Integrated Bounds for Disintegrated Storage

Alon Berger

Integrated Bounds for Disintegrated Storage

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering

Alon Berger

Submitted to the Senate of
the Technion — Israel Institute of Technology
Shebat 5779 Haifa January 2019

The research thesis was done under the supervision of Prof. Idit Keidar in the Andrew and Erna Viterbi Faculty of Electrical Engineering.

Some results in this thesis have been published as articles by the author and research collaborators in the following conference during the course of the author's research period:

1. Alon Berger, Idit Keidar, and Alexander Spiegelman. Integrated Bounds for Disintegrated Storage. In *32nd International Symposium on Distributed Computing (DISC 2018)*, volume 121 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:18, 2018.

In addition, the author and other research collaborators published additional work, which is not included in this thesis, in the following conference and journal:

2. Sharad Chole, Andy Fingerhut, Sha Ma, Anirudh Sivaraman, Shay Vargaftik, Alon Berger, Gal Mendelson, Mohammad Alizadeh, Shang-Tse Chuang, Isaac Keslassy, Ariel Orda, and Tom Edsall. dRMT: Disaggregated Programmable Switching. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017, Los Angeles, CA, USA, August 21-25, 2017*, pages 1–14, 2017.

3. Adi Hanuka, Maor Itzhak, Alon Berger, Mony Orbach, Eli Shoshan, Levi Schächter, and Daniel Briscoe. A novel eyelid motion monitor. *Graefe's Archive for Clinical and Experimental Ophthalmology*, 255(9):1811–1817, 2017.

The generous financial support of the Meyer Foundation is gratefully acknowledged.

Acknowledgments

I would like to thank several people who made this research possible, fruitful, and enjoyable.

- I thank my advisor, Idit Keidar, for your guidance, support, patience, and optimism. Thank you for teaching me how to perform research, and for seeing the big picture. It is always a pleasure dropping by your office for advice and brainstorming. I have learned a great deal from you, and enjoyed every moment of it.
- I wish to thank Alexander (Sasha) Spiegelman, with whom I worked closely on these results. Thank you for insisting on improving the results and never settling half-way through. Your high standards and persistence are admirable. It was truly great collaborating with you.
- To the members of Idit's research group; Sasha, Hagar Meir, Naama Kraus, Noam Shalev, Kfir Lev-Ari, and Dani Shaket: thank you for making every meeting interesting, helping in problem solving, caring for others' work, and for the laughs at the annual asado events.
- I would like to thank Isaac Keslassy, Shay Vargaftik, and Gal Mendelson, for giving me my first valuable lessons in research.
- I thank Yuval Cassuto, Gregory Chockler, Rati Gelashvili, and Yuanhao Wei for many insightful discussions on space bounds for coded storage and emulations of large registers from smaller ones.
- I would like to thank the administrative staff, Orly Babad-Tamir, Danit Cohen, Mira Aran, Hovav Gazit, and Roy Mitrany, for making this trip a smooth and efficient one.
- I wish to thank my friends who made the past years enjoyable. Thank you for all of the meals, for so much laughter, and for support. In particular I would like to mention Itay Tsabary and Hagar Meir, my office mates and some of my real life best friends. I am glad and appreciative to have formed such great friendships.
- Lastly, I would like to thank my family, for pushing me to the limit, supporting me, and encouraging me to reach the sky. Especially my wife Galia, thank you for your support and sacrifice, for taking interest in my research, and for bearing with me despite the long nights working. I am very lucky to have you by my side.

Contents

List of Tables

Abstract

In recent years we have seen an exponential increase in storage capacity demands, creating a need for big data storage solutions. Distributed storage is a common approach to solve this necessity: data is typically stored on various nodes, which are accessed asynchronously by clients. Such storage systems that support writing values and reading them are commonly referred to as *registers*. A common requirement of registers is *wait-freedom*, meaning that, given enough actions, any client performing an operation on the register eventually completes it.

Several algorithms for wait-free registers exist: Byzantine fault-tolerant registers cope with the possibility that some fraction of the nodes might be malicious by replicating data across distinct nodes. Coded storage solutions wish to alleviate the space cost of replication by having the nodes store code words, rather than entire values. Other kinds of registers are designed to store values from a large domain, while each of the nodes is a register whose capacity is limited to storing values from a small domain.

In this research thesis, we point out a somewhat surprising similarity between non-authenticated Byzantine storage, coded storage, and certain emulations of shared registers from smaller ones. A common characteristic in all of these is the inability of reads to safely return a value obtained in a single atomic access to shared storage. We collectively refer to such systems as *disintegrated storage*, and show integrated space lower bounds for asynchronous regular wait-free emulations in all of them. In a nutshell, if readers are invisible, i.e., their presence is unknown to the writer, then the storage cost of such systems is inherently exponential in the size of written values; otherwise, it is at least linear in the number of readers. Our bounds are asymptotically tight to known algorithms, and thus justify their high costs.

Abbreviations

SW	—	Single Writer
SR	—	Single Reader
MR	—	Multiple Readers

Notation

τ	—	The number of data blocks needed to recover a value
D	—	The value size
L	—	The maximal number of data blocks stored in a reader's local data
R	—	The number of readers in systems containing visible readers
O	—	The set of all objects
n	—	The number of shared objects, $n = \|O\|$
Π	—	The set of all processes
$\mathbb{B}$	—	The domain of data blocks
$\mathbb{V}$	—	The domain of values
$\mathbb{N}$	—	The set of natural numbers
$p.data$	—	The array of blocks locally stored at a process $p \in \Pi$
$o.data$	—	A block stored in the shared object $o \in O$
$e.meta$	—	meta-data stored at an entity e
$o.\mathtt{a}_p$	—	An action $\mathtt{a}$ performed by process p at object o
$\mathcal{A}$	—	An algorithm
r	—	A run
t	—	A time in a run
t_r	—	The final time in a finite run r
$r' \setminus r$	—	The suffix of an extension r' of r that starts at t_r
$e.data(r, t)$	—	The set of blocks stored in $e.data$ at time t in a run r
$Labels(b)$	—	The set of labels associated with a data block b
$\mathbb{W}_v^{\mathcal{A}}$	—	The set of `write` operations of a value v invoked in runs of algorithm $\mathcal{A}$
$\mathbb{W}_V^{\mathcal{A}}$	—	The set of `write` operations of values from a set V invoked in runs of algorithm $\mathcal{A}$
$\mathbb{W}_{\mathbb{V}}^{\mathcal{A}}$	—	The set of `write` operations invoked in runs of algorithm $\mathcal{A}$
$\langle w, k\rangle$	—	The label created by the k^{th} `update` in a `write` operation w
$S\text{–}labels\ (v, r, t)$	—	The set of labels of value v in shared storage at time t of run r
$L\text{–}labels_p\ (v, r, t)$	—	The set of labels of value v in a reader p's local storage at time t of run r
$All\text{–}labels_p\ (v, r, t)$	—	The set of labels of value v either in shared storage or in a reader p's local storage at time t of run r
$values_p\ (r, t)$	—	The set of values of which a label exists in a reader p's local storage at time t of a run r
$S\text{–}labels\ (w, r, t)$	—	The set of labels of `write` w in shared storage at time t of run r
$L\text{–}labels_p\ (w, r, t)$	—	The set of labels of `write` w in a reader p's local storage at time t of run r
$All\text{–}labels_p\ (w, r, t)$	—	The set of labels of `write` w either in shared storage or in a reader p's local storage at time t of run r
$writes_p\ (r, t)$	—	The set of `write`s of which a label exists in a reader p's local storage at time t of a run r

Chapter 1

Introduction

1.1 Space bounds for encoded, multi-register, and Byzantine storage

In many data sharing solutions, information needs to be read from multiple sources in order for a single value to be reconstructed. One such example is coded storage, where multiple storage blocks need to be obtained in order to recover a single value that can be returned to the application [5, 9, 10, 16, 18, 19, 24, 25]. Another example arises in shared memory systems, where the granularity of atomic memory operations (such as load and store) is limited to a single word (e.g., 64 bits) and one wishes to atomically read and write larger values [24]. A third example is replicating data to overcome Byzantine faults (without authentication) or data corruption, where a reader expects to obtain the same block from multiple servers in order to validate it [1, 2, 20].

We refer to such systems collectively as *disintegrated storage* systems. We show that the need to read data in multiple storage accesses inherently entails high storage costs: exponential in the data size if reads do not modify the storage, and otherwise linear in the number of concurrent reads. This stands in contrast to systems that use non-Byzantine replication, such as ABD [6], where, although meta-data (e.g., timestamps) is read from several sources, the recovered value need only be read from a single source.

1.2 Our results

We consider a standard shared storage model (see Chapter 2). We refer to shared storage locations (representing memory words, disks, servers, etc.) as *objects*. To strengthen our lower bounds, we assume that objects are responsive, i.e., do not fail; the results hold *a fortiori* if objects can also be unresponsive [20]. Objects support general read-modify-write operations by asynchronous *processes*. We study *wait-free* emulations of a shared *regular register* [21].

Chapter 3 formally defines *disintegrated storage*. We use a notion of *blocks*, which are parts of a value kept in storage – code blocks, segments of a longer-than-word value, or full copies of a replicated value. A key assumption we make is that each block in the shared storage pertains to a single write operation; a similar assumption was made in previous studies [11, 25]. The disintegration property then stipulates that a reader must obtain some number $\tau > 1$ of blocks pertaining to a value v before returning v. For example, τ blocks are needed in τ-out-of-n coded storage, whereas $\tau = f + 1$ in f-tolerant Byzantine replication, in executions where no object actually

	Invisible Reads	**Visible Reads**
General Case	$\tau + (\tau - 1)\left\lceil \frac{2^D-1}{L} \right\rceil$	$\tau + (\tau - 1) \cdot \min\left(\left\lceil \frac{2^D-1}{L} \right\rceil , R \right)$
Common Write (e.g., coded storage)	$\tau \cdot 2^D$	$\tau + (\tau - 1) \cdot \min\left(2^D - 1 , R\right)$

Table 1.1: Lower bounds on shared storage space consumption, in units of blocks; D is the value size, $\tau > 1$ is the number of data blocks required in order to recover a value, $L \geq 1$ is the maximal number of blocks stored in a reader's local data, and R the number of readers.

fails. To strengthen our results, we allow the storage to hold unbounded meta-data (e.g., timestamps), and count only the storage cost for blocks. Note that the need to obtain τ blocks implies that meta-data cannot be used instead of actual data.

In Chapter 4 we give general lower bounds that apply to all types of disintegrated storage – replicated, coded, and multi-register. We first consider *invisible reads*, which do not modify the shared storage. This is a common paradigm in storage systems and is often essential where readers outnumber writers and have different permissions. In this case, even with one reader and one writer, the storage size can be exponential; specifically, if value sizes are D (taken from a domain of size 2^D), then we show a lower bound of $\tau + (\tau - 1)\left\lceil \frac{2^D-1}{L} \right\rceil$ shared blocks, where L is the number of blocks in a reader's local storage. That is, if the local storage of the reader is not exponential in the value size, then the shared storage is.

