

RAMS: A RDMA-enabled I/O Cache Architecture for Clustered network Servers

Peng Gu, Jun Wang
Computer Science and Engineering Department
University of Nebraska-Lincoln

Abstract

Abstract: Previous studies show that intra-cluster communication easily becomes a major performance bottleneck for a wide range of small write-sharing workloads especially read-only workloads in modern clustered network servers. A Remote Direct Memory Access (RDMA) technique has been recommended by many researchers to address the problem but how to well utilize RDMA is still in its infancy. This paper proposed a novel solution to boost intra-cluster communication performance by creatively developing a RDMA-enabled collaborative I/O cache Architecture called RAMS, which aims to smartly cache the most recently used RDMA-based intra-cluster data transfer processes for future reuse. RAMS makes two major contributions to facilitate the RDMA deployment: 1) design a novel RDMA-based user-level buffer cache architecture to cache both intra-cluster transferred data and data references; 2) develop three propagated update protocols to attack a RDMA read failure problem. Comprehensive experimental results show that three proposed new update protocols of RAMS can slash the RDMA read failure rate by 75%, and indirectly boost the system throughput by more than 50%, compared with a baseline system using Remote Procedure Call (RPC).

Keywords: cluster, server, RDMA, cache, RDMA read failure, Web.

1. Introduction

Previous studies show that intra-cluster communication may become a major performance bottleneck for cluster-based servers and advocate adopting Remote Direct Memory Access (RDMA) technique to attack the problem. Carrera et al. studied an 8-node clustered Web server by replaying four real-world WWW

traces. Their results show that more than 50% of the CPU time is spent on intra-cluster communication for all traces in clustered servers because of the expensive TCP protocol handling [1]. They recommend adopting a user-level communication technique to address the problem. Emerging user-level network communication techniques do provide opportunities to address the intra-cluster communication problem. The RDMA technique grants applications with direct control over network communications and achieves major benefits such as low processor overhead, remote memory reads/writes and zero-copy. Several state-of-the-art research work have been reported [2, 3, 4, 5] on developing software libraries of MPI, MPICH [6] and MPICH2 [7] to incorporate high-performance application-level communication techniques such as RDMA and Remote Memory Access (RMA) for parallel data-intensive applications in clusters connected by InfiniBand [8] and Myrinet [9].

The authors notice that, in order to incorporate RDMA technique, two distinct factors from standard RPC technique based on TCP/UDP protocol must be considered: 1) the sender and the receiver must know the virtual address of the transferred data before the transfer is initiated, and 2) since the data transfer occurs at the user level, the RDMA read operation may fail when the requested data has already been swapped out of the physical memory on the source side (called *RDMA read failure*). Both issues are either not addressed or not fully addressed in above-mentioned work. During a RDMA transfer process, one task is to keep RDMA-related pages memory-resident. Wiring pages for long periods of time leads to underutilization of memory. A wiring on-demand policy that pins the requested data page on-demand per RDMA operation by the host is preferred since it can increase the utilization of memory space. For systems employing this policy, frequent RDMA read failures would potentially lead to a serious performance bottleneck for cluster-based servers.

This paper develops a new RDMA-based collabo-

native I/O cache system architecture called RAMS to address the above problems. RAMS is constructed by symmetrically deploying a RDMA-based, user-level, caching software component called CacheRDMA on each cluster node. Each peering CacheRDMA works collaboratively to provide efficient services for object¹ sharing and exchange in clustered servers. The CacheRDMA manages its buffer cache into two partitions, an outgoing cache partition saving data to be sent out to remote nodes and an incoming cache saving data sent from remote nodes. In addition, the cache reference directory design in CacheRDMA is a two-edge sword: 1) saving additional history virtual address information of the in-memory data to be RDMA to avoid RDMA read failure, which is *a unique feature from existing user-level cache systems*, and 2) realizing collaborative caching by fetching the requested data (use RDMA read) from remote nodes. Furthermore, RAMS develops three propagated update protocols to adaptively propagate directory cache updates to resolve the RDMA read failure problem, which may become a potential performance bottleneck. A configurable, trace-driven, discrete-event RAMS simulator for medium-scale clustered servers (up to 256 nodes) is developed, compared with a RPC-based data communication baseline system. We choose one representative application – cluster-based Web server – as a sample testbed and three real-world Web server traces for validation. A comprehensive set of experimental results prove that, compared with RPC baseline system, RAMS boosts the system throughput by more than 50% by significantly improving the intra-cluster communication performance in cluster-based Web server.

