Failure Detection over RDMA

Pavel Georgiev

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2019

Abstract

The collection and exploitation of data are becoming of utmost importance for an increasing number of companies. Their data-driven services are the keystone in the way they interact with their customers and the value they can provide. To optimize their performance and throughput in the transfer of the data the use of the state-of-the-art networking solution Remote Direct Memory Access (RDMA) is getting more prevalent. Furthermore, the number of components in data centres keeps growing, which makes failures more of the expectation rather than the exception. To achieve fault tolerance in those circumstances a failure detection implementation over RDMA is needed.

In this paper, we explore the implementation of such a failure detector. The paper studies the methods chosen for detecting the crashes and reaching consensus over a suspected component failure within an asynchronous and distributed setting. It also outlines the details around the integration of RDMA methods for communication between nodes. Finally, the paper analyses the performance difference between the implementation using RDMA for communication versus the one using TCP/IP.

Acknowledgements

Acknowledgements go here.

Contents

Chapter 1

Introduction

Big Data market revenues are continuously growing with a projected increase of $42B in 2018 to $103B in 2027 [15] (see Figure A.1). Because of the amount of data companies collect and exploit, they strive for the most optimal data transfer and computation speeds they can achieve. The amount of data they work with not only directly ties with an increase in their revenue but also helps them scale their solutions to a global level.

It has been confirmed that Moore's law is coming to its natural end [6, 16] so we can no longer rely on big improvements in performance by vertical scaling a machine by upgrading its hardware. However, the shift towards distributed computing allowed for the increase in computational power to continue. Gone are the days of a single server within a data centre. By following the trends on Top500.org [1] we can see that it has been the case for a long time that the work of a single logical service is done over many machines grouped in physical clusters. From data storage [5] to batch or stream data processing [2], the number of problems that could be solved in a distributed manner is countless. In order to reach peak throughput and minimal latency, companies are placing their focus on optimizing the intra-cluster and inter-cluster data transfer process. Regular TCP/IP connections, without any optimizations or specialized hardware, can become a bottleneck for the efficiency of a distributed system. With Remote Direct Memory Access (RDMA) implementations having the potential for better throughput, lower latency and less central processing unit(CPU) interrupts [11] it is becoming clear why the use of RDMA methods is getting more and more prevalent. Wider adoption is enabled by the development of the technology and vendors releasing RDMA-compliant interconnecting hardware [4]. From network architecture standards like InfiniBand [12] to using RDMA over Converged Ethernet (RoCE) [17] to Internet-wide RDMA protocols like iWARP [9], there are an increasing number of ways of integrating RDMA into data centres with different fabrics and protocols.

Because of the shift toward distributed computing, the number of components within a system has substantially grown, which further increases the rate of expected failure. After analysis on data of 22 high-performance computing (HPC) systems, Gibson and Schroeder [14] suggest that the failure rate grows in a rate proportional to the number of processor chips in the system. With some systems in their data showing a rate of 1100 failures per year, it is almost expected that a node will fail during execution of

a process. In order to maintain the expected performance of the system, there should be measures in place to make sure the system is fault tolerant and that a failure of one node won't bring the whole service down.

Similarly, the same principles hold on comparable systems that employ RDMA methods for their data transfer. Failure detection mechanisms and properties have been studied before [13, 14, 15, 16] but little work has been done on implementing one over RDMA with most of the work being done focusing on Replicated State Machines (RSM) [13].

1.1 Goals

The main goal is to implement a failure detector over RDMA.

In order to achieve this and evaluate the results the following contributions have been made:

- Implementation of a micro testing framework to emulate a multi-node setup in a distributed system environment
- Implementation of failure detection over TCP/IP
- Evaluation of adaptive timeout algorithm in the failure detector implementation
- Integration of RDMA methods of communication in the failure detector implementation
- Analysis of the performance difference between RDMA and TCP/IP based failure detector implementation

1.2 Report outline

- **Chapter 2**: Gives the necessary background needed to understand the nature of the work presented in the paper. Specifically, it introduces important concepts about failure detection (Section 2.1) and RDMA (Section 2.2).
- **Chapter 3**: Describes the implementation of a failure detector. It goes into details about the class of failure detector needed, implementation of dynamic timeout and choice of consensus protocol.
- **Chapter 4**: Describes the way the implementation is altered to work with RDMA.
- **Chapter 5**: Analyses the performance of the difference in performance of the failure detector when communication is done over RDMA versus when it is done over TCP/IP.
- **Chapter 6**: Summarizes the work that is done and observations have been made in the process of researching and implementing the desired outcomes. It also outlines possible ways the implementation could be improved or expanded upon.