Chapter 5 studies a more restrictive flavor of disintegrated storage, called *τ-common write*, where a reader needs to obtain τ blocks produced by the *same* `write`(v) operation in order to return v. In other words, if the reader obtains blocks that originate from two different `write`s of the same value, then it cannot recognize that they pertain to the same value, as is the case when blocks hold parts of a value or code blocks rather than replicas. In this case, the shared storage cost is high independently of the local memory size. Specifically, we show a bound of $\tau \cdot 2^D$ blocks with invisible readers. In systems that use symmetric coding (i.e., where all blocks are of the same size, namely at least D/τ bits), this implies a lower bound of $D \cdot 2^D$ bits. For a modest value size of 20 bytes, the bound amounts to $2.66 \cdot 10^{37}$ TB, and for 1KB values it is a whopping $1.02 \cdot 10^{2457}$ TB.

We further consider *visible reads*, which can modify the objects' meta-data. In this case, readers may indicate to the writers that a read is ongoing, and signal to them which blocks to retain. Using such signals, the exponential bound no longer holds – there are register emulations that store a constant number of values per reader [2, 5, 13, 17, 24]. We show that such linear growth with the number of readers is inherent. Our results are summarized in Table 1.1.

These bounds are tight as far as regularity and wait-freedom go: relaxing either requirement allows circumventing our results [1, 20]. As for storage cost, our lower bounds are asymptotically tight to known algorithms, whether reads are visible [2, 5, 17, 24] or not [7, 16, 19, 23].

We note that the study of the inherent storage blowup in asynchronous coded systems has only recently begun [11, 25] and is still in its infancy. In this work, we point out a somewhat surprising similarity between coded storage and other types of shared memory/storage, and show unified lower bounds for all of them. Chapter 6 concludes the thesis and suggests directions for future work.

1.3 Related work and applicability of our bounds

Several works have studied the space complexity of register emulations. Two recent works [11, 25] show a dependence between storage cost and the number of *writers* in crash-tolerant storage, identifying a trade-off between the cost of replication ($f + 1$ copies for tolerating f faults) and that of τ-out-of-n coding (linear in the number of writers). Though they do not explicitly consider disintegrated storage, it is fairly straightforward to adapt the proof from [25] to derive a lower bound of τW blocks with W writers. Here we consider the case of single-writer algorithms, where this bound is trivial. Other papers [3, 15] show limitations of multi-writer emulations when objects do not support atomic read-modify-write, whereas we consider single-writer emulations that do use read-modify-write.

Chockler et al. [14] define the notion of *amnesia* for register emulations with an infinite value domain, which intuitively captures the fact that an algorithm "forgets" all but a finite number of values written to it. They show that a wait-free regular emulation tolerating non-authenticated Byzantine faults with invisible readers cannot be amnesic, but do not show concrete space lower bounds. In this work we consider a family of disintegrated storage algorithms, with visible and invisible readers, and show concrete bounds for the different cases; if the size of the value domain is unbounded, then our invisible reader bounds imply unbounded shared storage.

Disintegrated storage may also correspond to emulations of large registers from smaller ones, where τ is the size of the big register divided by the size of the smaller one. Some algorithms in this vein, e.g., [24], indeed have the disintegration property, as the writer writes τ blocks to a buffer and a reader obtains τ blocks of the same write. These algorithms are naturally subject to our bounds. Other algorithms, e.g., [12, 13, 21], do not satisfy our assumption that each block in the shared storage pertains to a single `write` operation, and a reader may return a value based on blocks written by *different* `write` operations. Thus, our bounds do not apply to them. It is worth noting that these algorithms nevertheless either have readers signal to the writers and use space linear in the number of readers, or have invisible readers but use space exponential in the value size. Following an earlier publication of our work in [8], Wei [27] showed that these costs – either linear in the number of visible readers or exponential in the value size with invisible ones – are also inherent in emulations of large registers from smaller ones that *do* share blocks among writes, albeit do not use meta-data at all. Several questions remain open in this context: first, Wei's bound is not applicable to all types of storage we consider (in particular, Byzantine), and does not apply to algorithms that use timestamps. Second, we are not familiar with any regular register emulations where readers write-back data, and it is unclear whether our bound may be circumvented in this way.

Non-authenticated Byzantine storage algorithms that tolerate f faults need to read a value $f+1$ times in order to return it, and are thus τ-disintegrated for $\tau = f+1$. Note that while our model assumes objects are responsive, it *a fortiori* applies to scenarios where objects may be unresponsive. Some algorithms circumvent our bound either by providing only safe semantics [20, 22], or by forgoing wait-freedom [1]. Others use channels with unbounded capacity to push data to clients [7, 23] or potentially unbounded storage with best-effort garbage collection [19].

As for coded storage, whenever τ blocks are required to reconstruct a value, the algorithm is τ-disintegrated. And indeed, previous solutions in our model require unbounded storage or channels [9,10,16,18,19], or retain blocks for concurrent visible readers, consuming space linear in the number of readers [5]. Our bounds justify these costs. Our assumption that each block in the shared storage pertains to a single value is satisfied by almost all coded storage algorithms we are aware of. The only exception is [26], which indeed circumvents our lower bound but does not conform to regular register semantics. Other coded storage solutions, e.g., [4], are not subject to our bound because they may recover a value from a single block.

Chapter 2

Preliminaries

Shared storage model We consider an asynchronous shared memory system consisting of two types of entities: A finite set $O = \{o_1, \ldots, o_n\}$ of objects comprising *shared storage*, and a set Π of processes. Every entity in the system stores *data*: an object's data is a single block from some domain $\mathbb{B}$, whereas a process' data is an array of up to L blocks from $\mathbb{B}$. We assume a bound L on the number of blocks in the data array of each process. In addition, each entity stores potentially unbounded meta-data, *meta*. We denote an entity e's data as $e.data$ and likewise for $e.meta$. A system's *storage cost* is the number of objects in the shared storage, n.

Objects support atomic `get` and `update` *actions* by processes. We denote by $\mathtt{a}_p$ an action $\mathtt{a}$ performed by p and by $o.\mathtt{a}_p$ an $\mathtt{a}_p$ action applied to o. An $o.\mathtt{update}_p$ is an arbitrary read-modify-write that possibly writes a block from $\mathbb{B}$ to $o.data$ and modifies $o.meta$, $p.meta$, and $p.data$. An $o.\mathtt{get}_p$ may replace a block in $p.data$ with $o.data$ and may modify $p.meta$.

Algorithms, configurations, and runs An *algorithm* defines the behaviors of processes as deterministic state machines, where state transitions are associated with actions. A *configuration* is a mapping to states (data and meta) from all system components, i.e., processes and objects. In an *initial configuration* all components are in their initial states.

We study algorithms (executed by processes in Π) that emulate a high-level functionality, exposing high-level operations, and performing low-level `get`s/`update`s on objects. We say that high-level operations are *invoked* and *return* or *respond.* Note that, for simplicity, we model `get`s and `update`s as instantaneous actions, because the objects are assumed to be atomic, and we do not explicitly deal with object failures in this thesis.

A *run* of algorithm $\mathcal{A}$ is a (finite or infinite) alternating sequence of configurations and actions, beginning with some initial configuration, such that configuration transitions occur according to $\mathcal{A}$. Occurrences of actions in a run are called *events.* The possible events are high-level operation invocations and responses and `get`/`update` occurrences. We use the notion of time t during a run r to refer to the configuration reached after the t^{th} event in r. For a finite run r consisting of t events we define $t_r \triangleq t$. Two operations are *concurrent* in a run r if both are invoked in r before either returns. If a process p's state transition from state $\mathcal{S}$ is associated with a low-level action $\mathtt{a}_p \in \{\mathtt{get}_p, \mathtt{update}_p\}$, we say that $\mathtt{a}_p$ is *enabled* in $\mathcal{S}$. A run r' is an *extension* of a (finite) run r if r is a prefix of r'; we denote by $r' \setminus r$ the suffix of r' that starts at

t_r. If a high-level operation *op* has been invoked by process p but has not returned by time t in a run r, we say that *op*'s invocation is *pending* at t in r. We assume that each process' first action in a run is an invocation, and a process has at most one pending invocation at any time.

For $e \in \Pi \cup O$, we denote by $e.data(r,t)$ the set of distinct blocks stored in $e.data$ at time t in a run r. Since for an object o, $|o.data\,(r,t)| = 1$, we sometimes refer to $o.data\,(r,t)$ as the block itself, by slight abuse of notation. We say that p *obtains* a block b at time t in a run r, if $b \notin p.data\,(r,t)$ and $b \in p.data\,(r,t+1)$.

Register emulations We study algorithms that emulate a shared *register* [21], which stores a value v from some domain $\mathbb{V}$. We assume that $|\mathbb{V}| = 2^D > 1$, i.e., values can be represented using $D > 0$ bits. For simplicity, we assume that each run begins with a dummy initialization operation that writes the register's initial value and does not overlap any operation. The register exposes high-level $\texttt{read}_p$ and $\texttt{write}_p(v)$ operations of values $v \in \mathbb{V}$ to processes $p \in \Pi$. We consider single-writer (SW) registers where the application at only one process (the *writer*) invokes `write`s, and hence omit the subscript p from $\texttt{write}(v)$. The remaining $R \triangleq |\Pi| - 1$ processes are limited to performing `read`s, and are referred to as *readers*. For brevity, we refer to the subsequence of a run where a specific invocation of a $\texttt{write}(v)/\,\texttt{read}_p$ is pending simply as a $\texttt{write}(v)/\,\texttt{read}_p$ operation.

We assume that whenever a $\texttt{read}_p$ operation is invoked at time t in a run r, $p.data\,(r,t)$ is empty. We consider two scenarios: (1) *invisible reads*, where `read`s do not perform `update`s, and (2) *visible reads*, where `read`s may perform `update`s that update meta-data (only) in the shared storage. Note that readers do *not* write actual data, which is usually the case in regular register emulations, defined below. In a single-reader (SR) register $R = 1$, and if $R > 1$ the register is multi-reader (MR). If the states of the writer and the objects at the end of a finite run r are equal to their respective states at the end of a finite run r', we say that t_r and $t_{r'}$ are *indistinguishable* to the writer and objects, and denote: $t_r \approx_w t_{r'}$.

Our safety requirement is *regularity* [21]: a `read` rd must return either the value of the last `write` w that returns before rd is invoked, or the value of some write that is concurrent with rd. For liveness, we require *wait-freedom*, namely that every operation invoked by a process p returns within a finite number of p's actions. In other words, if p is given infinitely many opportunities to perform actions, it completes its operation regardless of the actions of other processes.