The remaining part of this paper is organized as follows: we describe the design and implementation of RAMS in Section 2, in Section 3 we present our experimental methodology, and analyze the results in Section 4, we discuss the related work in Section 5 and give the conclusion remarks in Section 6.

2. The Design and Implementation of RAMS

The software component of RAMS is a lightweight, distributed middleware software substrate interfacing applications that employ RDMA as intra-cluster communication technique in the clustered server. To achieve good portability, RAMS is implemented as a multi-thread application-level software component layered on top of a user-level VI Provider Library (VIPL) [10] or modular VI Provider API (M-VI) [11].

2.1. RAMS System Architecture

RAMS works in the middle between the application server layer and OS layer. As explained before, RDMA data transfer bears two distinct factors from RPC that must be considered for implementing RAMS: 1) the sender and the receiver must know the virtual address of the transferred data by each other before the data transfer, and 2) since the data transfer occurs at the user level, the RDMA read operation may fail when the requested data has already been swapped out of the physical memory on the source side. To address both problems, as a building block for RAMS on each node, CacheRDMA is developed including the following components:

1. an incoming buffer cache saving incoming data received from remote nodes
2. an incoming cache reference directory for incoming buffer cache that tracks the virtual address of recent incoming data that is transferred by RDMA
3. an outgoing buffer cache saving outgoing data sent out to remote nodes
4. an outgoing cache reference directory for outgoing buffer cache that tracks the virtual address of recent outgoing data that is transferred by RDMA

Two particular cache reference directories in CacheRDMA are developed to help avoid RDMA read failure, save repeated and expensive memory registration and deregistration processes and realize collaborative caching by fetching the requested data (use RDMA read) from remote nodes. The design of both the incoming cache and outgoing cache is used to support collaborative caching and help avoid RDMA read failure.

During a RAMS working process, external requests from outside cluster would be firstly delivered to a dispatcher router, and then the router dispatches requests to the corresponding cluster nodes. Upon receiving a request from the dispatcher on a cluster node, the application server processes the request with possible help from RAMS. Because RAMS provides collaborative I/O caching service, the requests may benefit from the hits in RAMS buffer cache on both local and remote cluster nodes.

In the event of a RDMA data transfer, RAMS would check if the requested data has been RDMAed from other remote nodes before and the remote virtual address of such a RDMA transfer is cached. If so, RAMS has a choice of obtaining the data via RDMA read operation (Here we assume that the Network Interface Card (NIC) used in our system is VI compatible and has the ability to issue a RDMA read operation). If not, a *three-message model* is developed: first RAMS may send a

¹ Here the object can be file, URL document, storage object, etc.

notification message to the remote node that has the requested data. Second, upon receiving such a request, the remote node would initiate a disk I/O to fetch the requested data from its local file system to prepare the data. Third, when the requested data is read into RAMS buffer cache on the remote site, the RDMA read is initiated. This three-message model is typically used for RAMS cache initialization. It is more expensive than a normal RDMA read action in case of an incoming cache hit in RAMS because the three-message model requires two round intra-cluster communications, and some disk I/Os involved while the normal RDMA read operation in RAMS requires only one round communication and no disk I/Os engaged unless the RDMA read fails.

2.2. Designing the Propagated Update Protocols for RDMA Read Failure

One important design issue of RAMS is to reduce the RDMA read failure rate as much as possible. The reason is that the network I/O performance will be seriously degraded in the event of a high RDMA read failure rate. The idea is to keep the reference directory caches up-to-date. Our goal is to develop a good propagated update protocol that guarantees a low RDMA read failure rate while introduces a modest network bandwidth consumption. Three heuristic propagated update protocols are developed to conduct update operations only if necessary (i.e., at an appropriate time and with an appropriate frequency). To help analyze the design tradeoff, we also develop two bound protocols, including lazy update and aggressive update. They are elaborated in the following paragraphs.