1.3 Tools

The source code for the failure detector has been written in C[1] language. In order to streamline the development workflow, the open-source build process management software CMake[2] language has been used. The library ZeroMQ[3] has been utilised for the initial development of the failure detector over TCP/IP. This is a high-performance messaging library that is aimed at distributed applications. Another tool that helped in the development was Docker[4]. It allowed for an easy way to simulate a distributed network.

[1] https://en.wikipedia.org/wiki/C_(programming_language)
[2] https://cmake.org/
[3] http://zeromq.org/
[4] https://www.docker.com

Chapter 2

Background

For the rest of the paper, if not mentioned otherwise, every system discussed would be a distributed system that consists of a finite set of nodes n. For simplicity would assume that every node runs a single process. Therefore, we have a set of processes $\pi = \{p_1, p_2, p_3 \dots p_n\}$. Also, we would assume that every node in the system is interconnected with the rest of the nodes by a reliable channel and can communicate with them by sending messages.

We would assume an asynchronous model of distributed computing for the systems we are discussing, which means that no timing assumptions are made. This allows for a great general model because we need not concern ourselves with delays caused by computations, network congestion, scheduled delays etc. Although we don't measure physical time, for convenience we would use a monotonically increasing virtual clock τ that increases at every event occurrence.

2.1 Failure Detection

In this section we are not concerned with the software implementation of a failure detector or the hardware that is running underneath; for implementation details refer to Chapter 3. Instead, we focus on the properties a failure detector has and discuss ways to detect failures in an asynchronous system.

2.1.1 Consensus Problem

In distributed computing, it is important to be able to achieve system reliability even in a presence of a failed component. This requires processes to agree on a certain aspect – computed value, elected leader, clock synchronization etc. Different approaches decide the consensus value differently, but all consensus protocols end with a unanimous decision at the end of their run. The consensus problem is bounded by the following properties [3]:

- **Termination**: every correct process eventually decides upon a value
- **Uniform integrity**: every process decides at most once

- **Agreement**: no two correct processes decide differently
- **Uniform validity**: if a process decides on a value v, then v was proposed by some process

There is also an extension of the general consensus problem called the *Uniform Consensus* problem, which enforces that no two processes decide differently.

2.1.2 Impossibility Result

An important thing to note is the proof by Fischer et al. [7], which states that in an asynchronous distributed system, even a single failed process can render the reach of consensus *impossible*. Simply put, this stems from the fact that in an asynchronous system we have no way of knowing whether a process has crashed or it is taking a long time to communicate with the rest of the nodes.

2.1.3 Unreliable Failure Detectors

There are several ways to remedy the impossibility of reaching consensus, explained in Subsection 2.1.2, by making a minimal set of assumptions on top of the asynchronous model. We would focus on the unreliable failure detector abstraction described by Chandra and Toueg [3], which allows us to build reliable distributed systems. To combat the difficulty of determining if a process has crashed or just behaving abnormally slow, the authors propose the introduction of failure detectors that can make mistakes.

In this abstraction, we have a set of distributed failure detectors. That set consists of local failure detector modules attached to every process in our system, which monitors the status of the rest of the processes in the system. Every failure detector module maintains a list of suspected crashes and can append or delete suspected processes from that list. For example, node p_i can suspect that node p_j has crashed and put it in its list of suspects. If later p_i decides that a mistake has been made it can remove p_j from the list. Also, two processes can have different lists of suspected failures.

Failure detectors can make mistakes so we should expect live processes to be suspected as crashed and the other way around. To be useful to us, failure detectors should also provide correct information about the status of the system. To judge the degree of trustworthiness of the information that failure detectors provide we classify them by their *completeness* and *accuracy*.

2.1.4 Failure Detector's Properties

Completeness

Failure detectors are classified into two groups based on their completeness [3]:

- **Strong completeness**: eventually *every* crashed process is permanently suspected by *every* correct process
- **Weak completeness**: eventually *every* crashed process is permanently suspected by *some* correct process

Completeness	Accuracy			
	Strong	Weak	Eventual Strong	Eventual Weak
Strong	*Perfect* $\mathcal{P}$	*Strong* $\mathcal{S}$	*Eventually Perfect* $\diamond\mathcal{P}$	Eventually Strong $\diamond\mathcal{S}$
Weak	$\mathcal{Q}$	Weak $\mathcal{W}$	$\diamond\mathcal{Q}$	Eventually Weak $\diamond\mathcal{W}$

Table 2.1: Eight classes of failure detectors defined by the accuracy and completeness properties [3]

Completeness by itself is not useful because we can trivially satisfy strong completeness, by suspecting every process, if there are no limitations on the number of mistakes we can make.

Accuracy

By introducing the accuracy property, we have a way of classifying the number of mistakes expected from a failure detector. A failure detector can be classified as having:

- **Strong accuracy**: no process is suspected before it crashes
- **Weak completeness**: at least one correct process is never suspected

Strong accuracy and even weak accuracy properties are hard to satisfy since there must be at least one alive process that is never suspected. Because of this Chandra and Toueg [3] introduce the concept of *eventual* satisfiability. The resulting *eventual strong accuracy* and *eventual weak accuracy* properties are much like their non-eventual, or often called perpetual, counterparts but only require the accuracy constraint to be satisfied after a certain point in time.

