _d2, _d3, _d4) {                    \
    uint64_t _curr_level = get_curr_trace_level();                            \
    if (_level >= _curr_level) {                                              \
        log_trace(_level, mem_dest,  _tag, _desc,                             \
            (ulong_t) _d1, (ulong_t) _d2, (ulong_t) _d3, (ulong_t) _d4);      \
                                                                              \
        trace_display( get_cpu_id(),                                          \
            __LINE__, __FILE__, __FUNCTION__,                                 \
            _desc, _tag,                                                      \
            (uint64_t) _d1, (uint64_t) _d2, (uint64_t) _d3, (uint64_t) _d4);  \
    }                                                                         \
}
```

Figure 15: Pseudocode for granular tracing

A granular tracing mechanism is proposed which allows for the dynamic specification of a trace level during program execution (Fig. 15). The use of a macro is advantageous over an inline function call since it expands within the calling function. Inline functions are generally preferred over macros due to simpler verification by leveraging the compiler verbose output, however in this case the desire is to provide a simplified API to the calling function that allows for instrumentation with caller details. The module being traced will contain a master variable that can be set via IOCTL or other out-of-band mechanism. If the *_curr_level* is of greater or equal value than the passed in *_level*, then the tracing actions are performed. Otherwise the tracing actions are circumvented and the only cost incurred is obtaining the *_curr_level* and the evaluation of the conditional.

The trace_display() feature within the macro is optional, and allows for output of the contents to the system console or filesystem. It provides an instrumentation to acquire additional data, beyond the *_tag* and *_d1* through *_d4* word values. The console approach is advantageous over file output in that during a system crash scenario, the historical data is immediately available via console history, whereas the file output or normal trace mechanisms would require accessing the data via debugger raw memory reads. The debugger is clearly a process of higher cognitive complexity than inspecting the formatted output to console. To facilitate trace output review, the CPU ID, filename, line number, and function name descriptions are included along with a variable length text description. The additional data allows for quickly correlating a trace point to the source code which is most useful in a console/terminal environment, whereas the standard trace facility in log_trace() only provides the sense data d1-4 and a 8byte tag value which must be manually correlated with the source code to find the file, line, and function.

Trace Level		Trace Output								
		1	2	3	4	5	6	7	8	9
All On	On									
All Off	Off									
Error	1									
Rare	2									
Reserved	3									
Function Entry and Exit	4									
Function Internals	5									
Reserved	6									
HOT Function Entry and Exit	7									
HOT Function Internals	8									
Data Dumps	9									

Figure 16: Granular and dynamically tunable tracing levels

The architecture for tracing levels is based on tracing only the *_curr_level* value and all values below (Fig. 16). Error traces are always traced by default, but if a user sets

a trace *_level* of 5, then tracing will occur for 5 and all levels below. In trace level table example settings, an incremental selection of levels separate normal data paths from the performance paths. The intent is to correlate the highest trace levels to the amount of performance impact to the data path.

5.3.2 Memory Management

With a traditional NIC driver, the numbers of resources in use are bounded to a small manageable footprint. A typical NIC driver contains the basic operations previously outlined (Fig. 4). The number of resources and data structures required to achieve operation are trivial and manageable without the need for elaborate memory allocators or trackers. As RDMA increasingly becomes commonplace as a superset of the NIC features, the complexity in the high number and dynamic nature of the resources becomes a crucial design point for software maintainability. Since a typical adapter can scale up to ~64K or 16M RDMA connections, and each connection typically contains anywhere from 6-10 distinct memory resources, a robust and efficient memory tracking mechanism is required for RAS.

The model proposed herein allows for the dynamic allocation, tracking, and servicing of memory resources with minimal serialization and logic to address both process level operations and the constraints of interrupt level processing. Additionally, integrated memory utilization statistics are provided to allow for a high level view of resource utilization without the need for external memory statistics mechanisms.

5.3.2.1 Leak Detection

Memory leaks issues within a kernel environment typically require the detection of the event and the traceback to the module responsible for the leak. Commercial products [29] exist to address these issues; however they come at both an economic and complexity cost since it applies to an instrumented environment only. The dynamic analysis capability provided by these tools is only valid if the actual leak condition is encountered during execution. A memory tracking mechanism service is proposed which is separate but compiled into the RNIC driver and allows for field enablement such that if the issue is not found in a lab environment, it can be enabled in the customer environment to detect the problem. Clearly, a commercial product to detect memory leaks is unlikely to be installed and deployed on customer production systems.