Chapter 3

Disintegrated storage

As noted above, existing wait-free algorithms of coded and/or Byzantine-fault-tolerant storage with invisible readers may store all values ever written [7, 9, 11, 16, 18, 19, 23]. This is because if old values are erased, it is possible for a slow reader to never find sufficiently many blocks of the same value so as to be able to return it. If readers are visible, then a value per reader is retained. We want to prove that these costs are inherent. The challenge in proving such space lower bounds is that the aforementioned algorithms use unbounded timestamps. How can we show a space lower bound if we want to allow algorithms to use unbounded timestamps? We address this by allowing meta-data to store timestamps, etc., and by not counting the storage cost for meta-data. For example, the above algorithms store timestamps in meta-data alongside data blocks and use them to figure out which data is safe to return, but still need τ actual blocks/copies of a value in order to return it. Note that for the sake of the lower bound, we do not restrict how meta-data is used; all we require is that the algorithm read τ data blocks of the same value (or `write`), and we do not specify how the algorithm knows that they pertain to the same value (or `write`). To formalize the property that the algorithm returns τ blocks pertaining to the same value or `write`, we need to track, for each block in the shared storage, which `write` produced it. To this end, we define *labels*. Labels are only an analysis tool, and do not exist anywhere. In particular, they are not timestamps, not meta-data, and not explicitly known to the algorithm. As an external observer, we may add them as abstract state to the blocks, and track how they change.

Labels We associate each block b in the shared or local storage with a set of labels, $\mathit{Labels}(b)$, as we now explain. For an algorithm $\mathcal{A}$ and $v \in \mathbb{V}$, denote by $\mathbb{W}_v^{\mathcal{A}}$ the set of `write`(v) operations invoked in runs of $\mathcal{A}$. For $V \subseteq \mathbb{V}$, we denote $\mathbb{W}_V^{\mathcal{A}} \triangleq \bigcup_{v \in V} \mathbb{W}_v^{\mathcal{A}}$, and let $\mathbb{W}^{\mathcal{A}} \triangleq \mathbb{W}_{\mathbb{V}}$. For clarity, we omit $\mathcal{A}$ when obvious from the context, and refer simply to $\mathbb{W}_v$, $\mathbb{W}_V$, and $\mathbb{W}$. We assume that the k^{th} `update` event occurring in a `write` operation $w \in \mathbb{W}$ tags the block b it stores (if any) with a unique label $\langle w, k\rangle$, so $\mathit{Labels}(b)$ becomes $\{\langle w, k\rangle\}$.

Whereas our assumption that each block in the shared storage pertains to a single write rules out associating multiple labels with such a block, we do allow the reader's meta-data to recall multiple accesses encountering the same block. For example, when blocks are copies of a replicated value, the reader can store one instance of the value in local memory and keep a list of the objects where the value was encountered. To this end, a block in a reader's *data* may be tagged with multiple labels: when a reader p

obtains a block b from an object o at time t in a run r, the block b in $p.data\,(r, t+1)$ is tagged with $Labels(o.data\,(r,t))$; if at time $t' > t$ $p.data$ still contains b and p performs an action on an object o' s.t. $o'.data\,(r,t') = b$ and the latter is tagged with label ℓ, then p adds ℓ to $Labels(b)$ (regardless of whether b is added to $p.data$ once more). When all copies of a block are removed from $p.data$, all its labels are "forgotten". We emphasize that labels are not stored anywhere, and are only used for analysis.

We track the labels of a value $v \in \mathbb{V}$ at time t in a run r using the sets $S\text{–}labels\,(v,r,t)$, of labels in the shared storage, $L\text{–}labels_p\,(v,r,t)$, of labels in process p's local storage, and $All\text{–}labels_p\,(v,r,t)$, a combination of both. Formally,

- $S\text{–}labels\,(v,r,t) \triangleq \left(\bigcup_{o\in O} Labels(o.data\,(r,t))\right) \cap (\mathbb{W}_v \times \mathbb{N})$.
- $L\text{–}labels_p\,(v,r,t) \triangleq \left(\bigcup_{b\in p.data(r,t)} Labels(b)\right) \cap (\mathbb{W}_v \times \mathbb{N})$.
- $All\text{–}labels_p\,(v,r,t) \triangleq L\text{–}labels_p\,(v,r,t) \cup S\text{–}labels\,(v,r,t)$.

For a time t in a run r and $p \in \Pi$, we define $values_p\,(r,t) \triangleq \{v \in \mathbb{V} \mid L\text{–}labels_p\,(v,r,t) \neq \emptyset\}$.

Similarly, we track labels associated with a particular `write` $w \in \mathbb{W}$ accessible by process $p \in \Pi$ at time t in a run r:

- $S\text{–}labels\,(w,r,t) \triangleq \left(\bigcup_{o\in O} Labels(o.data\,(r,t))\right) \cap (\{w\} \times \mathbb{N})$.
- $L\text{–}labels_p\,(w,r,t) \triangleq \left(\bigcup_{b\in p.data(r,t)} Labels(b)\right) \cap (\{w\} \times \mathbb{N})$.
- $All\text{–}labels_p\,(w,r,t) \triangleq L\text{–}labels_p\,(w,r,t) \cup S\text{–}labels\,(w,r,t)$.

We define $writes_p\,(r,t) \triangleq \{w \in \mathbb{W} \mid L\text{–}labels_p\,(w,r,t) \neq \emptyset\}$. Note that for all $v \in \mathbb{V}$ and $w \in \mathbb{W}_v$, (1) $S\text{–}labels\,(w,r,t) \subseteq S\text{–}labels\,(v,r,t)$, (2) $L\text{–}labels_p\,(w,r,t) \subseteq L\text{–}labels_p\,(v,r,t)$, and (3) $All\text{–}labels_p\,(w,r,t) \subseteq All\text{–}labels_p\,(v,r,t)$.

Since readers do not write-back:

Observation 1. *If the t^{th} event in a run r is of a reader $p \in \Pi$, then for all $v \in \mathbb{V}, w \in \mathbb{W}$: $All\text{–}labels_p\,(v,r,t) \subseteq All\text{–}labels_p\,(v,r,t-1)$ and $All\text{–}labels_p\,(w,r,t) \subseteq All\text{–}labels_p\,(w,r,t-1)$.*

Disintegrated storage Intuitively, in disintegrated storage register emulations, for a `read`$_p$ to return v, p must encounter $\tau > 1$ blocks corresponding to v that were produced by separate `update` events. To formalize this, we use labels:

Definition 2 (τ-disintegrated storage)**.** *If a return of $v \in \mathbb{V}$ by a `read`$_p$ invocation is enabled at time t in a run r then $|L\text{–}labels_p\,(v,r,t)| \geq \tau$.*

Thus, a reader can only return v if it recalls (in its local memory) obtaining blocks of v with τ different labels.

A more restrictive case of τ-disintegrated storage occurs when readers cannot identify whether two blocks pertain to a common value unless they are produced by a common write that identifies them, e.g., with the same timestamp. This is the case when value parts or code words, rather than full replicas, are stored in objects.

To capture this case, for a block $b \in \bigcup_{e\in O\cup\Pi} e.data$, a value $v \in \mathbb{V}$, and a `write` $w \in \mathbb{W}_v$, if $\exists k \in \mathbb{N}$ s.t. $\langle w,k\rangle \in Labels(b)$, we say that w is an *origin write* of b and v is an *origin value* of b. Common write τ-disintegrated storage is then defined:

Definition 3 (common write τ-disintegrated storage)**.** *If a return of $v \in \mathbb{V}$ by a* `read`$_p$ *invocation is enabled at time t in a run r then $\exists w \in \mathbb{W}_v \ : \ |L\text{–}labels_p\,(w, r, t)| \geq \tau$.*

Note that we do not further require $p.data$ to actually hold τ blocks with a common write, because the weaker definition suffices for our lower bounds. For brevity, we henceforth refer to a common write τ-disintegrated storage algorithm simply as τ-common write.

Permanence Our lower bounds will all stem, in one way or another, from the observation that in wait-free disintegrated storage, every run must reach a point after which some values (and in the case of common write, also some `write`s) must *permanently* have a certain number of blocks in the shared storage. This is captured by the following definition:

Definition 4 (permanence)**.** *Consider a finite run r, $k \in \mathbb{N}$, a set $S \subseteq \mathbb{V}$, and a set of readers $\Theta \subset \Pi$. Let $z \in \mathbb{V} \cup \mathbb{W}$ be a value or a* `write` *operation. We say that z is* $\langle k,\ \Theta,\ S\rangle$-permanent *in r if in every finite extension r' of r s.t. in $r' \setminus r$ readers in Θ do not take actions and* `write`*s are limited to values from S, $|S\text{–}labels\,(z, r', t_{r'})| \geq k$.*

Intuitively, this means that the shared storage continues to hold k blocks of z as long as readers in Θ do not signal to the writer and only values from S are written. For brevity, when the particular sets S and Θ are not important, we refer to the value shortly as k-permanent. The observation below follows immediately from the definition of permanence:

Observation 5. *Let $v \in \mathbb{V}$, $w \in \mathbb{W}_v$, $k \in \mathbb{N}$, $V_2 \subseteq V_1 \subseteq \mathbb{V}$, $\Theta_1 \subseteq \Theta_2 \subset \Pi$.*

1. *If w is $\langle k,\ \Theta_1,\ V_1\rangle$-permanent in a finite run r then v is $\langle k,\ \Theta_1,\ V_1\rangle$-permanent in r.*
2. *If v is $\langle k,\ \Theta_1,\ V_1\rangle$-permanent in a finite run r then v is $\langle k,\ \Theta_2,\ V_2\rangle$-permanent in all finite extensions r' of r where in $r' \setminus r$* `write`*s are limited to values from V_1 and readers in Θ_1 do not take actions.*

Since each object holds a single block associated with a single label:

Observation 6. *For time t in a run r, the number of objects is: $n \geq \left|\bigcup_{v \in \mathbb{V}} S\text{–}labels\,(v, r, t)\right|$.*

Thus, if there are m different k-permanent values in a run, then $n \geq mk$. We observe that with invisible readers, the set Θ is immaterial:

Observation 7. *Consider $k \in \mathbb{N}$, $V \subseteq \mathbb{V}$, and a finite run r with an invisible reader $p \in \Pi$. If $z \in \mathbb{V} \cup \mathbb{W}$ is $\langle k,\ \{p\},\ V\rangle$-permanent in r then z is $\langle k,\ \emptyset,\ V\rangle$-permanent in r.*

The specific lower bounds for the four scenarios we consider differ in the number of permanent values/`write`s and the number of blocks per value/`write` ($k = \tau - 1$ or $k = \tau$) we can force the shared storage to retain forever in each case. Interestingly, our notion of permanence resembles the idea that an algorithm is not *amnesic* introduced in [14] (see Section 1.3), but is more fine-grained in specifying the number of permanent blocks and restricting executions under which they are retained.

Chapter 4

Lower bounds for disintegrated storage

In this chapter we provide lower bounds on the number of objects required for τ-disintegrated storage regular wait-free register emulations. Section 4.1 proves two general properties of regular wait-free τ-disintegrated storage algorithms. We show in Section 4.2 that with invisible `read`s, unless the readers' local storage size is exponential in D, the storage cost of such emulations is at least exponential in D. Finally, Section 4.3 shows that if `read`s are visible, then the storage cost increases linearly with the number of readers.