2.2.1. Two Bound Protocols We develop two bound protocols to set up both bottom-bound and up-bound performance lines to help justify three proposed propagated update protocols of RAMS.

Lazy propagated update protocol This simply protocol is designed in that, RAMS never performs status updates for all cache reference directories in RAMS. Namely, the incoming cache directory on each node is never actively updated. In this method, the incoming cache directory entries may only be updated at two situations: RDMA read failure and cache replacement.

Aggressive propagated update protocol This protocol goes to the other extreme compared with the lazy one: whenever a status change of any RDMA outgoing directory entry occurs, the host node always propagate the update information to all related nodes (because there may be many nodes that have ever requested the same data from this node). However, the intra-cluster network bandwidth involved with this protocol may be exhausted and lead to an overloaded status.

2.2.2. Developing RAMS Propagated Update Protocols To strike a good tradeoff between a low RDMA read failure rate and a modest network bandwidth consumption introduced, we consider a hybrid approach by combining both bound protocols mentioned above, and develop three heuristic propagated update protocols. Among three protocols design, a threshold tracks the RDMA read failure rate on each node online is adopted on each cluster node as the water mark to heuristically invoke a propagated update operation. This threshold is used to bound the RDMA read failure rate within an acceptable range. When the actual RDMA read failure rate goes beyond the threshold, a propagated update progress is initiated. After the process, the RDMA incoming cache directory entries linked to the initiated node are up-to-date. Either the sender node or the receiver one will be chosen to calculate the RDMA read failure rate and proactively initiate such a propagated update process. Moreover, either a single-node or an entire-cluster (based on the accumulated RDMA read failure rate on the entire cluster) may be chosen to initiate a propagated update process.

We develop three propagated update protocols based on the selection of a host node that performs the RDMA read failure rate calculation and initiates the propagated update process.

Sender Initiated Update Protocol This protocol chooses the data sender node to collect the statistic information of its RDMA read failure rate. The data sender node counts the overall number of incoming RDMA read requests and calculates the total number of failures online. When such a failure rate exceeds a pre-defined threshold (10% in this paper), then the sender node would initiate a propagated update process to update RDMA incoming cache directories of those nodes from which the sender has ever fetched the data using RDMA read.

Receiver Initiated Update Protocol

In this protocol, we are monitoring the RDMA failure rate on the data receiver sides, i.e., the RDMA reader nodes. The receiver node itself is conscious of the success/failure status of each outgoing RDMA read request.

Accumulated Update Protocol To implement this update policy, we assign a master node (typically with light load) to calculate the overall RDMA read failure rate for the whole cluster. We count the total number of all the RDMA read requests of all cluster nodes and the total number of the RDMA read failures. When the aggregate RDMA read failure rate exceeds the threshold, the master

node would initiate the propagated reference directory update process.

3. Experimental Methodology

In this section, we describe the experimental methodology used to compare the performance of five different propagated update protocols in RAMS and a RPC-based baseline system.

3.1. Simulators

We developed a discrete-event, trace-driven clustered Web server based on a distributed Web server simulator called `dws_sim` from University of Saskatchewan. Beyond the `dws_sim`, we added around 4,000-line C++ codes to implement both RPC and RDMA based data communication protocols and corresponding virtual memory and Network Interface Card management in the RDMA communication model developed by Rutgers University [1], with an emphasis on intra-cluster communication. In addition, we developed necessary software components of RAMS to study its performance impact on clustered Web servers. To feed multiple traces with different formats into the simulator, we also developed a trace format transfer program to prepare the traces in a uniform format. The simulator can emulate a clustered Web server with up to 256 nodes, and up to 100 simulations are allowed to be run in a single execution of the simulator program. The simulator can flexibly accept many system configuration parameters through a configuration file to perform a comprehensive sensitivity study, including the number of cluster node, the file system cache size on each node.

3.2. Configuring simulation experiments

Our simulator models a clustered Web server running on 4 to 256 cluster nodes connected by a high-speed, RDMA-enabled LAN. Clients are connected by a router employing round-robin request dispatch policy. At a more detailed level, each node is comprised of a CPU, a memory chip, a NIC, and a disk storage subsystem. Each of these devices is modeled as a service center with one or more queues implemented to simulate the queuing effect. The simulation parameters estimate a Giganet VIA/cLAN router, a VIA Giganet LAN, an 2G Hz Pentium IV CPU, and an IBM Deskstar 75GXP disk. We derived some parameters from real measurements on a single node machine, with others filled by reference to the previous research work [1, 12].

