Can RocketBufs Make an Impact?

Bryant Curto
University of Waterloo

Manoj Adhikari
University of Waterloo

Shahzaib Ali
University of Waterloo

Abstract

RocketBufs is a framework for building high-performance, in-memory Message Oriented Middleware (MOM) systems produced by researchers from the University of Waterloo. The authors claim that RocketBufs provides MOM developers access to modern data center networking technologies that can be easily utilized to increase performance.

In this study, we repeat the performance analysis conducted in the original paper. Then, we discuss the challenges of adapting Redis, a popular industry grade MOM system, to run on top of RocketBufs. This provides some insights into challenges for building or adapting other MOM systems to use RocketBufs.

Our work supports the argument that RocketBufs is a unique solution to an important problem. However, some hurdles exist that prevent it from being easily integrated into existing MOM systems.

1 Introduction

Message-Oriented Middleware (MOM) systems provide a clean method of communication between disparate software entities. Also referred to as publish/subscribe systems, they are any middleware infrastructure that provide messaging capabilities [1]. They are a popular class of software systems designed to support loosely-coupled messaging in modern distributed applications [5]. MOM systems are the foundation upon which modern distributed enterprise systems are built. Companies that have high demands for message sharing can benefit from the improved performance of MOM systems.

One approach of improving the performance of MOM systems is to leverage modern and emerging data center networking technologies, especially those that offer kernel-bypass features to enable high-throughput and/or low latency communication such as Remote Direct Memory Access (RDMA) [12]. Similar to RDMA, there are other modern network communication technologies such as DPDK and TCPDirect which can significantly improve the performance of MOM systems. Most MOM systems use kernel-based TCP for communication and don't utilize modern data center networking technologies. This is because the API's and abstractions provided by each of these technologies are very different from TCP which makes them very challenging to adapt to.

RocketBufs [5] is a framework designed to improve accessibility of modern and emerging networking technologies for MOM systems. It aims to assist developers in making use of these networking technologies and seamlessly switch between technologies as needed.

The claims of the RocketBufs paper regarding its performance and ease of use have not been evaluated by anyone other than the original authors. We are the first to attempt a reproduction of their experiments. We go beyond a reproduction to consider more real-world scenarios. In achieving this endeavor, we consider the challenges associated with adapting Redis, an industry grade MOM system, to use RocketBufs.

Through our work, we make the following contributions:

1. We reproduce a subset of the performance analysis conducted in the RocketBufs research paper. We evaluate the performance of RBMQ, a proof-of-concept MOM system built on top of RocketBufs, using TCP.

2. We discuss the challenges faced in building or adapting an MOM system to run on top of RocketBufs by considering the challenges in adapting Redis, an industry grade MOM system.

2 Background and Related Work

2.1 Background

A Message-Oriented-Middleware (MOM) System is any middleware infrastructure that provides messaging capabilities. A client of an MOM system can send messages to, and receive messages from, other clients of the messaging system with one or more servers acting as intermediaries in message delivery [2]. We refer to sending clients as publishers, receiving clients as subscribers, and intermediary servers as brokers.

RocketBufs is a framework which provides infrastructure for building high-performance, in-memory MOM systems. It provides memory-based buffer abstractions and APIs designed to work efficiently with different network communication technologies (e.g., TCP and RDMA). It offers MOM system developers easy access to modern network communication technologies [5], and thereby any performance improvements that come about through use of these technologies. In the original RocketBufs paper, a publish-subscribe message queuing system named RocketBufs Message Queue (RBMQ) is implemented using RocketBufs and its performance is compared with a popular MOM system called Redis.

Redis is a widely used, production quality, in-memory data structure store that can be used as an MOM System [7]. Companies like Engine Yard, Github, Craigslist, Disqus, Digg, and Blizard are part of the growing list of Redis adopters [9].

2.2 Related Work

2.2.1 Networking Technology Libraries

Numerous APIs are available to MOM system developers that can be used to take advantage of modern networking technologies.