Combination of the completeness and accuracy properties give us 8 types of failure detectors.

For the rest of the paper, the classes of failure detectors would be referred by the notation given in Table 2.1.

2.2 RDMA

2.2.1 Motivation

With Moore's law close to the end of its natural path [16, 6], CPU improvements focus less on clock speed and more on multi-core processors. This has led to a stagnation or even decrease in the clock speeds of modern processors compared to their predecessors. On the other hand, network technology is improving rapidly. With a 10Gbps and above network interconnects available, suddenly, the CPU interrupt that the reception of a network packet triggers do not seem so insignificant.

Advancements in networking have demanded few features from a future-proof network protocol:

- Hardware offloading
- Kernel Bypass
- Zero-copy transfers
- One-sided, polling-based processing that would replace the CPU interrupts

2.2.2 Benefits

Remote Direct Memory Access (RDMA) is the answer to those demands. RDMA allows one computer to directly access the memory of another remote computer without involving the operating systems of either of the machines.

Remote Direct Memory Access (RDMA) is the answer to those demands. RDMA allows one computer to directly access the memory of another remote computer without involving the operating systems of either of the machines.

RDMA also allows for zero-copy transfers. This means that no copies of data are needed to or from the memory buffer. This enables the data to be read directly from another node's memory. Without RDMA a network transfer would require two traversals on the memory bus – to memory and back. Furthermore, those transfers would compete for memory bandwidth with the rest of the processes running on the machine. This results in lower CPU load, higher throughput and lower latency with RDMA proven to achieve sub 10μs transfers in a controlled benchmark [10].

2.2.3 Adoption

The native RDMA protocol can run only over a lossless network and thus requires the InfiniBand network fabric [12]. That means that in order to use the protocol changes to the data centre's network fabrics and NICs would be necessary, such that they are RDMA-enabled.

However, there is also an alternative that is using protocols that describe how RDMA should act in order to emulate lossless network when running over Ethernet [9, 17, 8]. This eliminates the economic and technical aspect of repurchasing the infrastructure of a data centre and allowing for speeds comparable but that can be slower when compared to running over InfiniBand. Regardless, the results of implementing RDMA over Ethernet (RoCEv2) at Bing show us that it is a lot faster than using TCP – 99th percentile latencies being 90μs and 700μs for RDMA and TCP respectively [8].

Chapter 3

Failure detector implementation

3.1 Testing framework

3.2 Heartbeats

3.3 Adaptive timeout

3.4 PAXOS

Chapter 4

Failure detection over RDMA

Chapter 5

Analysis

Chapter 6

Conclusion

6.1 Overview

6.2 Future work

Bibliography

[1] Top500 supercomputer sites. https://www.top500.org/. [Online; Accessed on 01/25/2019].

[2] Inc. Amazon Web Services. Lambda architecture for batch and stream processing. https://d1.awsstatic.com/whitepapers/lambda-architecure-on-for-batch-aws.pdf, October 2018. [Online; Accessed on 01/25/2019].

[3] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM)*, 43(2):225–267, 1996.

[4] Mellanox Technologies Cisco. Benefits of remote direct memory access over routed fabrics. [Online; Accessed on 01/25/2019].

[5] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, 31(3):8, 2013.

[6] Manesh Dubash. Moore's law is dead, says gordon moore. https://www.techworld.com/news/tech-innovation/moores-law-is-dead-says-gordon-moore-3576581/. [Online; Accessed on 01/25/2019].

[7] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE, 1982.

[8] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. Rdma over commodity ethernet at scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 202–215. ACM, 2016.

[9] Intel. Understanding iWARP: Delivering Low Latency to Ethernet. [Online; Accessed on 01/25/2019].

[10] Anuj Kalia, Michael Kaminsky, and David G. Andersen. Design guidelines for high performance RDMA systems. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 437–450, Denver, CO, 2016. USENIX Association.

[11] Michael Oberg, Henry M Tufo, Theron Voran, and Matthew Woitaszek. Evaluation of rdma over ethernet technology for building cost effective linux clusters. In *7th LCI International Conference on Linux Clusters: The HPC Revolution*, 2006.

[12] Gregory F Pfister. An introduction to the infiniband architecture. *High Performance Mass Storage and Parallel I/O*, 42:617–632, 2001.

[13] Marius Poke and Torsten Hoefler. Dare: High-performance state machine replication on rdma networks. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, pages 107–118. ACM, 2015.