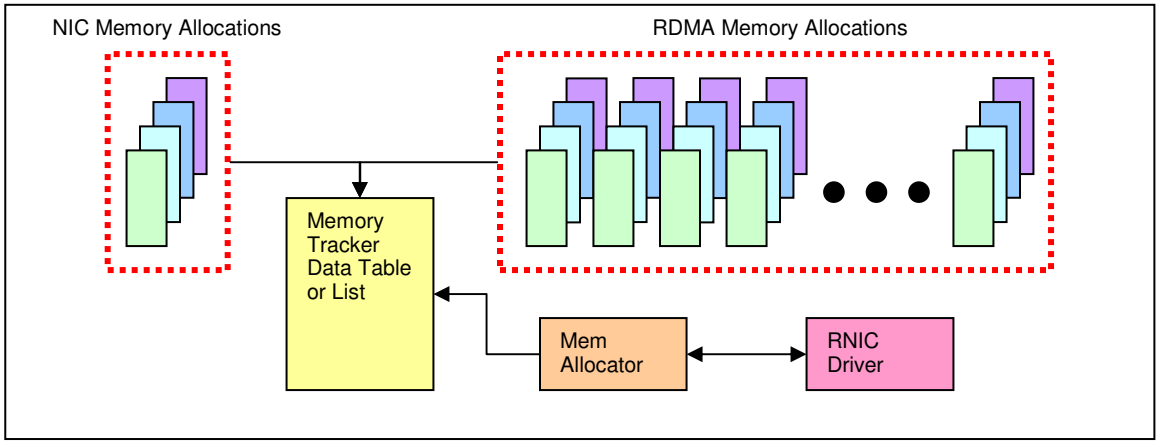

Figure 17: Conceptual memory tracker

The basic memory tracker concept (Fig. 17) revolves around a wrapper to all memory allocation functions which instrument the operation to keep a record for reference. This differs from any records and tracking kept by the general OS services in that the sole purpose is deterministic memory leak detection with human readable output

indicating the specific source line of code at fault. A memory allocation will be executed via a C macro call, which expands to record the filename, line number, and calling function. This information along with relevant data such as memory type, alignment, and size are included in an entry within the Memory Tracker table. Note that the memory tracker table can alternately be implemented as a linked list, hence the location for recording the data is implementation specific. In order to reduce serialization of memory table updates, an atomic mechanism to add and delete a record are preferred. As memory is freed, its corresponding table entry is atomically removed, thus clearing the record.

The memory tracker services are best implemented by allocating the table resources at RNIC driver initialization such that all subsequent memory allocations are tracked. If a table mechanism is used then the driver developer clearly needs to be aware of the limits in entries. This is given since the scope of resources and number of Endpoints is a well known value as discussed in the memory management section introduction.

At device Close time, a simple routine to traverse the table and detect the presence of record entries would indicate that a memory leak condition has occurred. Once the condition is detected, the record information can be used to generate an error message to the system log or other OS dependent mechanism. A service representative or customer can now clearly detect the module, file, line, and function in error such that the allocating function can be reviewed for root cause of the leak. In a production environment this is the most likely case, while in a development environment it would be preferable to assert the system such that this condition does not go undetected if the system logs are not being observed.

An additional benefit of the memory tracker is that per table statistics can be automatically generated to obtain the memory footprint of all allocations associated with the table. Therefore no out-of-band mechanism for memory statistics is required.

5.3.2.2 Coalesced Structures

Error recovery from inability to allocate memory is a basic essential feature of all software. Typically programs allocate memory granularly for each resource that suffices for managing a small number of resources, but leads to resource sprawl (Fig. 18.A) as the number scales. Furthermore, granular allocations necessitate software error handling and recovery logic for each failure. A guideline for circumventing resource sprawl, reducing software error recovery logic, and simplifying the memory layout, is to have coalesced generic allocations. When allocating an Endpoint, the parameters for each resource are function input parameters and can thus be calculated to obtain the total memory size required. Once total size is known, a single allocation can be used (Fig. 18.B), which reduces the insufficient memory error recover logic to a single memory error operation. An additional benefit to this approach is that during system assert time, memory inspection via debuggers is simplified since all structure contents are coalesced into a single location.

