Towards Tunable RDMA Parameter Selection at Runtime for Datacenter Applications

Kai Wang, Dian Shen, Fang Dong, Chengtian Zhang
Southeast University, Nanjing, China

ABSTRACT

Because of the low-latency and high-throughput benefits of RDMA, an increasing number of applications in datacenters are re-designed with RDMA to boost the performance. Among various low-level hardware primitives provided by RDMA, exposed as parameters of APIs, the application designers select and hardcode them to exploit all the performance benefits of RDMA. However, with the dynamic nature of datacenter application, the hardcoded and *fixed* parameter selection fails to take full advantages of RDMA capabilities, which can cause up to 35% throughput performance loss. To address this issue, we present a *tunable* RDMA parameter selection framework, which allows parameter tuning at runtime, adaptive to the dynamic application and server status. To attain the native RDMA performance, we use a lightweight decision tree to reduce the overhead of RDMA parameter selection. Finally, we implement the *tunable* RDMA parameter selection framework with native RDMA API to provide a more abstract API. To demonstrate the effectiveness of our method, we implement a key-value service based on the abstract API. Experiment results show that our implementation has only a very small overhead compared with the native RDMA, while the optimized key-value service achieves 112% more throughput than Pilaf and 66% more throughput than FaRM.

KEYWORDS

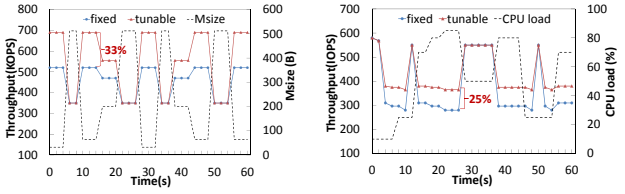
RDMA, Parameters, Tunable

1 INTRODUCTION

RDMA provides abundant parameters related to low-level details of hardware [4], which are exposed as API for applications. The application designers have to select these parameters case by case and hardcode them with RDMA API. As a result, these applications all transmit data with a *fixed* parameter selection framework.

However, the hardcoded and *fixed* parameter selection fails to take full advantages of RDMA capabilities in datacenter environments, where application features and resource states keep changing during the execution of applications. For example, 1) Application features. Message size will affect the choice of inline/non-inline mode. Compared with non-inline mode, inline mode can provide lower latency and higher throughput, but only for data packets within certain size, so inline/non-inline mode needs to be switched according to the size of the data packages. 2) Resource states. The change of CPU resources has a certain impact on the polling strategy. Because there are mainly two kinds of polling strategies: busy polling and event-triggered polling. And the busy polling strategy can provide lower latency and higher throughput compared with event-triggered polling at the expense of corresponding higher requirements for CPU resources, hence the polling strategy need to be adjusted in pace with the CPU usage status of the server.

As a result, the hardcoded and *fixed* parameter selection is unable to make the best use of RDMA capabilities, which will result in up to 35% throughput performance loss as shown in Figure 1.



(a) Inline/non-inline (b) Polling strategy

Figure 1: Throughput loss under fixed parameter setting. During the execution of applications, message size varies from 32B to 512B, When msize is small, fixed setting (non-inline) results in 520 KOPS. If switching to inline, the throughput can increase to 690 KOPS. CPU load varies from 10% to 85%, When CPU load is high, fixed setting (busy polling) results in 280 IOPS. If switching to event-triggered polling, the throughput can increase to 365 IOPS.

This paper presents a *tunable* RDMA parameter selection framework, which allows parameter tuning at runtime, adaptive to the dynamic application and server status to take full advantages of RDMA capabilities. We find out some relationship results between different RDMA parameters and application features, resource states based on theoretical analysis and comparative experiments. Furthermore, to attain the performance of native RDMA, we convert these relationship results into a lightweight decision tree based on a decision tree generation algorithm. Finally, we implement the *tunable* RDMA parameter selection framework with native RDMA API to provide a more abstract API called *CommonRDMA*.

2 SYSTEM DESIGN

We propose a *tunable* RDMA parameter selection framework, which allows parameter tuning at runtime, adaptive to the dynamic application and server status to take full advantages of RDMA capabilities. As illustrated in Figure 2, *CommonRDMA* is designed on top of the native RDMA APIs and a lightweight Decesion Tree (DT) that is used to select the RDMA parameters at runtime.

We find out some relationship results between different RDMA parameters and application features, resource states based on theoretical analysis and comparative experiments, including connection type, signal/unsignal, verbs, inline/non-inline, poll strategy, etc. It is worth noting that we pay attention to presenting the framework of *tunable* RDMA parameter selection, however, the RDMA parameters, application features and resource states may not comprehensive enough. As a result, some new parameters, application features and resource states could be added to this framework.

3 PRELIMINARY EVALUATION

In this section, we will firstly compare the performance gap between *CommonRDMA* and native RDMA APIs, and then we implement a key-value system called CKV to evaluate the effectiveness of *CommonRDMA* on concrete applications. Since we primarily consider the RDMA parameters selection regardless of the data structure of