Libraries such as rsocket and libfabric provide new APIs and access to modern networking technologies such as RDMA. Further, they attempt to perform RDMA-based optimizations [5]. However, these solutions do not provide a comparable framework to RocketBufs.

Mellanox's Messaging Accelerator (VMA) requires no application change because it provides standard socket TCP, UDP (Unicast, Multicast) to the application layer. In addition it enables Kernel bypass which reduces kernel overhead by creating direct application to network adapter access. It purports benefits such as lower latency, higher throughput, and lower CPU utilization [10]. However, this solution also doesn't provide a comparable framework to RocketBufs and doesn't enable the ability to easily switch to a different data center networking technology.

The key differences between other libraries and RocketBufs is that RocketBufs provides a middleware with much higher level APIs and abstractions that are designed to naturally and explicitly support the building of a wide range of efficient message-oriented systems. We could find no other frameworks that allow applications to utilize modern data center networking technologies and permit the transition to another networking technology with no alterations to application code.

2.2.2 Redis Network Communication

Redis has previously been adapted to utilize modern data center networking technologies.

Tang et al. show that adapting Redis to use RDMA instead of TCP sockets can improve performance by increasing throughput, reducing latency and reducing CPU utilization [12]. However, since Redis is adapted directly to make use of RDMA, taking advantage of a different networking technology (e.g., DPDK) would require a completely new implementation. In contrast, once RocketBufs is able to support a new networking technology, any MOM system developer wishing to make use of the new technology need to only change a configuration file.

Qi et al. design a high performance Redis system by replacing the traditional Unix socket interface with Fast Event Driven RDMA RPCs (FeRR) [11]. The authors make additional modifications to Redis such as optimizing the Redis serialization protocol. Further, they replace Redis' single thread framework with a parallel task engine. These additional modification to Redis make it difficult to distinguish what benefits were gained through using RDMA and what benefits were due to other changes. This paper similarly focuses on adapting Redis to use RDMA directly and suffers from the same inability to adapt to other networking technologies.

3 Experimental Setup

3.1 Hardware

The hardware for our performance evaluation is comprised of a cluster of thirteen machines. The machine running the broker contains an Intel Xeon E5-2660 v3 CPU with 10 cores clocked at 2.60GHz and 512 GBs of RAM. We call this machine the broker machine. The remaining twelve machines in the cluster are used to run clients (i.e., publishers or subscribers) and are called client machines. These machines contain an Intel Xeon CPU D-1540 with 8 cores clocked at 2.00 GHz and 64 GB of RAM. Four 40 Gbps NICs are configured on the broker machine to connect to 4 mutually exclusive subsets of client machines in the cluster connected via their own single 40 Gbps NIC. As a result, the broker has

up to 160 Gbps of bi-directional bandwidth available to communicate with clients. Each subset is configured to communicate over an exclusive network. By designating each subset of machines be used for either publishers or subscribers, we avoid publisher and subscriber messages interfering with one another. We did not saturate the NICs for any of our experiments. All machines run Ubuntu 18.04.1 with Linux kernel version 4.18.0.

Note that the hardware specifications are nearly identical to those used in the original performance evaluation of RBMQ [5, 4] with the exception of OS version (Ubuntu 18.04.2). As a result, we expect similar results.

3.2 Setup

3.2.1 Redis

In our evaluation, we use Redis version 6.0.9. Each Redis publisher instance spawns ten threads, each of which continuously publishes messages to one of ten topics. Redis uses a Request/Response Protocol [8]. As a result, each publisher thread publishes a message to a topic by submitting the message to the Redis broker, waiting for acknowledgement, and then submitting the next message.

Each Redis subscriber instance spawns ten threads such that each thread subscribes to a single topic. Once subscribed, each subscriber thread simply counts the number of messages it receives from the broker. We compare RBMQ with Redis instead of another MOM system (e.g., RabbitMQ) because of the similarity in throughputs of the systems reported in the RocketBufs paper.