4.1 General properties

We first show that because readers must make progress even if the writer stops taking steps, at least $2\tau - 1$ blocks are required regardless of the number of readers.

Claim 8. *Consider $v_1, v_2 \in \mathbb{V}$ and a run r of a wait-free regular τ-disintegrated storage algorithm with two consecutive responded* `write`*s $w_1 \in \mathbb{W}_{v_1}$ followed by $w_2 \in \mathbb{W}_{v_2}$. Let $p \in \Pi$ be a reader s.t. no* `read`$_p$ *is pending in r. Then there is a time t between the returns of w_1 and w_2 when $|S\text{–}labels\,(v_1, r, t)| \geq \tau$ and $|S\text{–}labels\,(v_2, r, t)| \geq \tau - 1$.*

Proof. We first argue that at the time t_i, $i \in \{1, 2\}$ when w_i returns, $|S\text{–}labels\,(v_i, r, t_i)| \geq \tau$. Assume the contrary. We build a run r' identical to r up to t_i. In r', only process p performs actions after time t_i. Next, invoke a `read`$_p$ operation rd. By regularity and wait-freedom, rd must return v_i. Before performing actions on objects, $p.data\,(r', t_i)$ is empty, thus, from τ-disintegrated storage, p must encounter at least τ blocks with an origin value of v_i in order to return it. Since no process other than p takes actions, $|S\text{–}labels\,(v_i, r', t')| < \tau$ for all $t' \geq t_i$ onward, so rd cannot find these blocks and does not return v_i, a contradiction. It follows that in r' at t_i, and hence also in r at t_i, $|S\text{–}labels\,(v_i, r, t_i)| \geq \tau$.

Next, if at t_1, $|S\text{–}labels\,(v_2, r, t_1)| \geq \tau - 1$ then we are done. Otherwise, observe that objects are accessed one-at-a-time. Therefore, and since $|S\text{–}labels\,(v_2, r, t_1)| < \tau - 1$, there exists a time t between t_1 and t_2 when $|S\text{–}labels\,(v_2, r, t)| = \tau - 1$.

Finally, assume that $|S\text{–}labels\,(v_1, r, t)| < \tau$. Build a run r'' identical to r up to t, where again only p takes actions after t. As above, it follows by regularity, τ-disintegrated storage, and $p.data\,(r'', t) = \emptyset$, that rd never returns, in violation of wait-freedom. It follows that $|S\text{–}labels\,(v_1, r'', t)| = |S\text{–}labels\,(v_1, r, t)| \geq \tau$. □

The following lemma states that every non-empty set V can be split into two disjoint subsets, where one contains a value that is $(\tau - 1)$-permanent with respect to the other subset. The idea is to show that in the absence of such a value, a reader's accesses to the shared storage may be scheduled in a way that prevents the reader from obtaining τ labels of the same value. The logic of the proof is the following: we restrict `write`s to a set of values V, and consider the set S of values with blocks in $p.data \cap V$. If no value in S is $(\tau-1)$-permanent, then we can bring the shared storage to a state where none of the values in S have τ labels, preventing the reader from obtaining the τ labels required to return. By regularity, readers cannot return other values. The formal proof is slightly more subtle, because it needs to consider $L\text{–}labels_p$ as well as labels in the shared storage. It shows that the total number of labels of values in S (in both the shared and local storage) remains below τ whenever p takes a step.

Lemma 9. *Consider a non-empty set of values $V \subseteq \mathbb{V}$, a set of readers $\Theta \subset \Pi$, a reader $p \in \Pi \setminus \Theta$, and a finite run r of a wait-free regular τ-disintegrated storage algorithm. Then there is a subset $S \subseteq V$ of size $1 \leq |S| \leq L$ and an extension r' of r where some value $v \in S$ is $\langle \tau-1,\ \Theta \cup \{p\},\ V \setminus S \rangle$-permanent and s.t. in $r' \setminus r$ `write`s are limited to values from V and readers in Θ do not take steps.*

Proof. Assume by contradiction that the lemma does not hold. We construct an extension r' of r where a $\texttt{read}_p$ operation includes infinitely many actions of p yet does not return. To this end, we show that the following property holds at specific times in $r' \setminus r$:

$$\varphi(\hat{r}, t) \ \triangleq\ \forall v \in values_p(\hat{r}, t) \cap V \ :\ |All\text{–}labels_p(v, \hat{r}, t)| < \tau.$$

First, extend r to r_0 by returning any pending $\texttt{read}_p$ and `write`, invoking and returning a $\texttt{write}(v_0)$ for some $v_0 \in V$ (the operations eventually return, by wait-freedom), and finally invoking a $\texttt{read}_p$ operation rd without allowing it to take actions. We now prove by induction that for all $k \in \mathbb{N}$, there exists an extension r' of r_0 where (1) $\varphi(r', t_{r'})$ holds and in $r' \setminus r$: (2) `write`s are restricted to values from V, (3) p performs k actions on objects following rd's invocation, and (4) rd's return is not enabled, and (5) processes in Θ do not take steps.

Base: for $k = 0$, consider $r' = r_0$. (3,5) hold trivially. (2) holds since the only `write` in $r' \setminus r$ is of $v_0 \in V$. Since p performs no actions following the invocation of rd, $p.data(r', t_{r'})$ is empty. Therefore, (1) $\varphi(r', t_{r'})$ is vacuously true, and $L\text{–}labels_p(v, r, t)$ is empty for all $v \in \mathbb{V}$, thus (4) rd's return is not enabled by τ-disintegrated storage.

Step: assume inductively such an extension r_1 of r_0 with $k \geq 0$ actions performed by p following rd's invocation. Since rd cannot return, by wait-freedom, an action $\mathtt{a}_p$ is enabled on some object. We construct an extension r_2 of r_1 by letting $\mathtt{a}_p$ occur at time t_{r_1}. We consider two cases:

1. p does not obtain a block with an origin value in $V \setminus values_p(r_1, t_{r_1})$ at $\mathtt{a}_p$. Thus $values_p(r_2, t_{r_2}) \cap V \subseteq values_p(r_1, t_{r_1}) \cap V$. Then, by Observation 1 and the inductive hypothesis, (1) $\varphi(r_2, t_{r_2})$ holds and thus, by τ-disintegrated storage, rd cannot return any value $v \in values_p(r_2, t_{r_2}) \cap V$ at t_{r_2}. (4) It cannot return any other value in $values_p(r_2, t_{r_2})$ by regularity, and r_2 satisfies the induction hypothesis for $k+1$, as (2,3,5) trivially hold.

2. p obtains a block with origin value $u \in V \setminus values_p(r_1, t_{r_1})$ at time t_{r_1}. Then $|L\text{–}labels_p(u, r_2, t_{r_2})| = 1$. By Observation 1 and the inductive hypothesis, for all $v \in values_p(r_2, t_{r_2}) \setminus \{u\}$, $|L\text{–}labels_p(v, r_2, t_{r_2})| < \tau$, and thus rd's return is not enabled at time t_{r_2} by τ-disintegrated storage and regularity.

Let $S = values_p(r_2, t_{r_2}) \cap V$, and note that $|S| \geq 1$ (since $u \in S$) and that $|S| \leq |p.data| \leq L$. By the contradicting assumption, u is not $\langle \tau-1,\ \Theta \cup \{p\},\ V \setminus S \rangle$-permanent in r_2, thus there exists an extension r_3 of r_2 s.t. $|\textit{S–labels}\,(u, r_3, t_{r_3})| < \tau - 1$ and in $r_3 \backslash r_2$ `write`s are limited to values from $V \setminus S$ and no readers in $\Theta \cup \{p\}$ take steps (3,5 hold). Since p takes no steps in $r_3 \backslash r_2$, we have that $\textit{L–labels}_p\,(u, r_3, t_{r_3}) = \textit{L–labels}_p\,(u, r_2, t_{r_2})$, yielding:

$$|\textit{All–labels}_p\,(u, r_3, t_{r_3})| \le |\textit{L–labels}_p\,(u, r_2, t_{r_2})| + |\textit{S–labels}\,(u, r_3, t_{r_3})| < 1 + (\tau - 1) = \tau. \tag{4.1}$$

All `write`s invoked after t_{r_2} are from $\mathbb{W}_{V \setminus S}$ (2 holds), and therefore do not produce new labels associated with values in S. Since no values in S are written after t_{r_1} and readers' actions do not affect the sets *S–labels*, by Observation 1, we have that $\forall v \in S$, $\textit{All–labels}_p\,(v, r_3, t_{r_3}) \subseteq \textit{All–labels}_p\,(v, r_1, t_{r_1})$, and since $\varphi\,(r_1, t_{r_1})$ holds (inductively) and $S \setminus \{u\} \subseteq \textit{values}_p\,(r_1, t_{r_1}) \cap V$,

$$\forall v \in S \setminus \{u\} \,:\, |\textit{All–labels}_p\,(v, r_3, t_{r_3})| \le |\textit{All–labels}_p\,(v, r_1, t_{r_1})| < \tau. \tag{4.2}$$

From Equations 4.1 and 4.2, and since $\textit{values}_p\,(r_3, t_{r_3}) \cap V = \textit{values}_p\,(r_2, t_{r_2}) \cap V = S$, we get $\varphi\,(r_3, t_{r_3})$ (1). Since *rd*'s return was not enabled at time t_{r_2} and it took no actions since, its return is still not enabled (4), and we are done. □

4.2 Invisible reads

We now consider a setting of a single reader and single writer where `read`s are invisible. To show the following theorem, we "blow up" the shared storage by repeatedly invoking Lemma 9, each time adding one more $(\tau - 1)$-permanent value, yielding the following bound:

Theorem 10. *The storage cost of a regular τ-disintegrated storage wait-free SRSW register emulation where* `read`*s are invisible is at least $\tau + (\tau - 1)\left\lceil \frac{2^D - 1}{L} \right\rceil$ blocks.*

When readers are invisible, the set Θ is of no significance, so we consider $\emptyset$. Given a set of values V, the value added by Lemma 9 is $\langle \tau - 1,\ \emptyset,\ V \setminus S\rangle$-permanent for a smaller set of values $V \setminus S$ where $|S| \le L$. Therefore, we can invoke Lemma 9 $m = \left\lceil \frac{2^D - 1}{L} \right\rceil - 1$ times before running out of values, showing the following:

Lemma 11. *Let $p \in \Pi$ be an invisible reader. There exist finite runs $r_0, ..., r_m$ and sets of values $V_0 \supset V_1 \supset ... \supset V_m$ and $U_0 \subset U_1 \subset ... \subset U_m$, such that for all $0 \le k \le m$:*

1. *$|V_k| \ge 2^D - Lk$, $|U_k| = k$, $V_k \cap U_k = \emptyset$, and*
2. *all elements of U_k are $\langle \tau - 1,\ \emptyset,\ V_k\rangle$-permanent in r_k.*

Proof. By induction. Base: r_0 is the empty run, $V_0 = \mathbb{V}$ and $U_0 = \emptyset$. Assume inductively that the lemma holds for $k < m$. Since $m < \frac{2^D - 1}{L}$, we get: $|V_k| > 2^D - L\frac{2^D - 1}{L} = 1$. Since V_k is non-empty and $|\emptyset| < R$, by Lemma 9 there exist an extension r_{k+1} of r_k where `write`s in $r_{k+1} \setminus r_k$ are limited to values from V_k, a set $S \subset V_k$, $1 \le |S| \le L$, and a value $v \in S$ that is $\langle \tau - 1,\ \{p\},\ V_k \setminus S\rangle$-permanent in r_{k+1}.