[14] Bianca Schroeder and Garth A Gibson. Understanding failures in petascale computers. In *Journal of Physics: Conference Series*, volume 78, page 012022. IOP Publishing, 2007.

[15] SiliconANGLE. Forecast of big data market size, based on revenue, from 2011 to 2027 (in billion u.s. dollars). Statista - The Statistics Portal, Statista `https://www.statista.com/statistics/254266/global-big-data-market-forecast/`. [Online; Accessed on 01/25/2019].

[16] Tom Simonite. Moore's law is dead. now what? - mit technology review. `https://www.technologyreview.com/s/601441/moores-law-is-dead-now-what/`. [Online; Accessed on 01/25/2019].

[17] Mellanox Technologies. RoCE in the Data Center, Oct 2016. [Online; Accessed on 01/25/2019].

Appendices

Appendix A

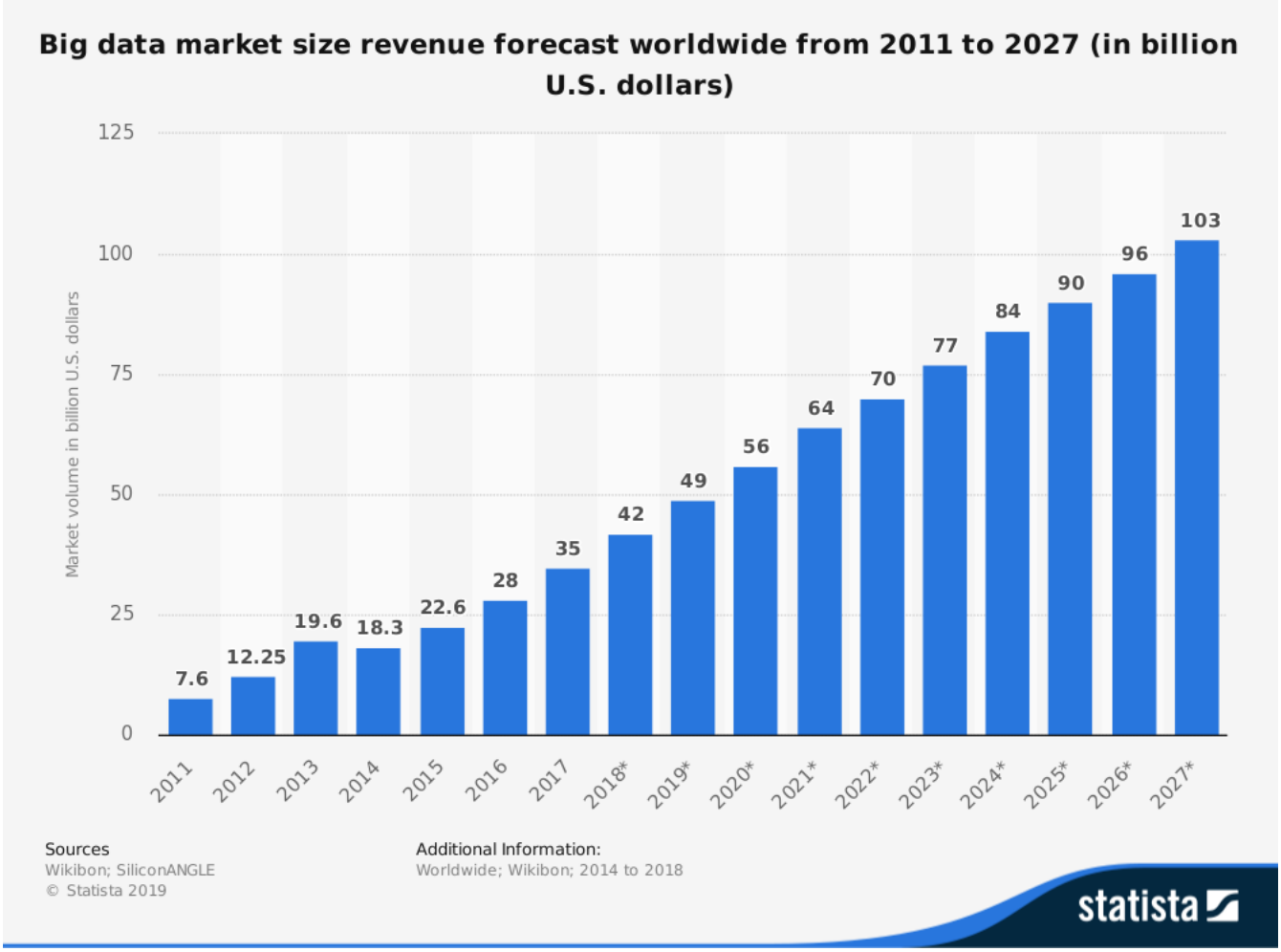

Figure A.1: Forecast of Big Data market size, based on revenue, from 2011 to 2027 (in billion U.S. dollars) [15]