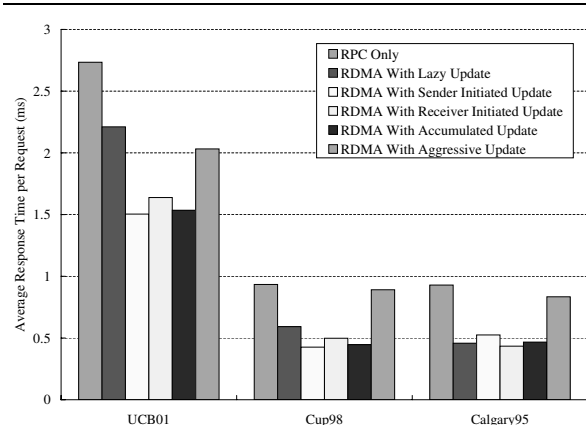


Figure 1. Average Response Time per Request under Three Traces in a 64-node Cluster.

4. Experimental Results and Discussion

This section describes the performance evaluation of three propagated update protocols of RAMS and RPC baseline system by replaying three real-world Web server traces. The performance metrics used in this paper includes Average Response Time per HTTP Request, and System Throughput,

4.1. RAMS versus RPC baseline system

We used two metrics to compare the performance of RAMS and RPC baseline system, average response time per HTTP request and system throughput. We replayed three real-life traces in five propagated update protocols of RAMS and the baseline system, varied the number of node from 4 to 256, repeated the trace-replay experiments, and collected results.

The first result of average intra-cluster network latency per request is shown in Figure 1, as collected by feeding three real-world traces into both system simulators in a 64-node cluster. From the figure we can see, all propagated update protocols of RAMS except the aggressive update protocol achieve a much less average response time, more than 50%, in two traces compared with that of RPC baseline system. Among different propagated update protocols, RDMA with sender initiated update protocol always generate low average response time. The UCB trace result in longer average response time. The reason may be bigger working data set and higher request density nature of the UCB trace. The RAMS is not much sensitive to the type of trace.

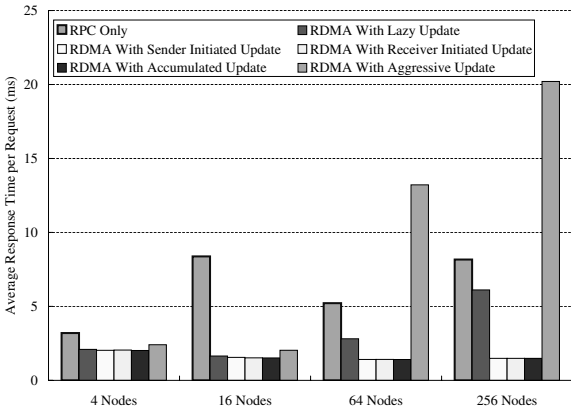


Figure 2. Average Response Time per Request under Different-Sized Clusters.

To see the results in different sized clustered Web servers, we varied the cluster size from 4 to 256, and plotted the results in Figure 2 by replaying the UCB trace (without specific explanation, the following results were collected by the same way). Similarly, RAMS outperforms RPC base line system in terms of the average request response time, with RDMA with accumulated propagate strategy achieving the best for all sized clusters.

We also collected the results of system throughput, a direct end-to-end performance measurement metric, as shown in Figure 3. From the figure, we can easily tell that the system throughput was greatly improved by the introduction of RAMS into cluster-based Web server. An average two factor of improvement of system throughput is achieved in all propagated update protocols of RAMS versus RPC baseline system. Again, RDMA with sender initiated propagate strategy performs the best as expected from its best performance gain in terms of intra-cluster transfer latency among three alternative propagated update protocols in RAMS.

4.2. Propagated Update Protocols of RAMS

In this section, we study the RDMA data transfer process in detail to help develop design trade offs among three propagated update protocols of RAMS.