3.2.2 RBMQ

Each RBMQ publisher instance spawns a multiple of ten threads, each of which continuously publishes messages using it's own personal RocketBufs buffer. RBMQ does not use a Request/Response Protocol. Publishers handle responses from the broker asynchronously. As a result, we configured publishers to publish messages at a constant rate of 50,000 messages per second, which we found to be consistent with Redis on average for our selected message sizes.

Each RBMQ subscriber instance spawns ten threads such that each thread subscribes to a non-overlapping subset of RocketBufs buffers to which messages are published. Once subscribed, each subscriber thread simply counts the number of messages it receives from the broker.

4 Evaluation

4.1 Methodology

Originally, four different types of RBMQ experiments are performed. These experiments vary in the transport protocol configurations that they use. These transport protocol configurations include client and broker both communicating via TCP with flow control enabled (RBMQ-tcp-tcp), via TCP with flow control disabled (RBMQ-tcp-no-fc), and via RDMA (RBMQ-rdma-rdma). The final configuration is publisher and broker communicating via TCP while broker and subscriber communicate via RDMA (RBMQ-tcp-rdma). In our reproduction, we evaluate the performance of RBMQ with clients communicating with the broker using TCP with flow control enabled (RBMQ-tcp-tcp). We do not evaluate the performance of RBMQ using TCP with flow control disabled (RBMQ-tcp-no-fc) because it is only used as a means of understanding the overhead of flow control [4]. Further, we do not evaluate the performance of RBMQ using RDMA (RBMQ-rdma-rdma and RBMQ-tcp-rdma) because it would be more difficult to compare both systems as Redis does not natively support use of RDMA. Most importantly, RBMQ using TCP with flow control enabled offers the most straight forward method of comparing performance as Redis uses TCP.

In our experiments, we use separate hosts for broker, publisher, and subscriber instances. Each publisher instance uses multiple threads to produce messages of a fixed size. Each subscriber instance subscribes to all publisher topics, thereby consuming all data produced by the publishers.

Prior to performing our experiments, we follow the steps described in the original evaluation [5, 4]. We disable the Linux irqbalance daemon, which is enabled by default in most current Linux distributions. Further, we disable hyperthreading on each machine to ensure that our measurements are not negatively influenced by threads running on the same physical core. Each experiment is run for a minimum of 150 seconds where data is collected for only the latter 120 seconds. This is to avoid variability caused by experiment initialization from influencing our results.

During our evaluation, we found our measurements to be highly sensitive to the number of publisher processes used. Using too few publishers results in the broker's resources not being saturated. However, using too many publishers can result in overhead from message buffering and flow control. In order to determine the optimal number of publishers, we first run the experiment with a sufficient number of publishers to saturate the cores of the broker's machine. We then rerun the experiment,

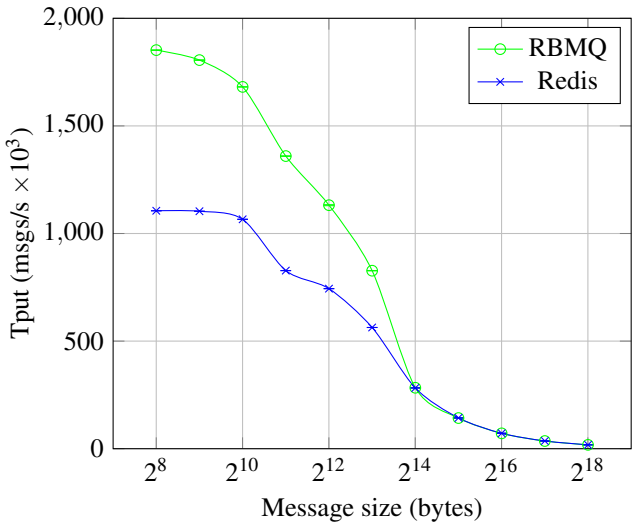

Figure 1: Throughput of RBMQ and Redis with zero subscribers and 95% confidence intervals.