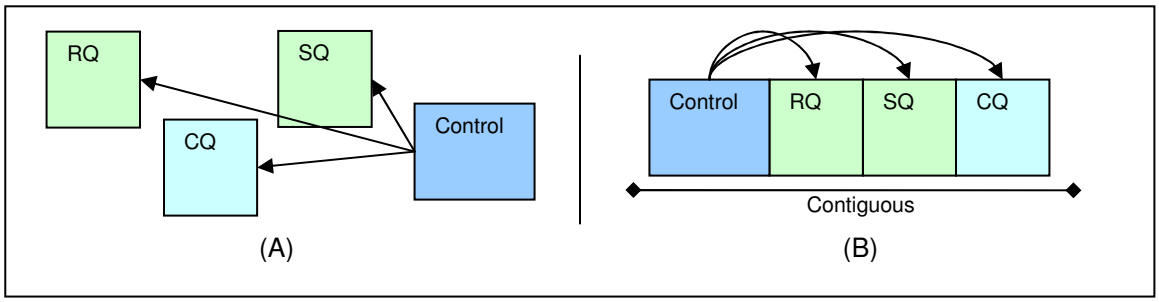

Figure 18: Memory allocation coalescing

In the provided figure, the CQ, RQ, and SQ resources are DMA mapped which typically require the starting addressed to be aligned on 4096B boundaries. Each if these is normally allocated separately. Given that the length of each resource is variable length, the ending addresses may not align on said boundaries, therefore padding will need to take place up to the next 4096B alignment at which the following resource starting address can be used. In order to simplify access to the memory a set of pointers in the control portion of the structure are used. The pointers will correspond to the offsets in the contiguous memory block. Normally these pointers would point to separately independent memory allocations.

5.3.2.3 Hexdump Markers

Memory contents accessed via debugger are typically in raw hex format, which introduces difficulty in quickly identifying structure boundaries (Fig. 19A), however as shown previously, an ASCII decode capability is typically available in modern OS debuggers. This capability is leveraged for raw memory inspection to allow readability via structure markers by inserting human readable ASCII values to uniquely identify the structure (Fig. 19B). When allocating memory that is purposed for DMA mapping, it must start on a 4096B boundary. Resources typically subjected to this constraint are CQ, SQ, RQ, and MRs, thus inserting a marker at the top of the structure results in an incorrect DMA offset (Fig. 19C). It can be argued that simply viewing the starting address is sufficient to identify the correct start location, however it means that structure marker protection mechanism is used inconsistently, which may be acceptable in some scenarios.

An acceptable option is memory allocation guidelines where padding is placed before and after the DMA aligned memory (Fig. 19D), such that a marker can be inserted

in the 8bytes immediately preceding the DMA address start alignment. This provides the benefit of clear and concise display in ASCII decode within the debugger and avoids determining the start of the memory space based solely on the address offset. The padding area is not completely wasted as it will typically hold the non-DMA mapped control structures. The software can also be instrumented such that at resource allocation time an inspection on the alignment of the starting resource address is performed. The latter action is a debug option as the structures will not change once the source is compiled and binaries are generated but, provide a means to verify software correctness after extensibility.

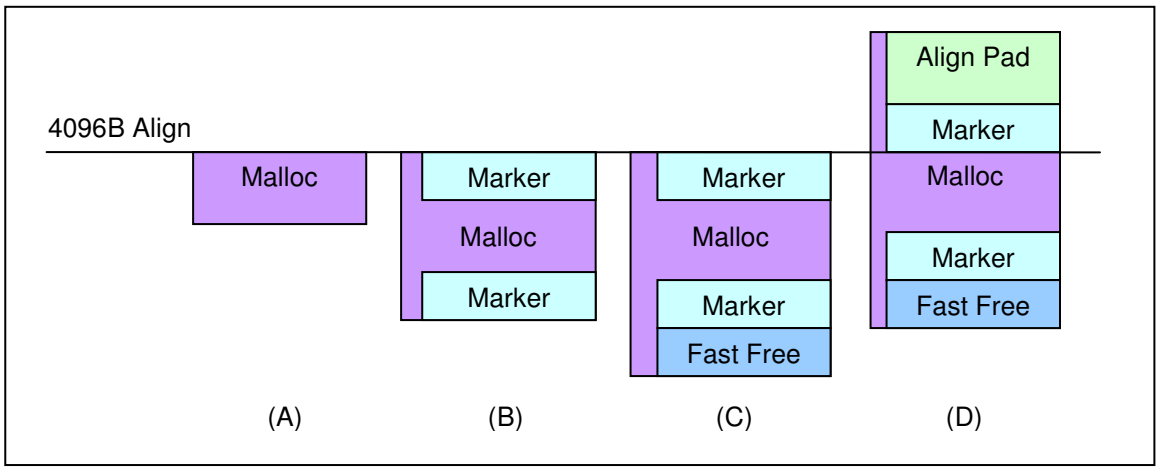

Figure 19: Generic managed memory and structure layout

In the example provided a single allocation is described, however, recall that in previous marker discussions, structures are nested and thus markers themselves are nested. In using a simplified example, it can be clearly shown that the ending padding space can be used for additional data storage that is invisible to the caller. This is treated as a black box value for the purposes of memory free operations. A call to allocate memory necessarily has a size parameter passed in. A Fast Free structure is appended to

the total allocation size for memory tracking purposes. The Fast Free structure contains details which indicate table offset or address or record, thus allowing for memory de-allocation without the need for searching for the record entry within the memory allocation table previously discussed in the memory leak detection concept.

