Information-Theoretic Lower Bounds on the Storage Cost of Shared Memory Emulation

Viveck R. Cadambe
EE Department
Pennsylvania State University
University Park, PA, USA
viveck@engr.psu.edu

Zhiying Wang
CPCC Center,
University of California, Irvine
Irvine, CA, USA
zhiying@uci.edu

Nancy Lynch
Department of EECS
MIT
Cambridge, MA, USA
lynch@csail.mit.edu

ABSTRACT

The focus of this paper is to understand storage costs of emulating an atomic shared memory over an asynchronous, distributed message passing system. Previous literature has developed several shared memory emulation algorithms based on replication and erasure coding techniques, and analyzed the storage costs of the proposed algorithms. In this paper, we present the first known information-theoretic lower bounds on the storage costs incurred by shared memory emulation algorithms. Our storage cost lower bounds are universally applicable, that is, we make no assumption on the structure of the algorithm or the method of encoding the data.

We consider an arbitrary algorithm A that implements an atomic multi-writer-single-reader (MWSR) shared memory variable whose values come from a finite set $\mathcal{V}$ over a system of N servers connected by point-to-point asynchronous links. We require that in every fair execution of algorithm A where the number of server failures is smaller than a parameter f, every operation invoked at a non-failing client terminates. We define the storage cost of a server in algorithm A as the logarithm (to base 2) of the number of states it can take on; the total storage cost of algorithm A is the sum of the storage cost of all servers. Previously, it was known that the storage cost cannot be smaller than $\frac{N}{N-f}\log_2|\mathcal{V}|$. We develop three new lower bounds on the storage cost of algorithm A.

- In our first lower bound, we show that if algorithm A does not use server gossip, then the total storage cost is lower bounded by $2\frac{N}{N-f+1}\log_2|\mathcal{V}| - o(\log_2|\mathcal{V}|)$.
- In our second lower bound we show that the total storage cost is at least $2\frac{N}{N-f+2}\log_2|\mathcal{V}| - o(\log_2|\mathcal{V}|)$ even if the algorithm uses server gossip.
- In our third lower bound, we consider algorithms where the write protocol sends information about the value in at most one phase. For such algorithms, we show that the total storage cost is at least $\nu^*\frac{N}{N-f+\nu^*-1}\log_2(|\mathcal{V}|) - o(\log_2(|\mathcal{V}|)$, where ν^* is the minimum of $f+1$ and the number of active write operations of an execution.

Our first and second lower bounds are approximately twice as strong as the previously known bound of $\frac{N}{N-f}\log_2|\mathcal{V}|$. Furthermore, our first two lower bounds apply even for regular, single-writer-single-reader (SWSR) shared memory emulation algorithms. Our third lower bound is much larger than our first and second lower bounds, although it is applicable to a smaller class of algorithms where the write protocol has certain restrictions. In particular, our third bound is comparable to the storage cost achieved by most shared memory emulation algorithms in the literature, which naturally fall under the class of algorithms studied. Our proof ideas are inspired by recent results in coding theory.

PODC'16, July 25-28, 2016, Chicago, IL, USA
© 2016 ACM. ISBN 978-1-4503-3964-3/16/07...$15.00
DOI: http://dx.doi.org/10.1145/2933057.2933118

CCS Concepts

•Mathematics of computing → Information theory; Coding theory; •Information systems → Storage management; •Theory of computation → Distributed algorithms;

Keywords

Atomic Memory; Distributed Storage; Shared Memory Emulation; Storage Cost

1. INTRODUCTION

The emulation of a consistent, fault-tolerant read-write shared memory in a distributed, asynchronous, storage system has been an active area of research in distributed computing theory. In their celebrated paper [3], Attiya, Bar-Noy, and Dolev devised a fault-tolerant algorithm for emulating a shared memory that achieves atomic consistency (linearizability) [17, 16]. Consider a distributed system with server nodes, write client and read client nodes, all of which are connected by point-to-point asynchronous links. The ideas of [3] can be used to design server, write and read protocols that implement an atomic shared memory even if the write and read operations are invoked concurrently with the following guarantee: every read or write operation invoked at a non-failing client terminates so long as the set of servers that fail is restricted to a minority. The algorithm of [3] used a replication-based storage scheme at the servers to attain fault tolerance. Following [3], several papers [15, 13, 1, 4, 21, 12, 5, 2] have developed algorithms that use *erasure coding* instead of replication for fault tolerance, with the goal of improving upon the storage efficiency of [3].