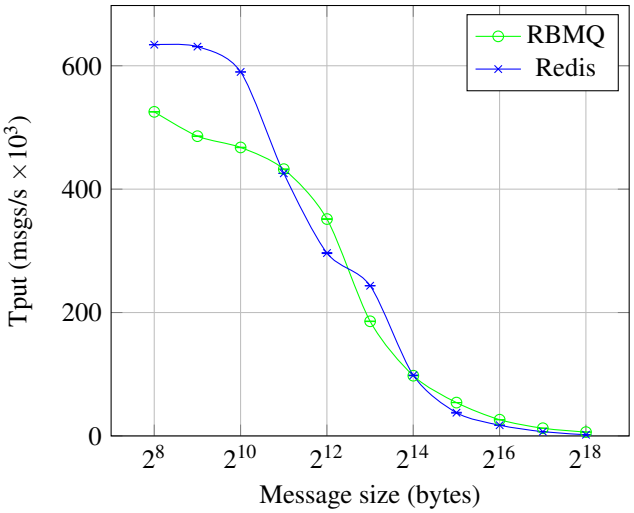

Figure 2: Throughput of RBMQ and Redis with one subscriber and 95% confidence intervals.

performing a binary search of the possible number of publishers until arriving at the number of publishers for which we observed best throughput.

4.2 Throughput with 0 Subscribers

Figure 1 plots the maximum throughput of messages that can be processed by the Redis and RBMQ brokers with no subscribers. In this experiment, publishers send messages to the broker which are immediately discarded. The plot also includes 95% confidence intervals, which are difficult to see as a result of their size. This experiment serves as a means of understanding how our experimental setup compares to that of the original.

The figure shows that RBMQ performs noticeably better than Redis for message sizes in range 2^8 to 2^{13} bytes. For message sizes of 2^8 bytes, RBMQ's throughput is approximately 67% greater than that of Redis. For message sizes in range 2^{14} to 2^{18} bytes, the performance of RBMQ and Redis is nearly identical.

Comparing our findings with those presented in the RocketBufs paper [5] and Hoang's thesis [4], we take note of three key things. First, our measurement of Redis' throughput is almost identical to that presented in the previous evaluation. Second, our measurement of RBMQ's throughputs for messages of size 2^8 to 2^{13} bytes is much higher than those originally observed. In the original evaluation, this version of RBMQ is reported to have a maximum throughput of approximately 900 thousand messages per second for messages of size 2^8 and 2^9 bytes. The throughput then drops as low as approximately 750 thousand messages per second for messages of size 2^{12} bytes. For messages of size 2^8 bytes, we measure RBMQ's throughput to be over 1.85 million messages per second, which is over double that originally measured. Third, for message sizes in range 2^{14} to 2^{19}, our throughput measurement of RBMQ is nearly identical to that presented in the original evaluation.

In the original evaluation, the number of publisher threads are steadily increased until the throughput stopped increasing. At most 20 publisher threads are used to saturate the broker. We believe that our approach in determining the optimal number of publishers to fully saturate the broker for messages of a given size (previously described) enables us to eke out additional performance from the RBMQ broker.

4.3 Throughput with 1 Subscribers

Figure 2 plots the maximum throughput of messages delivered by the Redis and RBMQ broker to a single subscriber. The figure also includes 95% confidence intervals, which are difficult to see as a result of their size.

The figure shows that Redis performs better than RBMQ for messages of size in range 2^8 to 2^{10} bytes. RBMQ's throughput is as much as 23% lower than that of Redis for messages of size 2^9 bytes. For messages of size 2^{11} bytes and larger, Redis and RBMQ perform similarly except for messages of size 2^{12} bytes and 2^{13} bytes. For the prior, RBMQ has a throughput of approximately 19% higher than that of Redis. For the latter, RBMQ has a throughput of approximately 24% lower.