The Fast Free structure is also protected by markers such that in the event of the memory user writing past the end of his allocation, the data integrity problem can be detected. When this condition occurs, the memory management logic will automatically perform a recovery operation which translates to a secondary mechanism for free via linear search in Memory Tracker table for the record matching the starting address of the memory at hand.

5.3.2.4 Cross-Mapped Memory

The final memory management feature proposed is Kernelspace to Userspace cross memory mapping, which allows for the kernel to read and write to Userspace memory. Recall that under the RDMA model, the user has a direct channel to the adapter and the kernel is not an active participant during data transfer operations. The major RAS issue introduced by this approach is that there is no direct communication between the Userspace Provider Library and the Kernelspace Provider driver to perform exchanges necessary for fast error notification between the components.

Under the OFED model, error communication between the Userspace and Kernelspace RNIC components is mediated via the OFED stack. The stack traverses both Userspace and Kernelspace via multiple components (Fig. 20). These components are separate from the RNIC Provider driver and Provider Kernelspace, however when a resource is created or destroyed in Userspace, the OFED threads perform a context switch into the kernel. The kernel Provider driver is then responsible for performing the actions

to make the Userspace resources visible to the RNIC. Given the information is readily available, and that the resources are coalesced, the kernel Provider driver can simply perform an indirection to cross map the memory. This is a minimal cost in terms of processing time; simply the cost of an additional structure for the kernel Provider to track. The Userspace users or resources are unaware that the cross memory mapping action has occurred.

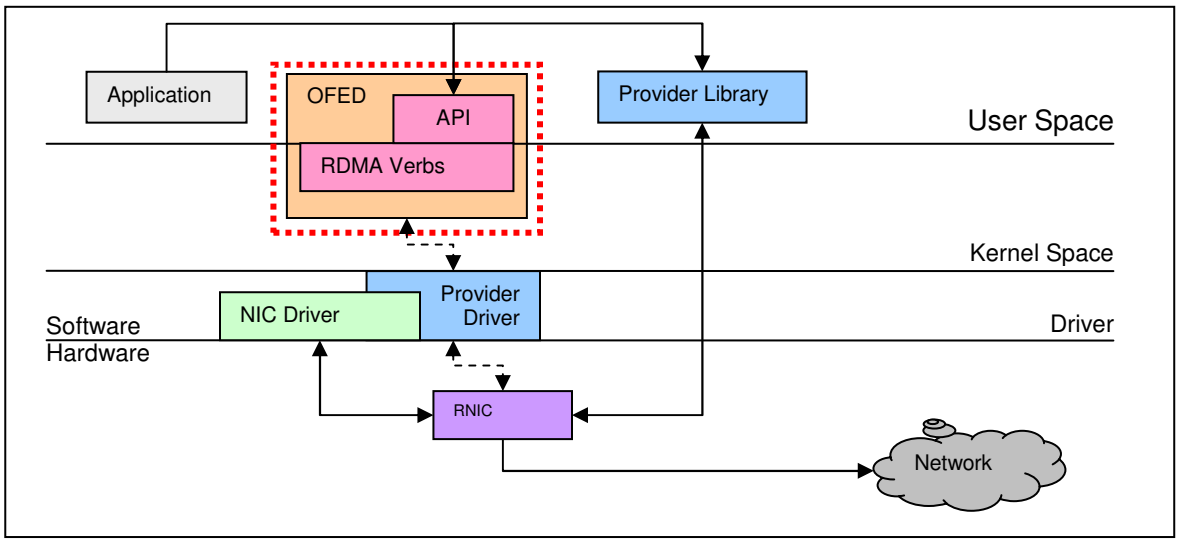

Figure 20: RDMA error notification model via OFED

During runtime, if fatal errors such as PCI errors that effectively disable the adapter occur, the Provider driver can write to the Userspace resource or Kernelspace structures indicating the error type. Now there is a direct communication channel via shared memory between the Provider and the specific resource owned by the Kernelspace (Fig. 21). This mechanism is independent of execution thread and thus serves as a mailbox type notification between the layers. When the Userspace application attempts to perform a send or receive action, the Provider Library will check the error structure and return to the user with an error code.