The RDMA read failure rate is a direct metric to reflect how well each propagated update protocol works for cluster-based Web server. We collected results of RDMA read failure rate for three protocols of RAMS along with two bound protocols in a 64-node cluster. Figure 4 shows that, three propagated update protocols significantly reduce the RDMA read failure rate by up

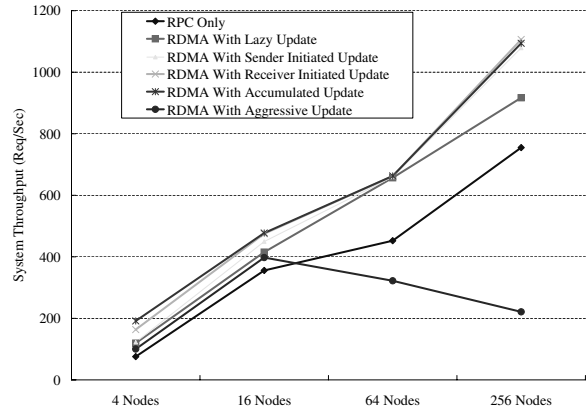


Figure 3. System throughput.

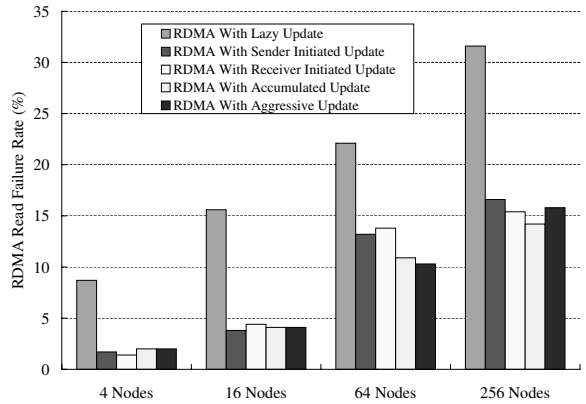


Figure 4. RDMA Read Failure Rates of RAMS Propagated Update Protocols.

to 75% compared with the lazy protocol, which delivers the highest RDMA read failure rate in RAMS. The result shows that, with increasing the number of nodes in the cluster, the RDMA read failure rate increased accordingly. The reason is that, with a round-robin dispatching strategy, larger number of nodes decreased the temporal locality of the trace requests. With degraded temporal locality, the data stored in RAMS cache simply has a high possibility to be swapped out to the disk, and thus increasing the RDMA read failure rate significantly.

5. Related Work

Emerging user-level network communication techniques provide opportunities to boost the performance of network and distributed file storage systems. Remote

Direct Memory Access (RDMA) is one of such representative techniques and will be quickly propagated in the future high-performance storage networking market.

One of the most successful storage applications, Direct Access File System (DAFS) [13], employs VIA/RDMA to achieve a better remote file service than NFS. Although some researchers are making progress [14, 15, 16], the design and implementation of DAFS is still in progress. Current DAFS versions involve complicated interface designs and implementation issues in order to support RDMA. They are not compatible with standard Unix or POSIX file I/O. Furthermore, DAFS has two major limitations: 1) it works for high-end storage systems such as Storage Area Network (SAN), not commodity computer systems (e.g., clustered systems) and 2) it deploys the RDMA data transfer service for client/server architecture (one to one mapping) between DAFS clients (e.g., Web or database application servers) and DAFS servers such as SAN, other than peer-to-peer architecture (many-to-many mapping), such as intra-cluster communication architecture. A federated DAFS project [17] studied how to make cluster-based direct access file servers scalable but it only works for small-scale DAFS servers up to eight nodes. This paper presents a new RAMS to incorporate RDMA into both commodity clustered server systems to achieve a much higher scalability (up to 256 nodes) and entire system performance.

6. Conclusions

This paper presented a novel solution to boost intra-cluster communication performance by creatively building a RDMA-based collaborative I/O cache system architecture for clustered server called RAMS. RAMS aims to cache the most recently used RDMA transfer processes for future usage. The paper makes two contributions to employ RDMA as a major intra-cluster communication technique: 1) develop a RDMA-based user-level buffer cache architecture on each cluster node to cache RDMA data transfer process; 2) develop three propagated update protocols to address the RDMA read failure problem. A comprehensive set of trace-driven simulation experiments have been conducted to evaluate system performance on RAMS, compared with a RPC-based data communication baseline system, in different sized clustered servers from 4 to 256 nodes. A cluster-based Web server is chosen as a sample platform for testing, along with three real-world traces. Experimental results show that three proposed update protocols of RAMS can reduce the RDMA read failure rate by 75%, and RAMS can boost the system throughput by more than 50%, compared with the RPC baseline system.

