RoBuSt: A Crash-Failure-Resistant Distributed Storage System*

Martina Eikel, Christian Scheideler, and Alexander Setzer

University of Paderborn, Germany
{martinah,scheideler,asetzer}@mail.upb.de

Abstract. In this work we present the first distributed storage system that is provably robust against crash failures issued by an adaptive adversary, i.e., for each batch of requests the adversary can decide based on the entire system state which servers will be unavailable for that batch of requests. Despite up to $\gamma n^{1/\log \log n}$ crashed servers, with $\gamma > 0$ constant and $n$ denoting the number of servers, our system can correctly process any batch of lookup and write requests (with at most a polylogarithmic number of requests issued at each non-crashed server) in at most a polylogarithmic number of communication rounds, with at most polylogarithmic time and work at each server and only a logarithmic storage overhead.

Our system is based on previous work by Eikel and Scheideler (SPAA 2013), who presented IRIS, a distributed information system that is provably robust against the same kind of crash failures. However, IRIS is only able to serve lookup requests. Handling both lookup and write requests has turned out to require major changes in the design of IRIS.

Keywords: Theory of Distributed Systems, DHT, Crash Failures, Denial-of-Service Attacks.

1 Introduction

One of the main challenges of a distributed system is that it is able to work correctly even if parts of the system fail to work. If a server experiences a crash failure it becomes unavailable to the other servers, i.e., it does not issue or respond to requests any more. Crash failures can be temporary or permanent, and if it is temporary, a server may either be back to its state when it crashed, or it may have lost all of its state. We will focus on crash failures where, whenever a server becomes available again, it is back to its state when it crashed. This is a reasonable assumption since for commercial servers it is extremely rare that their state cannot be recovered. However, a temporary unavailability is not that uncommon and can have many causes such as maintenance work, hardware or software glitches, or denial-of-service attacks. Especially denial-of-service attacks

* This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center “On-The-Fly Computing” (SFB 901) and by the EU within FET project MULTIPLEX under contract no. 317532
can be a serious threat because they are normally unpredictable, hard to prevent and they can cause the unavailability of a server for an extended period of time.

Predominant approaches in information and storage systems to deal with the threat of crash failures are to use redundancy: information that is replicated among multiple machines is likely to remain accessible even if some servers are unavailable. Unfortunately, in systems that consist of thousands of servers a complete replication of the data over all servers is not feasible. Hence, one needs to find an appropriate tradeoff between the amount of redundancy and the number of crashed servers the system can handle. One can easily show that if $\Theta(\log n)$ copies of a data item are placed randomly among $n$ servers, and these random positions are not known to the adversary, then any strategy of the adversary that blocks half of the servers will not block all of the copies, with high probability\footnote{“With high probability”, or short, “w.h.p.”, means a probability of at least $1 - 1/n^c$ where the constant $c$ can be made arbitrarily large.}.

The situation is completely different, however, when considering an adaptive adversary, i.e., someone who has complete knowledge about the system.

In a previous work, Eikel and Scheideler \cite{EikelScheideler2014} presented a distributed information system, called IRIS, that just needs a constant storage redundancy in order to be robust against an adaptive adversary that can crash up to $\Theta(n^{1/\log \log n})$ servers. Unfortunately, the system lacks the important ability to handle write requests, i.e., to add, remove and update data items. This work solves this problem.

1.1 Model and Preliminaries

We assume that the storage system consists of a static set $S = \{s_1, \ldots, s_n\}$ of $n$ reliable servers of identical type. The servers are responsible for storing the data as well as handling the user requests. We assume that all data items are of the same size, and that any data item $d$ is uniquely identified by a key $key(d)$. The universe of all possible keys is denoted by $U$, and $m := |U|$ is assumed to be polynomial in $n$. Furthermore, we assume that the size of the data items is at least $\Omega(\log n \log m)$. There are two types of user requests: $\text{lookup}(k)$ for $k \in U$, and $\text{write}(k, d)$ for $k \in U$ and a data item $d$. The user can issue a request by sending it to one of the servers in $S$. Given a $\text{lookup}(k)$ request, the system is supposed to either return the data item $d$ with $key(d) = k$, or to return NULL if no such data item exists. Given a $\text{write}(k, d)$ request, the system is supposed to store data item $d$ with key $k$ such that subsequent $\text{lookup}(k)$ requests can be answered correctly. Note that with a $\text{write}(\cdot)$ request the user can also update or remove data.

Every server knows about all other servers and can therefore directly communicate with any one of them. This does not endanger scalability since millions of IP addresses can easily be stored in main memory in any reasonable computer today and we assume the set of servers to be static. We use the standard synchronous message passing model for the communication between the servers. That is, time proceeds in synchronized communication rounds, or simply rounds,