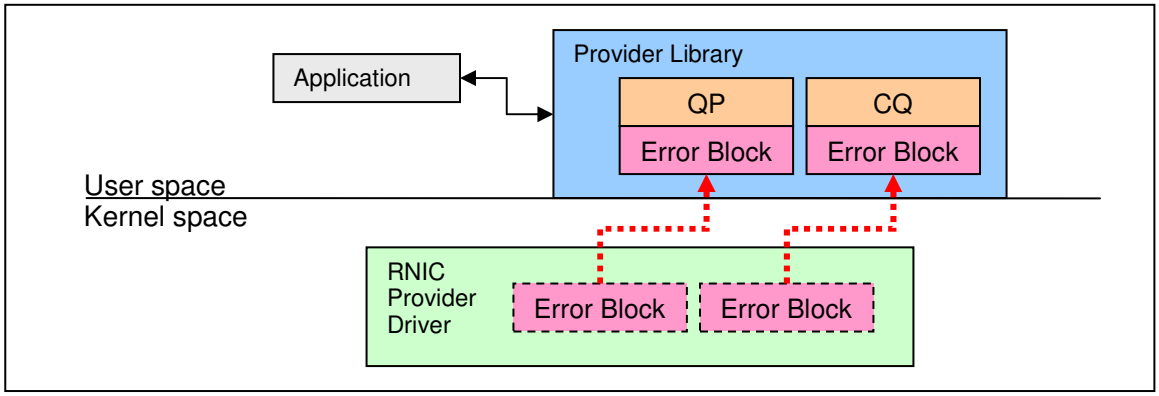

Figure 21: Cross mapped shared memory for asynchronous error notification

5.3.3 Resource Snapshot

Under the Sockets model, when an error arises the Socket is simply closed and the resources are freed. The application will typically perform an error recovery and eventually open another Socket to reestablish communications. This is the same behavior used for RDMA connection management, however in the latter case there is no central location for historical data as in the Sockets model that can leverage Kernelspace traces and memory for RAS. A mechanism to capture the resource state at error detection time is needed in order to preserve historical data for root cause analysis (Fig. 22). Given that the cross memory mapping feature provides read access to all Userspace data structures and resources, it is now possible perform a copy of each structure into Kernelspace as a linked list of individual resources and Endpoints. Once the snapshot is taken the users are free to perform any cleanup and error recovery actions without causing a loss of historical data. A service representative can then extract the historical data to perform root cause analysis.

The saved software state is complemented with the hardware state from the aforementioned RNIC offload state verification capability. The software view of the

RNIC state is accessible as is the hardware RNIC view, such that the RNIC vendor can perform a correlation between software and hardware views of states during error detection.

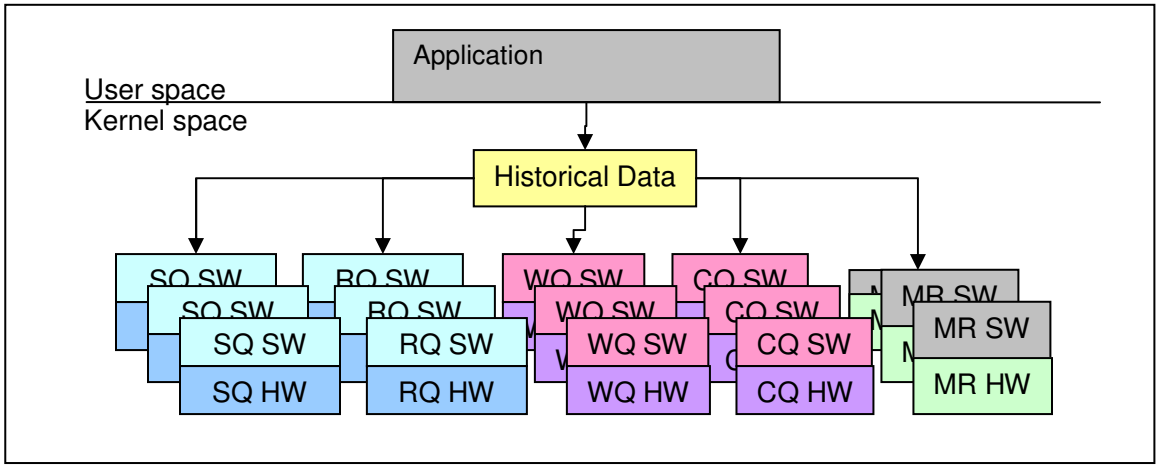

Figure 22: Snapshot of software and hardware structures at error detection time

5.3.4 Debugger Scripts

The debuggers included in most OSes are rich enough to allow for making C function calls directly from the debugger command line. This is an extremely useful feature for decoding complex data structures rather than viewing raw memory. Though the marker approach provides for a high level view of the memory layout by clearly identifying structure boundaries, contents, and individual variables; all are easiest to analyze via formatted output. Hence debugger scripts is a development guideline for displaying structure contents with the appropriate output so as to facilitate the finding of the variables of interest. An investment in creating a simple structure decoder will pay itself back many times over when problems arise, and the root cause identification process begins. Without structure decoders, a developer is left with repetitive manual

discovery and manual translation of structure contents. Furthermore it will complicate communication of the findings to other since one must explicitly describe the contents of raw memory to other.