References

- [1] L. I. Enrique V. Carrera, Srinath Rao and R. Bianchini, "User-level communication in cluster-based servers," in *Proceedings of 8th International Symposium on High Performance Computer Architecture (HPCA'8)*, (Cambridge, Massachusetts), pp. 275–288, February 02 - 06 2002.
- [2] J. Liu, W. Jiang, P. Wyckoff, D. K. Panda, D. Ashton, D. Buntinas, W. Gropp, and B. Toonen, "Design and implementation of MPICH2 over InfiniBand with RDMA support," in *Proceedings of 18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*, (New Mexico, USA), April 2004.
- [3] J. Wu, D. K. Panda, and P. Wyckoff, "High Performance Implementation of MPI Derived Datatype Communication over InfiniBand," in *Proceedings of 18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*, (New Mexico, USA), April 2004.
- [4] A. Wagner, D. Buntinas, D. K. Panda, and R. Brightwell, "Application-bypass reduction for large-scale clusters," in *Proceedings of IEEE International Symposium on Cluster Computing (Cluster 2003)*, (Hongkong, China), pp. 404–411, Dec. 2003.
- [5] J. Nieplocha, V. Tipparaju, M. Krishnan, G. Santhanaraman, and D. Panda, "Optimizing mechanisms for latency tolerance in remote memory access communication on clusters," in *Proceedings of IEEE International conference on cluster computing (Cluster 2003)*, (Hongkong, China), pp. 138–147, Dec. 2003.
- [6] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "High-performance, portable implementation of the MPI Message Passing Interface Standard," *Parallel Computing*, vol. 22, no. 6, pp. 789–828, 1996.
- [7] "MPICH version 2." <http://www-unix.mcs.anl.gov/mpi/mpich2/index.htm>.
- [8] "InfiniBand Architecture Specification, Volumes 1 and 2, Release 1.0.a." <http://www.infinibandta.org>.
- [9] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su, "Myrinet: A gigabit-per-second local area network," *IEEE Micro*, vol. 15, no. 1, pp. 29–36, 1995.
- [10] "Intel Corporation, VIDF Virtual Interface (VI) Architecture Developer's Guide, Revision 1.1 Draft." <http://developer.intel.com/design/servers/vi/developer>, 2000.
- [11] "National energy research scientific computing center (nersc), M-VIA: A High Performance Modular VIA for Linux." <http://www.nersc.gov/research/FTG/via/>, September 2002.
- [12] F. M. Cuenca-Acuna and T. D. Nguyen, "Cooperative caching middleware for cluster-based servers," in *Proceedings of 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10)*, (San Francisco, CA, USA), pp. 303–314, August 2001.
- [13] "DAFS collaborative, Direct Access File System protocol version 1.0." <http://www.dafscollaborative.org/tools/spec.shtml>, 2001.

- [14] J. Chase, R. Kisley, A. Gallatin, D. Anderson, R. Wickremesinghe, and K. Yocum, "Direct access file system (DAFS) demo." Released at the DAFS Collaborative Developer's Conference 6/18/2001.
- [15] M. DeBergalis, P. Corbett, S. Kleiman, A. Lent, D. Noveck, T. Talpey, and M. Wittle, "The direct access file system," in *Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST-03)*, Mar. 21-Apr.2 2003.
- [16] K. Magoutis, "Design and implementation of a direct access file system (DAFS) kernel server for FreeBSD," in *Proceedings of the third BSDCon Conference 2002 (BSDCon-02)*, (Berkeley, CA), pp. 65–76, USENIX Association, Feb. 11–14 2002.
- [17] M. Rangarajan, S. Gopalakrishnan, A. Arumugam, and L. Iftode. Presented as a Work-in-Progress at the 2nd Usenix Conference on File and Storage Technologies (FAST 2003), March 2003.