Let $V_{k+1} = V_k \setminus S$ and $U_{k+1} = U_k \cup \{v\}$. Note that, because $V_k \cap U_k = \emptyset$ and $v \in S \subset V_k$, we get that $V_{k+1} \cap U_{k+1} = \emptyset$ and $|U_{k+1}| = |U_k| + 1 = k + 1$. Since $1 \le |S| \le L$ we have that $V_k \supset V_{k+1}$ and $|V_{k+1}| = |V_k| - |S| \ge 2^D - L(k + 1)$. By the inductive assumption and Observation 5, all values in U_k are $\langle \tau - 1,\ \emptyset,\ V_{k+1}\rangle$-permanent in r_{k+1}. By Observation 7, v is also $\langle \tau - 1,\ \emptyset,\ V_{k+1}\rangle$-permanent in r_{k+1} and we are done. □

Our bound combines the $2\tau-1$ blocks of Claim 8 with the $(\tau-1)m$ from Lemma 11:

Proof (Theorem 10). Consider an invisible reader $p \in \Pi$ and construct r_m, V_m, and U_m as in Lemma 11. Note that V_m contains at least two distinct values that are not in U_m, since $V_m \cap U_m = \emptyset$ and $|V_m| \geq 2^D - Lm > 2^D - L\frac{2^D-1}{L} = 1$. Extend r_m to r_{m+1} by invoking and returning `write`(v) and `write`(v') for $v, v' \in V_m$.

By Claim 8, there is a time $t \geq t_{r_m}$ in r_{m+1} when there are $2\tau - 1$ blocks in the shared storage with origin values of v or v'. In addition, by Lemma 11, U_m consists of m values that are $\langle \tau-1,\ \emptyset,\ V_m \rangle$-permanent in r_m, and since `write`s in $r_{m+1} \setminus r_m$ are of values from V_m, the values in U_m remain $\langle \tau-1,\ \emptyset,\ V_m \rangle$-permanent in r_{m+1}. By Observation 6:

$$n \geq 2\tau - 1 + (\tau-1)m = \tau + (\tau-1)(m+1) = \tau + (\tau-1)\left\lceil \frac{2^D-1}{L} \right\rceil. \qquad \square$$

4.3 Visible reads

We now consider systems where readers may write meta-data in the shared storage. We use a similar technique as in Lemma 11, except that due to readers' `update`s, the indistinguishability argument can no longer be used. Instead, we invoke a new reader for each extension, and therefore the number of runs might be limited by the number of readers, R:

Theorem 12. *The storage cost of a regular τ-disintegrated storage wait-free MRSW register emulation with R readers is at least $\tau + (\tau-1) \cdot \min\left(\left\lceil \frac{2^D-1}{L} \right\rceil,\ R\right)$ blocks.*

To achieve this bound, we use Lemma 9 again to construct $N = \min\left(\left\lceil \frac{2^D-1}{L} \right\rceil,\ R\right) - 1$ extensions of the empty run (note that it does not assume invisible `read`s).

Lemma 13. *There exist finite runs $r_0, ..., r_N$, sets of values $V_0 \supset V_1 \supset ... \supset V_N$ and $U_0 \subset U_1 \subset ... \subset U_N$, and sets of readers $\Theta_0 \subset \Theta_1 \subset ... \subset \Theta_N$, such that for all $0 \leq k \leq N$:*

1. *$|V_k| \geq 2^D - Lk$, $|U_k| = |\Theta_k| = k$, $V_k \cap U_k = \emptyset$, and*
2. *all elements of U_k are $\langle \tau-1,\ \Theta_k,\ V_k \rangle$-permanent in r_k.*

Proof. By induction. Base: r_0 is the empty run, $V_0 = \mathbb{V}$, $\Theta_0 = U_0 = \emptyset$. Assume inductively such r_k, V_k, U_k, and Θ_k for $k < N$, and construct r_{k+1} as follows: since $R - |\Theta_k| > 0$, there is a reader $p \in \Pi \setminus \Theta_k$. Since $N < \frac{2^D-1}{L}$, we get $|V_k| > 2^D - LN > 1$. Therefore, by Lemma 9, there exist an extension r_{k+1} of r_k where in $r_{k+1} \setminus r_k$ `write`s are limited to values from V_k and readers in Θ_k do not take steps, a set $S \subseteq V_k$, $1 \leq |S| \leq L$, and a value $v \in S$ that is $\langle \tau-1,\ \Theta_k \cup \{p\},\ V_k \setminus S \rangle$-permanent in r_{k+1}.

Let $V_{k+1} = V_k \setminus S$ and $U_{k+1} = U_k \cup \{v\}$. Note that, because $V_k \cap U_k = \emptyset$ and $v \in S \subset V_k$, it follows that $V_{k+1} \cap U_{k+1} = \emptyset$ and $|U_{k+1}| = k+1$. Furthermore, since $1 \leq |S| \leq L$, we get: $V_k \supset V_{k+1}$ and $|V_{k+1}| \geq |V_k| - |S| \geq 2^D - L(k+1)$. Finally, let $\Theta_{k+1} = \Theta_k \cup \{p\}$. By the inductive assumption and Observation 5, all values in U_k are $\langle \tau-1,\ \Theta_{k+1},\ V_{k+1} \rangle$-permanent in r_{k+1}, and so all of U_{k+1} is $\langle \tau-1,\ \Theta_{k+1},\ V_{k+1} \rangle$-permanent in r_{k+1}, as needed. $\square$

From Lemma 13, in r_N there is a set of N $(\tau-1)$-permanent values, inducing a cost of $(\tau-1)N$. We use Claim 8 to increase the bound by $2\tau-1$ additional blocks.

Proof (Theorem 12). Construct r_N, V_N, U_N, and Θ_N as in Lemma 13. Note that, since $R-N \geq 1$, there exists $p \in \Pi \setminus \Theta_N$. Since $V_N \cap U_N = \emptyset$ and $|V_N| \geq 2^D - LN > 2^D - L\frac{2^D-1}{L} = 1$, $V_N \setminus U_N$ contains at least two values. Extend r_N to r_{N+1} by invoking and returning write(v) and write(v') for $v, v' \in V_N \setminus U_N$.

By Claim 8, there is a time $t \geq t_{r_N}$ in r_{N+1} when there are $2\tau-1$ blocks in the shared storage with origin values of v or v'. U_N consists of N additional values that are $\langle \tau-1,\ \Theta_N,\ V_N\rangle$-permanent in r_N, and since in $r_{N+1} \setminus r_N$ writes are of values from V_N and no reader in Θ_N takes steps, the values in U_N remain $\langle \tau-1,\ \Theta_N,\ V_N\rangle$-permanent in r_{N+1}. By Observation 6, the storage cost is:

$$n \geq 2\tau - 1 + (\tau-1)N = \tau + (\tau-1)(N+1) = \tau + (\tau-1)\cdot \min\left(\left\lceil \frac{2^D-1}{L} \right\rceil, R\right). \qquad \square$$

Chapter 5

Lower bounds for common write disintegrated storage

While the results of the previous chapter hold *a fortiori* for τ-common write algorithms, for this case we are able to show stronger results, independent of the local storage size. Intuitively, this is because readers can no longer reuse blocks they obtained from previous `write`s of the same value, and so we can prolong the execution that blows up the shared storage by rewriting values. Section 5.1 proves a general attribute of τ-common write algorithms. We show in Section 5.2 that even with a single reader (and a single writer), if `read`s are invisible, then the required storage cost is at least $\tau \cdot 2^D$. In Section 5.3 we prove a bound for visible `read`s.

5.1 General observation

In this section we define a property that is a special case of k-permanence, which additionally requires that the set of labels associated with a `write` does not change.

Definition 14 (Constancy)**.** *Consider a finite run r, $k \in \mathbb{N}$, a set $S \subseteq \mathbb{V}$, and a set of readers $\Theta \subset \Pi$. We say that a* `write` *$w \in \mathbb{W}$ is $\langle k,\ \Theta,\ S\rangle$-constant in r if in every finite extension r' of r s.t. in $r' \setminus r$ readers in Θ do not take actions and* `write`*s are limited to values from S, S–labels$(w, r', t_{r'}) = S$–labels(w, r, t_r) and $|S\text{–labels}(w, r', t_{r'})| = k$.*

Similarly to Observation 7, it can be shown that:

Observation 15. *Consider $V \subseteq \mathbb{V}$, $k \in \mathbb{N}$, and a finite run r with an invisible reader $p \in \Pi$. If $w \in \mathbb{W}$ is $\langle k,\ \{p\},\ V\rangle$-constant in r then w is $\langle k,\ \emptyset,\ V\rangle$-constant in r.*

We next prove a stronger variant of Lemma 9 that allows us to add a permanent `write` to the shared storage while some set $C \subseteq \mathbb{W}$ of `write`s are constant. Note that since the number of `write`s of a value v is infinite and the number of constant `write`s in a finite run is finite, for any non-empty $V \subseteq \mathbb{V}$, $\mathbb{W}_V \setminus C$ is non-empty.

Lemma 16. *Consider a non-empty set of values $V \subseteq \mathbb{V}$, a set of readers $\Theta \subset \Pi$, a reader $p \in \Pi \backslash \Theta$, and a finite run r of a wait-free regular τ-common write algorithm. Let C be a set of* `write`*s that are $\langle \tau - 1,\ \Theta,\ V\rangle$-constant in r. Then there is an extension r' of r where some $w \in \mathbb{W}_V \setminus C$ returns and is $\langle \tau - 1,\ \Theta \cup \{p\},\ V\rangle$-permanent, and s.t. in $r' \setminus r$* `write`*s are limited to $\mathbb{W}_V$ and readers in Θ do not take actions.*

Proof. Assume by way of contradiction that the lemma does not hold. We build an extension r' of r where a $\texttt{read}_p$ operation performs infinitely many actions of p yet does not return. To this end, we show that the following property holds at specific times in $r' \setminus r$:

$$\psi(\hat{r}, t) \triangleq \forall w \in \mathit{writes}_p(\hat{r}, t) \cap \mathbb{W}_V : |\mathit{All\text{–}labels}_p(w, \hat{r}, t)| < \tau.$$

Note that, by definitions of τ-common write and of *All–labels*, whenever $\psi(r', t)$ holds, no pending $\texttt{read}_p$ invocation can return a value $v \in \mathit{values}_p(r', t) \cap V$.