5.3.4 Component Dump

A system dump provides the closest means to debug a system data without actually operating on the failing system. This is the action of last resort from a RAS point of view, since it has the effect of taking a system offline and dumping all memory contents to a local or remote location to allow for inspection. The dump process time is relative to the memory size of the system hence larger systems will be offline for extended periods of time. It is not uncommon for systems with ~50GB of memory to take over 12hrs to complete the dump process. From a RAS point of view, this action should be avoided at all costs unless it is the result of an existing system crash where the system is already offline.

When an individual kernel component such as an RNIC is not operating correctly or displays intermittent errors which are not critical to system operation, it may be preferable to obtain a dump of only the RNIC HW, Provider, and Provider Library without a full system outage. The proposed component dump capability is composed of a set of OS services which allow for the registration of software memory allocations such that in the presence of an external trigger, memory reads and writes are quiesced and the memory contents are dumped in a manner similar to a system dump. The dump target is typically a remote dump device (Fig. 23) such that the root cause analysis occurs independently of the operation of the failed system.

The main advantage of this approach is that it allows for a small outage in the operation of a specific component instance, in this case the RNIC Provider and Provider Library, without affecting the overall operation of the system. After the dump is complete the RNIC and associated Provider and Provider Libraries are restarted and operation continues as normal.

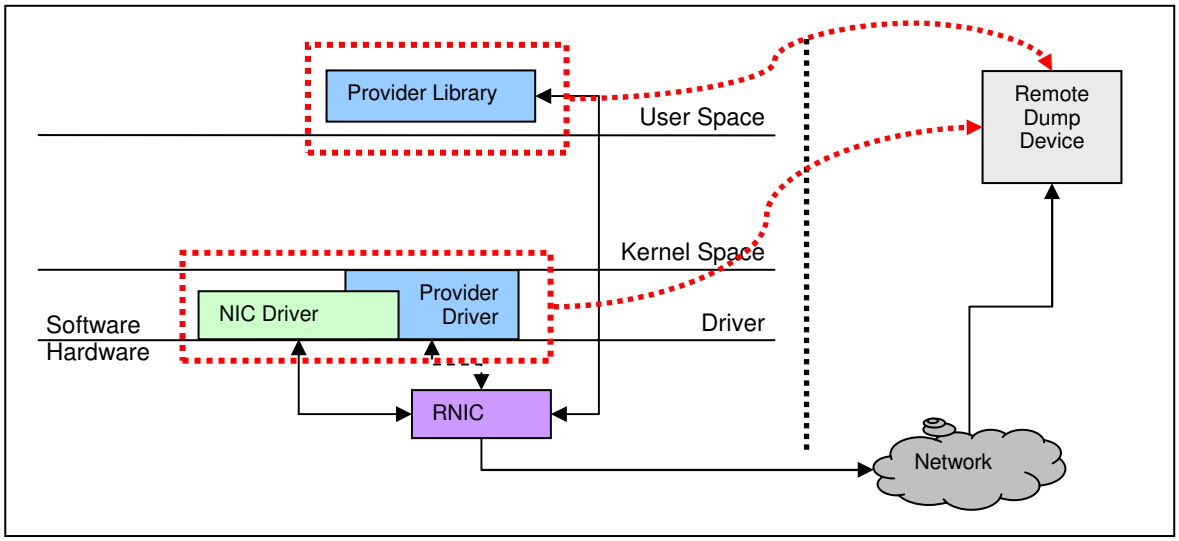

Figure 23: Component dump to remote device for root cause analysis

The component dump capability can be automatically triggered in the event of a PCI error such that a snapshot of the software state leading up to the event is available. Typically, all software resources are reset and only traces remain preserved for analysis. With the component dump capability now a more robust system is in place to automatically preserve additional data for root cause analysis.

6.0 CONCLUSION

As RDMA gains adoption in the enterprise data center a renewed focus on RAS, and the accompanying features expected by enterprise customers must be addressed for efficiency in problem determination and resolution. RDMA must evolve from the current focus on performance at the expense of maintainability and serviceability. Clearly, RAS architecture is a feature rich and constantly evolving area of software and hardware engineering, hence the RDMA RAS architecture address the key concepts necessary for success in the enterprise while placing the performance and RAS tradeoffs in the hands of the end users.

In summary, the RAS for RDMA communications provides an architecture and guidelines for efficient means of reducing the cognitive complexity associated with software maintenance in RDMA environments by applying a combination of new and traditional RAS approaches. From a customer point of view, the focus is on detecting and reporting the necessary data to the system vendor or service provider so as to restore the system to the correct operational state in the minimum amount of time. From the systems vendor point of view, the main goal is twofold. Firstly, a reduction in root cause identification time via software architectural features which provide the necessary forensics data. Secondly, via software design guidelines that reduce the cognitive complexity associated with software maintenance via decoders and memory managers.