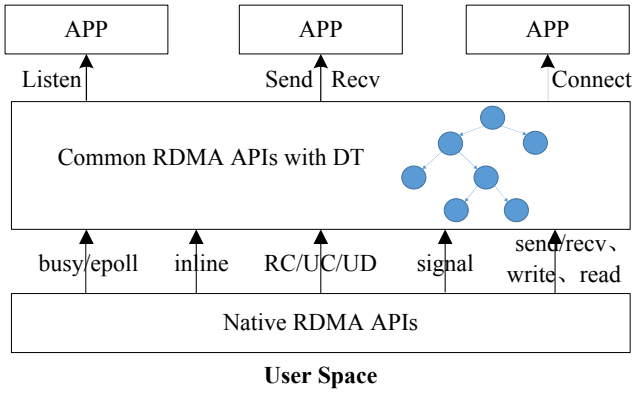
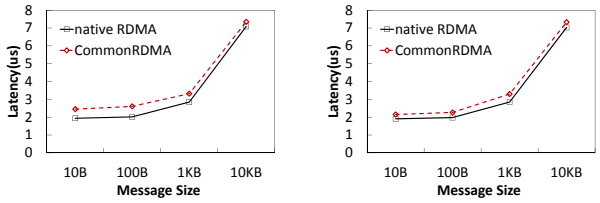


Figure 2: CommonRDMA Architecture



(a) SEND/RECV latency (b) WRITE latency

Figure 3: Performance comparison of CommonRDMA and native RDMA

the key-value system, we implement Pilaf [5] and FaRM [2] according to their respective RDMA parameters with native RDMA APIs based on Anuj Kalia's implementation [3]. Finally, we will show the performance comparison results between CKV, FaRM and Pilaf.

3.1 API Performance Comparison

To evaluate whether our *CommonRDMA* API could have a similar performance as native RDMA API, we test the latency of WRITE and SEND/RECV verbs with different message size.

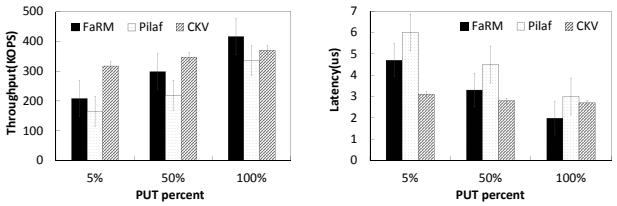
First, we directly use the tools (ib_write_lat, ib_send_lat) of Perftest Package to acquire the latency of native RDMA API. Then we implemente a simple benchmark based on *CommonRDMA* API that calculate the latency of each transmission. We collect the result of 1000 circles and calculate the average value of latency. As shown in Figure 4, although with encapsulation and decesion tree overhead, the extra latency introduced by *CommonRDMA* API is only about 0.5 us that means it has only a slight overhead over native RDMA.

3.2 Application Performance Comparison

To evaluate the ease-of-use and effectiveness of *CommonRDMA* on concrete applications, we implement a key-value system called CKV based on *CommonRDMA* API with only about 100 lines of code. We run all our throughput and latency experiments on two machines, one for client and the other for server.

3.2.1 Throughput Comparison. We now compare the end-to-end throughput of CKV against the simplified implementations of Pilaf and FaRM.

As shown in Figure 5(a), we compare the throughput of these systems for both read-intensive (95% GET, 5% PUT) and write-intensive (50% GET, 50% PUT) workloads for 48-byte items which is representative of real-life workloads [1].



(a) Throughput comparison (b) Latency comparison

Figure 4: Performance comparison for 48 byte key-value items

In CKV, both read-intensive and write-intensive workloads achieve around 350Kops throughput regardless of the workload composition since PUT and GET operation both need just one round trip with WRITE verb. By contrast, Pilaf and FaRM's throughput is related to the workload composition. For the read-intensive workloads, Pilaf only achieves 165 Kops throughput which is around 50% of the CKV's throughput and FaRM's throughput is about 60% of the CKV's throughput because of their multiple round trips with READ verb. However, with the proportion of PUT operation increasing, the throughput of Pilaf or FaRM also increases but it is still less than our system. As shown in Figure 5(a), CKV achieves 60% more throughput than Pilaf and 17% more throughput than FaRM for write-intensive workloads.

3.2.2 Latency Comparison. We now compare the end-to-end latency of CKV against the simplified implementations of Pilaf and FaRM.

Like the throughput, CKV's latency remains about 2.9 us regardless of the workload composition because CKV requires only one network round trip for any operation. FaRM and Pilaf require one round trip for PUT operation but require multiple round trips for GET operation. This causes their GET latency to be higher than the latency of a single RDMA READ. As a result, from the Figure 5(b), we can see that both FaRM and Pilaf achieve a worse latency than CKV. In detial, FaRM's latency is 51% more than CKV and Pilaf's latency is 93% more than CKV.

Finally, for the 100% PUT workloads, the latency of these three systems is similar because PUT operation always require only one round trip. Also, FaRM achieves a better latency than CKV due to the encapsulation and decesion tree overhead of *CommonRDMA*.

4 CONCLUSION AND FUTURE WORK

This paper presented a *tunable* RDMA parameter selection framework, which allows parameter tuning at runtime, adaptive to the dynamic application and server status. As for future work, we consider implementing or re-designing more complex applications (deep learning, big data) with *CommonRDMA* to evaluate the performance of *CommonRDMA* and finding out more application features, resource states to enrich the DT.

REFERENCES

[1] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny. Workload analysis of a large-scale key-value store. In *Proceedings of the SIGMETRICS'12*, 2012.
[2] A. Dragojević, D. Narayanan, O. Hodson, and M. Castro. Farm: Fast remote memory. In *Proceedings of the NSDI '14*, 2014.
[3] A. Kalia, M. Kaminsky, and D. G. Andersen. Using rdma efficiently for key-value services. In *Proceedings of the SIGCOMM '14*, 2014.
[4] A. Kalia, M. Kaminsky, and D. G. Andersen. Design guidelines for high performance rdma systems. In *Proceedings of the ATC '16*, 2016.
[5] C. Mitchell, Y. Geng, and J. Li. Using one-sided rdma reads to build a fast, cpu-efficient key-value store. In *Proceedings of the ATC '13*, 2013.