First, extend r to r_0 by returning any pending $\texttt{read}_p$ and $\texttt{write}$, invoking and returning a $\texttt{write}(v_0)$ for some $v_0 \in V$ (the operations eventually return, by wait-freedom), and finally invoking a $\texttt{read}_p$ operation rd without allowing it to take actions. We now prove by induction that for all $k \in \mathbb{N}$, there exists an extension r' of r_0 where (1) $\psi(r', t_{r'})$ holds, (2) no $\texttt{write}$ is pending at $t_{r'}$, and in $r' \setminus r$: (3) $\texttt{write}$s are restricted to $\mathbb{W}_V$, (4) p performs k actions on objects after invoking rd, (5) rd's return is not enabled, and (6) processes in Θ do not take steps.

Base: for $k = 0$, consider $r' = r_0$. (2,4,6) hold trivially. (3) holds since the only $\texttt{write}$ in $r' \setminus r$ is $w_0 \in \mathbb{W}_V$. Since p performs no actions following the invocation of rd, $p.\mathit{data}(r', t_{r'})$ is empty. Therefore, (1) $\psi(r', t_{r'})$ is vacuously true, and $\mathit{L\text{–}labels}_p(w, r', t_{r'})$ is empty for all $w \in \mathbb{W}_{\mathbb{V}}$, thus (5) rd's return is not enabled by τ-common write.

Step: assume inductively such an extension r_1 of r_0 with $k \geq 0$ actions by p following rd's invocation. Since rd cannot return, by wait-freedom, an action $\mathtt{a}_p$ is enabled on some object. We construct an extension r_2 of r_1 by letting $\mathtt{a}_p$ occur at time t_{r_1}. We then consider three cases:

1. p does not obtain a block with an origin $\texttt{write}$ in $\mathbb{W}_V \setminus \mathit{writes}_p(r_1, t_{r_1})$ at $\mathtt{a}_p$. Thus $(\mathit{writes}_p(r_2, t_{r_2}) \cap \mathbb{W}_V) \subseteq (\mathit{writes}_p(r_1, t_{r_1}) \cap \mathbb{W}_V)$. Then, by Observation 1 and the inductive hypothesis, (1) $\psi(r_2, t_{r_2})$ holds and thus, by τ-common write, rd cannot return any value $v \in \mathit{values}_p(r_2, t_{r_2}) \cap V$ at t_{r_2}. (5) It cannot return any other value in $\mathit{values}_p(r_2, t_{r_2})$ by regularity, and r_2 satisfies the induction hypothesis for $k+1$ as (2,3,4,6) trivially hold.

2. p obtains a block with origin $\texttt{write}$ $w' \in C \cap \mathbb{W}_V \setminus \mathit{writes}_p(r_1, t_{r_1})$ at $\mathtt{a}_p$. Then $|\mathit{L\text{–}labels}_p(w', r_1, t_{r_1})| = 0$. Since w' is $\langle \tau - 1,\ \Theta,\ V \rangle$-constant in r and in $r_1 \setminus r$ $\texttt{write}$s are restricted to $\mathbb{W}_V$ and processes in Θ do not take steps (inductively), then by definition of constancy, $|\mathit{S\text{–}labels}(w', r_1, t_{r_1})| = \tau - 1$. By Observation 1, for all $w \in \mathit{writes}_p(r_2, t_{r_2}) \cap \mathbb{W}_V$: $\mathit{All\text{–}labels}_p(w, r_2, t_{r_2}) \subseteq \mathit{All\text{–}labels}_p(w, r_1, t_{r_1})$. Therefore $|\mathit{All\text{–}labels}_p(w', r_2, t_{r_2})| \leq |\mathit{L\text{–}labels}_p(w', r_1, t_{r_1})| + |\mathit{S\text{–}labels}(w', r_1, t_{r_1})| = \tau - 1$. Together with the inductive hypothesis, $\forall w \in \mathit{writes}_p(r_2, t_{r_2}) \cap \mathbb{W}_V \setminus \{w'\}$, $|\mathit{All\text{–}labels}_p(w, r_2, t_{r_2})| \leq |\mathit{All\text{–}labels}_p(w, r_1, t_{r_1})| < \tau$; $\psi(r_2, t_{r_2})$ follows, thus (5) follows, and (2,3,4,6) trivially hold.

3. p obtains a block with origin $\texttt{write}$ $w' \in \mathbb{W}_V \setminus (\mathit{writes}_p(r_1, t_{r_1}) \cup C)$ at $\mathtt{a}_p$. Then $|\mathit{L\text{–}labels}_p(w', r_2, t_{r_2})| = 1$ and the number of labels of other $\texttt{write}$s in $\mathit{writes}_p(r_2, t_{r_2})$ does not increase following $\mathtt{a}_p$, thus rd's return is not enabled at t_{r_2} by τ-common write and regularity.

By the contradicting assumption, w' is not $\langle \tau - 1,\ \Theta \cup \{p\},\ V \rangle$-permanent in r_2, thus there is an extension r_3 of r_2 s.t. $|\mathit{S\text{–}labels}(w', r_3, t_{r_3})| < \tau - 1$ and in $r_3 \setminus r_2$ $\texttt{write}$s are limited to $\mathbb{W}_V$ and no readers in $\Theta \cup \{p\}$ take steps. We further extend r_3 to r_4 by letting any pending $\texttt{write}$ return (2,3,4,6 hold).

Let $S = \mathit{writes}_p(r_2, t_{r_2}) \cap \mathbb{W}_V$. Since every $w \in S$ returns before t_{r_2} by the inductive assumption, the `writes` in $r_4 \setminus r_2$ do not produce new labels associated with w. Since readers do not affect the sets *S–labels*, it follows that $\forall w \in S : \mathit{S\text{–}labels}(w, r_4, t_{r_4}) \subseteq \mathit{S\text{–}labels}(w, r_3, t_{r_3}) \subseteq \mathit{S\text{–}labels}(w, r_2, t_{r_2})$. Next, p takes no steps in $r_4 \setminus r_2$ (4 holds), thus $\forall w \in S : \mathit{L\text{–}labels}_p(w', r_4, t_{r_4}) = \mathit{L\text{–}labels}_p(w', r_2, t_{r_2})$. It follows that:

$$\left|\mathit{All\text{–}labels}_p\left(w', r_4, t_{r_4}\right)\right| \le \left|\mathit{L\text{–}labels}_p\left(w', r_2, t_{r_2}\right)\right| + \left|\mathit{S\text{–}labels}\left(w', r_3, t_{r_3}\right)\right| < 1 + (\tau - 1) = \tau. \quad (5.1)$$

Moreover, by Observation 1 and the inductive assumption that $\psi(r_1, t_{r_1})$ holds,

$$\forall w \in S \setminus \{w'\} : |\mathit{All\text{–}labels}_p(w, r_4, t_{r_4})| \le |\mathit{All\text{–}labels}_p(w, r_1, t_{r_1})| < \tau. \quad (5.2)$$

From Equations 5.1 and 5.2, and since $\mathit{writes}_p(r_4, t_{r_4}) \cap \mathbb{W}_V = \mathit{writes}_p(r_2, t_{r_2}) \cap \mathbb{W}_V = S$, we get (1) $\psi(r_4, t_{r_4})$. Since rd's return is not enabled at t_{r_2} and p took no actions since, its return is not enabled anywhere in $r_4 \setminus r_1$ (5), and we are done. □

5.2 Invisible reads

We prove the following theorem by constructing a run with an exponential number of τ-permanent values. The idea is to show that if there is a value in the domain for which there is no τ-permanent `write`, then infinitely many `writes` remain $(\tau - 1)$-constant, which is of course impossible.

Theorem 17. *The storage cost of a regular τ-common write wait-free SRSW register emulation where* `reads` *are invisible is at least $\tau \cdot 2^D$ blocks.*

Lemma 18. *Consider a non-empty set of values $V \subseteq \mathbb{V}$ and a finite run r. Let C be a set of* `writes` *that are $\langle \tau - 1,\ \emptyset,\ V\rangle$-constant in r. Then there exists an extension r' of r where* `writes` *in $r' \setminus r$ are limited to $\mathbb{W}_V$, and some $w \in \mathbb{W}_V \setminus C$ is either $\langle \tau - 1,\ \emptyset,\ V\rangle$-constant or $\langle \tau,\ \emptyset,\ V\rangle$-permanent in r'.*

Proof. Let $p \in \Pi$ be a reader. By Lemma 16, there is an extension r' of r where `writes` in $r' \setminus r$ are limited to $\mathbb{W}_V$ and some $w \in \mathbb{W}_V \setminus C$ returns and is $\langle \tau - 1,\ \{p\},\ V\rangle$-permanent. By Observation 7, if w is $\langle \tau,\ \{p\},\ V\rangle$-permanent in r', then w is $\langle \tau,\ \emptyset,\ V\rangle$-permanent in r' and the lemma follows. Otherwise, there exists an extension r'' of r' where in $r'' \setminus r'$ `writes` are limited to $\mathbb{W}_V$ and p takes no steps, and $|\mathit{S\text{–}labels}(w, r'', t_{r''})| < \tau$. Since w is $\langle \tau - 1,\ \{p\},\ V\rangle$-permanent in r', $|\mathit{S\text{–}labels}(w, r'', t_{r''})| = \tau - 1$.

We show that w is $\langle \tau - 1,\ \emptyset,\ V\rangle$-constant in r''. Consider an extension r''' of r'' where `writes` are limited to values from V and p takes no steps in $r''' \setminus r''$. Since w has already returned by time $t_{r''}$, no new blocks with an origin `write` of w can be added to the shared storage in r''' after $t_{r''}$. It follows that $\mathit{S\text{–}labels}(w, r''', t_{r'''}) \subseteq \mathit{S\text{–}labels}(w, r'', t_{r''})$. However, since w is $\langle \tau - 1,\ \{p\},\ V\rangle$-permanent in r', and in $r''' \setminus r'$ `writes` are limited $\mathbb{W}_V$ and p takes no steps, then $|\mathit{S\text{–}labels}(w, r''', t_{r'''})| \ge \tau - 1 = |\mathit{S\text{–}labels}(w, r'', t_{r''})|$. It follows that $\mathit{S\text{–}labels}(w, r''', t_{r'''}) = \mathit{S\text{–}labels}(w, r'', t_{r''})$. Thus, w is $\langle \tau - 1,\ \{p\},\ V\rangle$-constant in r''. The lemma follows from Observation 15. □

Claim 19. *Consider a finite run r and a non-empty $V \subseteq \mathbb{V}$. Then there is an extension r' of r s.t.* `writes` *in $r' \setminus r$ are limited to $\mathbb{W}_V$, and some $w \in \mathbb{W}_V$ is $\langle \tau,\ \emptyset,\ V\rangle$-permanent in r'.*

Proof. Consider an algorithm with storage cost n, and let $m = \lceil n/(\tau-1) \rceil + 1$. Assume by contradiction that the claim does not hold. We get a contradiction by constructing $m+1$ extensions of r; $r_0, ..., r_m$ with sets of `writes` $C_0 \subset C_1 \subset \cdots \subset C_m \subseteq \mathbb{W}_V$ s.t. for all $0 \le k \le m$:

1. `writes` in $r_k \setminus r$ are limited to $\mathbb{W}_V$, and
2. C_k is a set of k `writes` that are $\langle \tau-1,\ \emptyset,\ V \rangle$-constant in r_k.