The proposed RAS architecture addresses the aforementioned challenges and allows for extensibility of additional features in the future.

GLOSSARY

ACK	Acknowledgement
ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
CE	Converged Ethernet
DAPL	Direct Access Programming Library
DMA	Direct Memory Access
DoD	Department of Defense
DR	Disaster Recover
EEH	Enhanced Error Handling
FDDI	Fiber Distribution Data Interface
FFDC	First Failure Data Capture
FVT	Functional Verification Test
HCA	Host Channel Adapter
HPCS	High Productivity Computer Systems
HW	Hardware
IB	Infiniband
IOCTL	IO Control
IP	Internet Protocol
IPC	Inter-Process Communication
IST	Integration Systems Test
iWARP	Internet Wide Area RDMA Protocol
L3-4	Network and Transport Protocol Layers

Library	RNIC device driver in Userspace
MPA	Marker PDU Alignment
MR	Memory Region
MSS	Maximum Segment Size
MTTR	Mean Time To Repair
MTU	Maximum Transmit Unit
NIC	Network Interconnect Card
OFA	Open Fabrics Allicance
OFED	Open Fabrics Enterprise Distribution
OS	Operating System
PCI	Peripheral Component Interconnect
PD	Protection Domain
PDU	Protocol Data Unit
Provider	RNIC device driver in kernel
RAS	Reliability
RC	Reliable Connection
RD	Reliable Datagram
RDMA	Remove Direct Memory Access
RNIC	RDMA NIC
RoCE	RDMA over Converged Ethernet
RQ	Receive Queue
RX	Receive
SCTP	Streaming Control Transmission Protocol
SDP	Sockets Direct Protocol
SFDC	Second Failure Data Capture

SPoF	Single Point of Failure
SQ	Send Queue
SW	Software
TCP	Transmission Control Protocol
TOE	TCP Offload Engine
TX	Transmit
UC	Unreliable Connection
UD	Unreliable Datagram
ULP	Upper Layer Protocol
UT	Unit Test
VIA	Virtual Interface Architecture
WAN	Wide Area Network
WQ	Work Queue

REFERENCES

[1] RDMA over Converged Ethernet (RoCE) Annex A16. InfiniBand Trade Association, InfiniBand architecture. Specification Volume 1. Release 1.2.1. Nov. 2007 <http://www.infinibandta.com>. Apr. 2010

[2] "Architectural Specifications for RDMA over TCP/IP." RDMA Consortium. N.p., n.d. Web. 17 Apr. 2010. <http://www.rdmaconsortium.org/>.

[3] "HPCS-HIGH PRODUCTIVITY COMPUTER SYSTEMS." HPCS Application Analysis and Assessment. N.p., n.d. Web. 18 Nov. 2010. <http://www.highproductivity.org/kepner-HPCS.htm>.

[4] Loh, E., Van De Vanter, M. L, Votta, L.G. Can Software Engineering Solve the HPCS Problem? Proceedings Second International Workshop on Software Engineering for High Performance Computing System Applications, St. Louis, 15 May 2005.

[5] D. Dunning, G. Regnier, G. McAlpine, D. Cameron, B. Shubert, F. Berry, A. M. Merritt, E. Gronke, and C. Dodd. The virtual interface architecture. IEEE Micro, 18(2), 1998.

[6] InfiniBand Trade Association, InfiniBand architecture. Specification Volume 1. Release 1.2.1. Nov. 2007 <http://www.infinibandta.com>. Nov. 2007

[7] "IEEE 802.1: 802.1Qaz - Enhanced Transmission Selection." LMSC, LAN/MAN Standards Committee (Project 802). N.p., n.d. Web. 18 Nov. 2010. <http://www.ieee802.org/1/pages/802.1az.html>.

[8] "IEEE 802.1: 802.1Qbb - Priority-based Flow Control." LMSC, LAN/MAN Standards Committee (Project 802). N.p., n.d. Web. 18 Nov. 2010. <http://www.ieee802.org/1/pages/802.1bb.html>.

[9] Hauser, Brian . "CommsDesign - iWARP: Reducing Ethernet Overhead in Data Center Designs." CommsDesign. N.p., n.d. Web. 17 Apr. 2010. <http://www.commsdesign.com/design_corner/showArticle.jhtml?articleID=51202>

[10] Gregory F. Pfister. An introduction to the infiniband architecture. In Hai Jin, Toni Cortes, and Rajkumar Buyya, editors, High Performance Mass Storage and Parallel I/O: Technologies and Applications. IEEE/Wiley Press, New York, 2001.

[11] "The OpenFabrics Alliance." Open Frabrics Enterprise Distribution. N.p., n.d. Web. 18 Apr. 2010. <http://www.openfabrics.org/index.htm>.