Comparing our findings to those presented in Hoang's thesis [4], we note the following. First, our measured performance of Redis is noticeably higher for messages of size 2^8 to 2^{10} bytes. The throughput is originally re-

ported to be approximately 500 thousand messages per second for this range of message sizes. In both evaluations, throughput is approximately the same for messages of size 2^{11} bytes. Second, for values from 2^{12} to 2^{18} bytes, we observe a drop in Redis' throughput ranging from approximately 30% for the smaller message sizes and 70% for the larger when compared to the original evaluation of Redis. Third, we measure the performance of RBMQ to be higher for messages of size 2^8 to 2^{11} bytes by up to 30% compared with the original evaluation of RBMQ. Forth, for messages of size 2^{12} bytes, we observe similar throughput of RBMQ, but observe a decrease in throughput of up to 75% percent for message sizes in range 2^{13} to 2^{18} bytes.

As before, we believe that some of the disagreement with the findings in the original study may be attributed to the method we used to determine the number of publishers. Further, we think that differences could have been caused due to a different Redis version being used in the original evaluation (the version is not specified). To our knowledge, our setup and hardware is nearly identical to that described in the original paper. As a result, we would expect to observe nearly identical results.

The pattern in our measured throughput for Redis and RBMQ is similar to that in the original study for larger message sizes but the measured values are different. This suggests that our experimental setup differed from that of the original in one or more ways resulting in lower observed throughputs.

4.4 Assumptions

In performing a reproduction of this work, we make several assumptions due to the lack of experimental artifacts.

The RBMQ subscriber should be run using 10 threads. From the available code, we were able to infer that Redis subscribers utilized ten threads for handling messages from the broker. As a result, we made the assumption that the RBMQ subscriber should use the same (Figure 2 shows results using this setup). In order to validate this assumption, we measured the throughput for a number of message sizes where the RBMQ subscriber made use of only one thread. However, we found that this change resulted in the subscriber becoming a bottleneck such that, no matter the number of publishers, we were not able to saturate the broker.

Only one RBMQ broker should be run on the broker machine. We determined that the number of brokers must be at least equal to the number of subscribers. However, we were not able to determine how more or fewer brokers would affect the evaluation.

The RBMQ broker should send responses to the publisher after each message is successfully published. This option is disabled by default in RBMQ. However, Redis follows a Request/Response Protocol [8]. Leaving this option disabled would result in a weakening of the comparison of the two broker systems.

The Redis broker is composed of ten distinct, non-communicating Redis server instances, which was determined from the experimental artifacts and feedback from Hoang. We validate this setup in our benchmark with zero subscribers (see Figure 1). However, this setup appears unrealistic as the lack of communication between the Redis instances results in no coordination between instances. This configuration may not serve as a valid comparison for RBMQ. Without the overhead of coordination, measured throughputs of the Redis broker would be higher. To validate this, we create a Redis broker composed of a single Redis server instance making use of several threads for communicating with clients. We measure the maximum throughput of the Redis broker with one subscriber for a subset of message sizes. The multithreaded Redis broker is not able to achieve the same throughput as the multi-instance Redis broker. The results of this experiment can be seen in Figure 3 where the multithreaded Redis broker utilized 128 threads for communicating with clients.

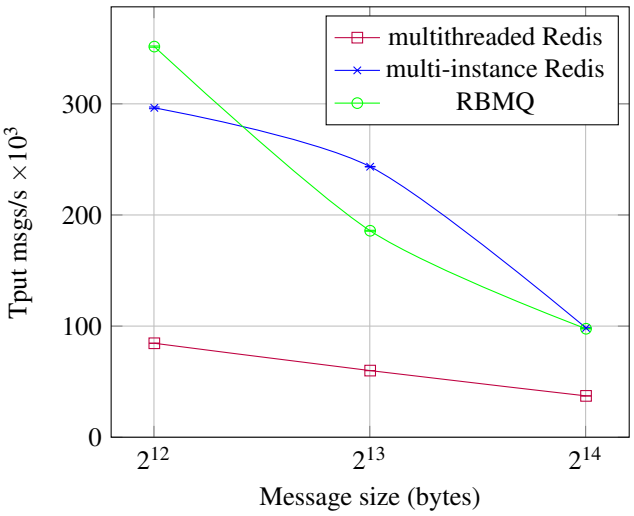

Figure 3: Throughput of RBMQ broker, multithread Redis broker, multi-instance Redis broker with one subscriber and 95% confidence intervals.