Note that in r_m, $\left\lceil \frac{n}{\tau-1} \right\rceil + 1$ `writes` are $\langle \tau-1,\ \emptyset,\ V \rangle$-constant, implying a storage cost greater than n by Observation 6, a contradiction.

The construction is by induction. The base case vacuously holds for $r_0 = r$, $C_0 = \emptyset$. Assume inductively that such r_k and C_k exist for $k < m$. By Lemma 18 there exists an extension r_{k+1} of r_k where some $w \in \mathbb{W}_V \setminus C_k$ is either $\langle \tau,\ \emptyset,\ V \rangle$-permanent or $\langle \tau-1,\ \emptyset,\ V \rangle$-constant, and `writes` in $r_{k+1} \setminus r_k$ are limited to $\mathbb{W}_V$. Since all `writes` in C_k are $\langle \tau-1,\ \emptyset,\ V \rangle$-constant in r_k they are also $\langle \tau-1,\ \emptyset,\ V \rangle$-constant in r_{k+1}. By the contracting assumption, w is not $\langle \tau,\ \emptyset,\ V \rangle$-permanent in r_{k+1} hence is $\langle \tau-1,\ \emptyset,\ V \rangle$-constant in the run. Let $C_{k+1} \triangleq C_k \cup \{w\}$. Therefore $|C_{k+1}| = k+1$ and all `writes` in C_{k+1} are $\langle \tau-1,\ \emptyset,\ V \rangle$-constant in r_{k+1}, as needed. □

We are now ready to prove our lower bound of $\tau \cdot 2^D$ blocks:

Proof (Theorem 17). We show that there exist $2^D + 1$ finite runs $r_0, r_1, \ldots, r_{2^D}$ and sets of values $V_0 \supset V_1 \supset \cdots \supset V_{2^D}$ and $U_0 \subset U_1 \subset \cdots \subset U_{2^D}$, such that for all $0 \le k \le 2^D$:

1. $|V_k| = 2^D - k$, $|U_k| = k$, $V_k \cap U_k = \emptyset$, and
2. all elements of U_k are $\langle \tau,\ \emptyset,\ V_k \rangle$-permanent in r_k.

By induction. Base: r_0 is the empty run, $V_0 = \mathbb{V}$, $U_0 = \emptyset$. Assume inductively that there exist such r_k, V_k, and U_k for $k < 2^D$, and construct r_{k+1} as follows: first, because $|V_k| = 2^D - k > 0$, by Claim 19 there is an extension r_{k+1} of r_k where `writes` in $r_{k+1} \setminus r_k$ are limited to $\mathbb{W}_{V_k}$ and some $w \in \mathbb{W}_{V_k}$ is $\langle \tau,\ \emptyset,\ V_k \rangle$-permanent.

Consider the value $v \in V_k$ written by w. By Observation 5, v is $\langle \tau,\ \emptyset,\ V_k \rangle$-permanent in r_{k+1}. Setting $V_{k+1} \triangleq V_k \setminus \{v\}$, we have that $|V_{k+1}| = |V_k| - 1 = 2^D - (k+1)$. Further let $U_{k+1} = U_k \cup \{v\}$. Note that, because $V_k \cap U_k = \emptyset$, we get $v \notin U_k$ and hence $V_{k+1} \cap U_{k+1} = \emptyset$ and $|U_{k+1}| = |U_k| + 1 = k+1$. Since $V_k \supset V_{k+1}$, then v is $\langle \tau,\ \emptyset,\ V_{k+1} \rangle$-permanent. Additionally, `writes` in $r_{k+1} \setminus r_k$ are from $\mathbb{W}_{V_k}$, thus by the inductive assumption and Observation 5, values in U_k are $\langle \tau,\ \emptyset,\ V_{k+1} \rangle$-permanent in r_{k+1}, and so all of U_{k+1} are $\langle \tau,\ \emptyset,\ V_{k+1} \rangle$-permanent in r_{k+1}.

Finally, U_{2^D} holds 2^D values that are $\langle \tau,\ \emptyset,\ \emptyset \rangle$-permanent in r_{2^D}. By Observation 6:

$$n \ge \tau \cdot 2^D.$$

□

5.3 Visible reads

To prove a lower bound on the cost of systems with visible `reads`, we create a similar construction, except that the number of extensions might be limited by the number of readers, R. Instead, the bound depends on $\min\left(2^D - 1\,,\ R\right)$:

Theorem 20. *The storage cost of a regular τ-common write wait-free MRSW register emulation is at least $\tau + (\tau - 1) \cdot \min\left(2^D - 1\,,\, R\right)$ blocks.*

Let $N = \min\left(2^D - 1\,,\, R\right) - 1$. We build a run with N $(\tau - 1)$-permanent values:

Lemma 21. *There exist finite runs $r_0, r_1, \ldots, r_N$, sets of values $V_0 \supset V_1 \supset \ldots \supset V_N$ and $U_0 \subset U_1 \subset \ldots \subset U_N$, and sets of readers $\Theta_0 \subset \Theta_1 \subset \ldots \subset \Theta_N$, s.t. for all $0 \le k \le N$:*

1. *$|V_k| = 2^D - k$, $|U_k| = |\Theta_k| = k$, $V_k \cap U_k = \emptyset$, and*
2. *all elements of U_k are $\langle \tau - 1,\ \Theta_k,\ V_k \rangle$-permanent in r_k.*

Proof. By induction. Base: r_0 is the empty run, $V_0 = \mathbb{V}$, $\Theta_0 = U_0 = \emptyset$. Assume inductively such r_k, V_k, U_k, and Θ_k for $k < N$, and construct r_{k+1} as follows: since $R - |\Theta_k| > 1$, there is a reader $p \in \Pi \setminus \Theta_k$. Moreover, $|V_k| > 2^D - N > 0$. Therefore, by Lemma 16, there is an extension r_{k+1} of r_k where `write`s in $r_{k+1} \setminus r_k$ are limited to $\mathbb{W}_{V_k}$, readers in Θ_k do not take steps in $r_{k+1} \setminus r_k$, and some $w \in W_{V_k}$ returns and is $\langle \tau - 1,\ \Theta_k \cup \{p\},\ V_k \rangle$-permanent in r_{k+1}.

Let $\Theta_{k+1} = \Theta_k \cup \{p\}$, and consider the value $v \in V_k$ written by w. By Observation 5, v is $\langle \tau - 1,\ \Theta_{k+1},\ V_k \rangle$-permanent. Let $V_{k+1} = V_k \setminus \{v\}$, then $|V_{k+1}| = 2^D - (k + 1)$. Further let $U_{k+1} = U_k \cup \{v\}$. Since $V_k \cap U_k = \emptyset$, we get that $V_{k+1} \cap U_{k+1} = \emptyset$ and $|U_{k+1}| = k + 1$.

Since $V_k \supset V_{k+1}$, v is $\langle \tau - 1,\ \Theta_{k+1},\ V_{k+1} \rangle$-permanent. In addition, in $r_{k+1} \setminus r_k$ `write`s are limited to $\mathbb{W}_{V_k}$ and readers in Θ_k do not take steps, and since $\Theta_k \subset \Theta_{k+1}$, then by the inductive assumption and Observation 5, all values in U_k are $\langle \tau - 1,\ \Theta_{k+1},\ V_{k+1} \rangle$-permanent. Therefore, all elements of U_{k+1} are $\langle \tau - 1,\ \Theta_{k+1},\ V_{k+1} \rangle$-permanent in r_{k+1}, as needed. □

From Lemma 21, in r_N there is a set of N $(\tau - 1)$-permanent values, inducing a cost of $(\tau - 1)N$. We use Claim 8 to increase the bound by $2\tau - 1$ additional blocks.

Proof (Theorem 20). Construct r_N, V_N, U_N, and Θ_N as in Lemma 21. Note that, since $R - N \ge 1$, there is a reader $p \in \Pi \setminus \Theta_N$. Since $V_N \cap U_N = \emptyset$ and $|V_N| = 2^D - N = 2^D - (\min\left(2^D - 1\,,\, R\right) - 1) \ge 2$, the set V_N contains two values, and they are not in U_N. Extend r_N to r_{N+1} by invoking and returning `write`(v) and `write`(v') for $v, v' \in V_N$.

By Claim 8, there is a time $t \ge t_{r_N}$ in r_{N+1} when there are $2\tau - 1$ blocks in the shared storage with origin values of v or v'. In addition, U_N consists of N values that are $\langle \tau - 1,\ \Theta_N,\ V_N \rangle$-permanent in r_N, and since in $r_{N+1} \setminus r_N$ `write`s are of values from V_N and no reader in Θ_N takes steps, the values in U_N remain $\langle \tau - 1,\ \Theta_N,\ V_N \rangle$-permanent in r_{N+1}. By Observation 6, the storage cost amounts to at least:

$$n \ge 2\tau - 1 + (\tau - 1)N = \tau + (\tau - 1)(N + 1) = \tau + (\tau - 1) \cdot \min\left(2^D - 1\,,\, R\right). \quad \square$$

Chapter 6

Discussion

We have shown lower bounds on the space complexity of regular wait-free τ-disintegrated storage algorithms. Although our bounds are stated in terms of blocks, there are scenarios where they entail concrete bounds in terms of bits. In replication, each block stores an entire value, and so the block sizes are D bits and $\tau \geq f+1$. Other applications use symmetric coding where all blocks are of equal size. Using a simple pigeonhole argument, it can be shown that in τ-disintegrated storage emulations that use symmetric coding and that are not $(\tau+1)$-disintegrated, the size of blocks is at least D/τ bits, yielding bounds of $D \cdot 2^D$ and $D + D\frac{\tau-1}{\tau} \cdot \min\left(2^D - 1\,,\, R\right)$ with invisible and visible readers, respectively.

Our lower bounds for the common write case explain, for the first time, why previous coded storage algorithms have either had the readers write or consumed exponential (or even unbounded) space. Similarly, they establish why previous emulations of large registers from smaller ones have either had the readers write, had the writer share blocks among different `write`s, or consumed exponential space.

Our work leaves several open questions. First, when replication is used as a means to overcome Byzantine faults or data corruption, our results suggest that there might be an interesting trade-off between the shared storage cost and the size of local memory at the readers, and a possible advantage to systems that apply replication rather than error correction codes: we have shown that, with invisible readers, the former require $\Omega(2^D/L)$ blocks, rather than the $\Omega(2^D)$ blocks needed by the latter. Whether there are algorithms that achieve this lower cost remains an open question. Second, it is unclear how the bounds would be affected by removing our assumption that each block in the shared storage pertains to a single write. Wei [27] has provided a partial answer to this question by showing that similar bounds hold without this assumption, but only in the case of emulating large registers from smaller ones *without* meta-data at all. Similarly, it would be interesting to study whether allowing readers to write data (and not only signals) impacts the storage cost. Finally, future work may consider additional sub-classes of disintegrated storage, e.g., with unresponsive objects, and show that additional costs are incurred in these cases.