[12] iWARP Protocol Kernel Space Software Implementation. Dennis Dalessandro, Ananth Devulapalli and Pete Wyckoff. Proceedings of the 20th IEEE International Parallel & Distributed Processing Symposium (IPDPS '06), Communication Architectures for Clusters Workshop, Rhodes Greece

[13] P. Frey, A Hasler, B Meltzer Server-Efficient High-Definition Media Dissemination. NOSSDAV ’09 Williamsburg, Virginia

[14] "RFC 5044 - Marker PDU Aligned Framing for TCP Specification." IETF Tools. N.p., n.d. Web. 19 Apr. 2010. <http://tools.ietf.org/html/rfc5044>.

[15] Liss, Liran. "RoCEE in OFED Update". Open Fabrics 2010 Sonoma Workshop. March 1 2010

[16] "Introduction to Receive-Side Scaling (Windows Driver Kit)." MSDN | Microsoft Development, Subscriptions, Resources, and More. N.p., n.d. Web. 18 Nov. 2010. <http://msdn.microsoft.com/en-us/library/ff556942%28VS.85%29.aspx>.

[17] IEEE Std 802.1AX-2008 IEEE Standard for Local and Metropolitan Area Networks — Link Aggregation. IEEE Standards Association. 3 November 2008. doi:10.1109/IEEESTD.2008.4668665

[18] A. Bhutani and Z. Mahmood, "Using NIC Teaming to Achieve High Availability on Linux Platforms," Dells Magazine, February 2003.

[19] D. Henderson, B.Warner, and J.Mitchell. IBMPOWER6 processor-based systems: Designed for availability. White paper, 2007.

[20] Minimizing the Hidden Cost of RDMA, Frey, P.W.; Alonso, G.; Distributed Computing Systems, 2009. ICDCS '09. 29th IEEE International Conference on Digital Object Identifier: 10.1109/ICDCS.2009.32 Publication Year: 2009 , Page(s): 553 – 560

[21] S. Cherem, L. Princehouse, and R. Rugina. Practical memory leak detection using guarded value-flow analysis. In PLDI '07: Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation, pages 480–491, 2007.

[22] Yichen Xie and Alex Aiken. Context- and path-sensitive memory leak detection. In Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of

software engineering (ESEC'05/FSE'05), volume 30 of SIGSOFT Software Engineering Notes, pages 115{125, New York, NY, USA, 2005. ACM Press.

[23] B. Cobb and S. Dutta. Storage Protection Keys on AIX Version 5.3. White paper, 2007.

[24] "Apache log4j 1.2 - log4j 1.2." Apache Logging Services Project - Welcome to Apache Logging Services. N.p., n.d. Web. 18 Nov. 2010. <http://logging.apache.org/log4j/1.2/>.

[25] "The AIX Error Logging Facility." Black Sheep Networks Inc.. N.p., n.d. Web. 16 Nov. 2010. <http://www.blacksheepnetworks.com/security/resources/aix-error-logging-facility.html>.

[26] "System Dump Facility." AIX 6.1 Information Center. N.p., n.d. Web. 29 Oct. 2010. <publib.boulder.ibm.com/infocenter/aix/v6r1/index.jsp?topic=/com.ibm.aix.kernelext/doc/kernextc/sysdumpfac.htm>.

[27] "Live Dump Facility." AIX 6.1 Information Center. N.p., n.d. Web. 7 Nov. 2010. <http://publib.boulder.ibm.com/infocenter/aix/v6r1/index.jsp?topic=/com.ibm.aix.kernelext/doc/kernextc/livedumpfac.htm>

[28] Mack, M. J.; Sauer, W. M.; Swaney, S. B.; Mealey, B. G.; IBM Journal of Research and Development Volume: 51 , Issue: 6 Digital Object Identifier: 10.1147/rd.516.0763 Publication Year: 2007 , Page(s): 763 - 774

[29] "PurifyPlus ." IBM - United States. N.p., n.d. Web. 16 Nov. 2010. <www-01.ibm.com/software/awdtools/purifyplus/>.

[30] "IEEE 802.1: 802.1Qau - Congestion Notification." LMSC, LAN/MAN Standards Committee (Project 802). N.p., n.d. Web. 18 Nov. 2010. <http://www.ieee802.org/1/pages/802.1au.html>.

VITA

Omar Cardona holds an associate in Humanities from the Universidad de Puerto Rico (1998) and a Bachelor in Computer Science from the Universidad Inter-Americana de Puerto Rico (2001). He is currently employed at IBM where he designs and architects solutions in IO Virtualization, High-Performance Communications, Device Drivers, and Ethernet Switching.

omarcardona@yahoo.com

This report was typed by Omar Cardona.