5 Using RocketBufs

This section presents the observed challenges in adapting Redis, a popular MOM system, to run on top of RocketBufs. To our knowledge, we are the first to provide feedback on the RocketBufs framework from the point of view of MOM developers. We leave the job of performing this adaptation to future work.

The first challenge in adapting Redis to use Rocketbufs is the framework's lack of a C interface. Redis is written in C, as too are many high performance applications as a result of the language's low overhead. Such an interface might be helpful in attracting MOM system developers to make use of the framework.

The second challenge is RocketBufs' asynchronous API. The RocketBufs interface is asynchronous and leaves completion events to be handled by a registered callback function. Redis has a synchronous design such that each client is managed using a single thread of execution [6]. Adapting Redis to use RocketBufs would potentially require adding thread synchronization structures. Asynchronous communication with clients would require reworking the system to handle communication failures occurring at any point after invocation. Building or adapting an MOM system to use RocketBufs will require an asynchronous model, which tends to be more difficult to implement and reason about. However, these concerns may apply primarily to Redis as many high performance applications already use an asynchronous model [3].

6 Conclusion

RocketBufs has the potential to enable MOM systems to quickly and easily make use of new and emerging network communication technologies, resulting in high-throughput and/or low latency network communication. In our study, we reproduce a subset of the performance evaluations of RBMQ, a MOM system built on top of RocketBufs. We are not able to validate the RocketBufs paper's claims about performance as a result of a lack of experimental artifacts. While some challenges exist in building or adapting MOM systems to use RocketBufs, it is our belief that RocketBufs is a unique solution to an important problem.

7 Acknowledgments

We would like to acknowledge Professor Brecht for his help and guidance in steering us toward a completed paper. We would also like to acknowledge Lori Paniak for helping set up the machines for experimentation. Without his help, we would have no results.

References

[1] CURRY, E. Message-oriented middleware. *Middleware for communications* (2004), 1–28.

[2] CURRY, E. *Message-Oriented Middleware*. John Wiley & Sons, Ltd, 2004, ch. 1, pp. 1–28.

[3] ENBERG, P., RAO, A., AND TARKOMA, S. Partition-aware packet steering using xdp and ebpf for improving application-level parallelism. In *Proceedings of the 1st ACM CoNEXT Workshop on Emerging In-Network Computing Paradigms* (New York, NY, USA, 2019), ENCP '19, Association for Computing Machinery, p. 27–33.

[4] HOANG, H. Building a framework for high-performance in-memory message-oriented middleware. Master's thesis, University of Waterloo, 2019.

[5] HOANG, H., CASSELL, B., BRECHT, T., AND AL-KISWANY, S. Rocketbufs: A framework for building efficient, in-memory, message-oriented middleware. In *Proceedings of the 14th ACM International Conference on Distributed and Event-Based Systems* (New York, NY, USA, 2020), DEBS '20, Association for Computing Machinery, p. 121–132.

[6] LABS, R. How fast is redis? `https://redis.io/topics/benchmarks`. Accessed: 2020-12-10.

[7] LABS, R. Introduction to redis. `https://redis.io/topics/introduction`. Accessed: 2020-12-08.

[8] LABS, R. Using pipelining to speedup redis queries. `https://redis.io/topics/pipelining`. Accessed: 2020-12-10.

[9] MACEDO, T., AND OLIVEIRA, F. *Redis Cookbook: Practical Techniques for Fast Data Manipulation*. " O'Reilly Media, Inc.", 2011.

[10] MELLANOX. What is vma?, Dec 2018.

[11] QI, X., HU, H., WEI, X., HUANG, C., ZHOU, X., AND ZHOU, A. High performance design for redis with fast event-driven rdma rpcs. In *International Conference on Database Systems for Advanced Applications* (2020), Springer, pp. 195–210.

[12] TANG, W., LU, Y., XIAO, N., LIU, F., AND CHEN, Z. Accelerating redis with rdma over infiniband. In *International Conference on Data Mining and Big Data* (2017), Springer, pp. 472–483.