Bibliography

[1] Ittai Abraham, Gregory Chockler, Idit Keidar, and Dahlia Malkhi. Byzantine disk paxos: optimal resilience with byzantine shared memory. *Distributed Computing*, 18(5):387–408, 2006.

[2] Ittai Abraham, Gregory Chockler, Idit Keidar, and Dahlia Malkhi. Wait-free regular storage from byzantine components. *Information Processing Letters*, 101(2):60–65, 2007.

[3] Marcos K. Aguilera, Burkhard Englert, and Eli Gafni. On using network attached disks as shared memory. In *Proceedings of the Twenty-second Annual Symposium on Principles of Distributed Computing*, PODC '03, pages 315–324, New York, NY, USA, 2003. ACM.

[4] Marcos Kawazoe Aguilera, Ramaprabhu Janakiraman, and Lihao Xu. Using erasure codes efficiently for storage in a distributed system. In *2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 336–345, June 2005.

[5] Elli Androulaki, Christian Cachin, Dan Dobre, and Marko Vukolić. Erasure-coded byzantine storage with separate metadata. In *International Conference on Principles of Distributed Systems*, pages 76–90. Springer, 2014.

[6] Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. Sharing memory robustly in message-passing systems. *Journal of the ACM (JACM)*, 42(1):124–142, January 1995.

[7] Rida A Bazzi and Yin Ding. Non-skipping timestamps for byzantine data storage systems. In *International Symposium on Distributed Computing*, pages 405–419. Springer, 2004.

[8] Alon Berger, Idit Keidar, and Alexander Spiegelman. Integrated bounds for disintegrated storage. *arXiv preprint arXiv:1805.06265*, 2018.

[9] Christian Cachin and Stefano Tessaro. Optimal resilience for erasure-coded byzantine distributed storage. In *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, pages 115–124. IEEE, 2006.

[10] Viveck R. Cadambe, Nancy Lynch, Muriel Medard, and Peter Musial. A coded shared atomic memory algorithm for message passing architectures. In *Network Computing and Applications (NCA), 2014 IEEE 13th International Symposium on*, pages 253–260. IEEE, 2014.

[11] Viveck R. Cadambe, Zhiying Wang, and Nancy Lynch. Information-theoretic lower bounds on the storage cost of shared memory emulation. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC '16, pages 305–313, New York, NY, USA, 2016. ACM.

[12] Soma Chaudhuri, Martha J Kosa, and Jennifer L Welch. One-write algorithms for multivalued regular and atomic registers. *Acta Informatica*, 37(3):161–192, 2000.

[13] Tian Ze Chen and Yuanhao Wei. Step Optimal Implementations of Large Single-Writer Registers. In Panagiota Fatourou, Ernesto Jiménez, and Fernando Pedone, editors, *20th International Conference on Principles of Distributed Systems (OPODIS 2016)*, volume 70 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[14] Gregory Chockler, Rachid Guerraoui, and Idit Keidar. Amnesic distributed storage. In *Distributed Computing*, pages 139–151. Springer, 2007.

[15] Gregory Chockler and Alexander Spiegelman. Space complexity of fault-tolerant register emulations. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, PODC '17, pages 83–92, New York, NY, USA, 2017. ACM.

[16] Dan Dobre, Ghassan Karame, Wenting Li, Matthias Majuntke, Neeraj Suri, and Marko Vukolić. Powerstore: proofs of writing for efficient and robust storage. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 285–298. ACM, 2013.

[17] Dan Dobre, Matthias Majuntke, and Neeraj Suri. On the time-complexity of robust and amnesic storage. In *International Conference On Principles Of Distributed Systems*, pages 197–216. Springer, 2008.

[18] Partha Dutta, Rachid Guerraoui, and Ron R. Levy. Optimistic erasure-coded distributed storage. In *Proceedings of the 22nd International Symposium on Distributed Computing*, DISC '08, pages 182–196, Berlin, Heidelberg, 2008. Springer-Verlag.

[19] Garth R Goodson, Jay J Wylie, Gregory R Ganger, and Michael K Reiter. Efficient byzantine-tolerant erasure-coded storage. In *Dependable Systems and Networks, 2004 International Conference on*, pages 135–144. IEEE, 2004.

[20] Prasad Jayanti, Tushar Deepak Chandra, and Sam Toueg. Fault-tolerant wait-free shared objects. *Journal of the ACM (JACM)*, 45(3):451–500, 1998.

[21] Leslie Lamport. On interprocess communication. *Distributed computing*, 1(2):86–101, 1986.

[22] Dahlia Malkhi and Michael Reiter. Byzantine quorum systems. *Distributed computing*, 11(4):203–213, 1998.

[23] Jean-Philippe Martin, Lorenzo Alvisi, and Michael Dahlin. Minimal byzantine storage. In *International Symposium on Distributed Computing*, pages 311–325. Springer, 2002.

[24] Gary L Peterson. Concurrent reading while writing. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 5(1):46–55, 1983.

[25] Alexander Spiegelman, Yuval Cassuto, Gregory Chockler, and Idit Keidar. Space bounds for reliable storage: Fundamental limits of coding. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC '16, pages 249–258, New York, NY, USA, 2016. ACM.

[26] Zhiying Wang and Viveck R. Cadambe. On multi-version coding for distributed storage. In *Communication, Control, and Computing (Allerton), 2014 52nd Annual Allerton Conference on*, pages 569–575. IEEE, 2014.

[27] Yuanhao Wei. Space complexity of implementing large shared registers. *arXiv preprint arXiv:1808.00481*, 2018.

תקציר

בשנים האחרונות ניכרת גדילה אקספוננציאלית בביקוש לנפח אחסון של מידע, ובכך פתרונות אחסון עבור נתוני עתק (Big Data) הפכו לנחוצים מתמיד. גישה רווחת לפתרון צורך זה הינה מערכות של אחסון מבוזר. במערכות שכאלו, מידע לרוב מאוחסן על פני צמתים רבים אשר אליהם ניגשים לקוחות באופן אסינכרוני. מערכות אחסון מסוג זה ואשר תומכות בכתיבת ערכים ובקריאתם נקראות ״רגיסטרים״ על פי רוב. דרישה שכיחה מרגיסטרים הינה חוסר-המתנה (Wait-Freedom), שמשמעותה כי, בהינתן די פעולות, כל לקוח שמבצע תהליך כתיבה או קריאה לרגיסטר יסיים אותו בסופו של דבר. בנוסף, הואיל ולקוחות ניגשים לרגיסטר ומבצעים בו פעולות בו-זמנית, נדרש להגדיר מהי ההתנהגות הרצויה מהמערכת כאשר מבוצעות בה מספר פעולות במקביל. הגדרה רווחת שאנו מתמקדים בה במחקר זה נקראת רגולריות (Regularity), ועיקרה הוא שערך שנקרא מהרגיסטר נדרש להיות הערך האחרון שנכתב לרגיסטר לפני תחילת הקריאה ממנו, או, לחילופין, ערך כלשהו שנכתב לרגיסטר בזמן שהקריאה התבצעה.

קיימים בספרות אלגוריתמים רבים המממשים רגיסטרים רגולריים וחסרי-המתנה: רגיסטרים חסינים בפני תקלות ביזנטיות (Byzantine fault-tolerant registers) הינם רגיסטרים שמתמודדים עם האפשרות שחלק מן הצמתים עלולים להיות זדוניים. מערכות מסוג זה פותרות את הבעיה על ידי שכפול מידע על פני צמתים שונים. אלגוריתמים אחרים הינם פתרונות של אחסון מקודד (Coded storage), המיועדים להקל את סיבוכיות המקום הגבוהה שעולה משכפול מידע, באמצעות אחסון של מילות קוד בצמתים, במקום שמירת ערכים שלמים בכל הצמתים. קיימים סוגי רגיסטרים נוספים שמטרתם שמירת ערכים מתחום ערכים גדול, כאשר הצמתים עצמם הינם רגיסטרים בעלי קיבולת שמוגבלת לערכים מתחום קטן.

במחקר זה אנו מצביעים על דמיון מפתיע בין מערכות אחסון חסין בפני תקלות ביזנטיות (ללא אימות של הלקוחות ואותנטיקציה של המידע), לבין מערכות אחסון מקודד ולבין אמולציות מסוימות של רגיסטרים משותפים אשר ממומשות על ידי רגיסטרים קטנים. תכונה משותפת לפתרונות הללו היא חוסר היכולת של קריאה להחזיר ערך בבטחה לאחר גישה אטומית יחידה לצמת משותף, אלא נדרשות מספר גישות לצמתים שונים על מנת להבטיח החזרת ערך חוקי. אנו קוראים למערכות מסוג זה בהכללה מערכות *אחסון חלקי* (Disintegrated Storage).

ניתן לסווג מערכות אחסון חלקי על פי סוג המידע הנשמר בכל צומת. אם לקוח קורא מידע מצומת ומסוגל לשייך אותו למידע שנוצר בכתיבה אחרת (כמו במקרה שבצומת נשמר עותק של הערך), אנו נאמר שזו מערכת אחסון חלקי בעלת *ערך משותף*. אחרת, אם לקוח לא מסוגל לזהות ששתי יחידות מידע משויכות לאותו ערך, אזי נקרא למערכות אלו מערכות אחסון חלקי בעלות *כתיבה משותפת*. מאפיין נוסף של מערכות אחסון חלקי הינו סוג הלקוחות הקוראים של המערכת: לקוח קורא יכול להיות *בלתי-נראה*, קרי הנוכחות שלהם אינה ידועה ללקוח הכותב. לחילופין, הלקוח הקורא יכול להיות *נראה*, ואז הלקוח הכותב מודע לקיומם.

קריטריון חשוב של מערכות מחשב בכלל ושל פתרונות אחסון בפרט הינו סיבוכיות המקום של המערכת, קרי כמה זיכרון ו/או שטח אחסון הפתרון מצריך. בעבודה זו אנו מראים חסמים תחתונים אחודים עבור סיבוכיות המקום של מערכות אסינכרוניות, רגולריות, חסרות-המתנה ובעלות אחסון חלקי. אנו מוכיחים כי במקרה שהלקוחות הקוראים הינם בלתי-נראים, סיבוכיות המקום של מערכות כאלו הינה אקספוננציאלית בגודל של הערכים הנכתבים, באופן אינהרנטי. במקרה שהלקוחות הקוראים נראים ללקוח הכותב, אנו מראים שסיבוכיות המקום היא לכל הפחות לינארית במספר הלקוחות הקוראים. החסמים הללו הינם הדוקים באופן אסימפטוטי לסיבוכיות המקום של אלגוריתמים קיימים, ובכך אנו מצדיקים את עלויות האחסון הגבוהות של אלגוריתמים אלו.

חסמים אחודים עבור אחסון חלקי

חיבור על מחקר לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעים בהנדסת חשמל

אלון ברגר

הוגש לסנט הטכניון – מכון טכנולוגי לישראל
שבט התשע״ט חיפה ינואר 2019

חסמים אחודים עבור אחסון חלקי

אלון ברגר