In erasure coding[1] which is studied in classical coding theory, each server stores a function of the value called a codeword symbol. A decoder that is able to access a sufficient number of codeword symbols recovers the value. The number of bits used to represent a codeword symbol is typically much smaller than the number of bits used to represent the value. As a consequence, erasure coding is well known to lead to smaller storage costs as compared to replication in the classical coding-theoretic set-up (See, for example, [11, 19, 8]). Here, we aim to understand storage costs of shared memory emulation, where in contrast with the classical coding-theoretic setup, multiple versions of the data object are to be stored in a consistent manner.

When erasure coding is used for shared memory emulation, new challenges arise. Since, in erasure coding, each server stores a codeword symbol and not the entire value, a read operation has to obtain a sufficient number of codeword symbols to decode the value being stored. When a write operation begins to write a new version of the data object, the old version cannot be deleted from the servers until a sufficient number of codeword symbols corresponding to the new version have been propagated to the servers. As a consequence, servers have to store codeword symbols corresponding to multiple versions of the data object to ensure that a reader can decode an atomically consistent version. Previous erasure coding based shared memory emulation algorithms [15, 13, 1, 4, 21, 12, 5] have noted that the number of versions to be stored at a server can be large if there are a large number of ongoing or failed write operations whose codeword symbols have not been propagated sufficiently. Because servers store codeword symbols corresponding to multiple versions, the storage cost of using erasure coding can be large, even if the number of bits in each codeword symbol is small compared to the number of bits used to represent the value.

Despite the vast amount of literature in the study of storage costs of shared memory emulation, some compelling and fundamental questions remain unanswered. Since a server can store an arbitrary function of all the symbols it receives, can we develop a sophisticated storage strategy that somehow compresses multiple versions at the servers and thereby results in smaller storage costs? If we add multiple phases to read and write protocols or include other algorithmic novelties, can we reduce the storage cost of shared memory emulation? In our paper, we obtain insights into these questions by developing novel impossibility results that lower bound the storage cost of an arbitrary atomic shared memory emulation algorithm.

2. SUMMARY OF RESULTS AND COMPARISONS WITH RELATED WORK

In this section, we first summarize the shared memory emulation and the classical coding theory set-ups. We then describe our storage cost lower bounds in Theorems 4.1 and 5.1. Then, we describe our storage cost lower bound of Theorem 6.5. Finally we compare our results to previously derived storage cost lower bounds.

[1]Server failures are modeled as erasures of codeword symbols; hence the term, erasure coding.

2.1 Set up

Shared Memory Emulation Set-up: We consider an arbitrary algorithm A that implements, over a network of N servers connected by point-to-point asynchronous links, an atomic multi-writer-single-reader (MWSR) shared memory variable whose values come from a finite set $\mathcal{V}$. The algorithm A is required to ensure that all operations terminate so long as the number of server failures is no larger than a parameter f. The storage cost of a server in algorithm A is measured as the logarithm of the number of possible states of the server, and the storage cost of algorithm A is the total storage cost over all the servers.

Classical Coding Theory Set-up: Consider a system with N servers, where a single version of a data object whose values come from a finite set $\mathcal{V}$ is to be stored. The value of the data object must be recoverable, so long as the number of server failures is no larger than a parameter f. The classical *Singleton bound* [18, 20] in coding theory implies that the total storage cost is at least $\frac{N \log_2 |\mathcal{V}|}{N-f}$ bits[2]. The lower bound of $\frac{N \log_2 |\mathcal{V}|}{N-f}$ on the storage cost is known to be tight in the classical coding-theoretic set-up for large values of $|\mathcal{V}|$ [20, 18].

The power of erasure coding is transparent when we want to design a storage system that tolerates failures of f server nodes and the number of server nodes N can be chosen freely. If we use replication, every server stores $\log_2 |\mathcal{V}|$ bits. Since we need at least $f+1$ servers to tolerate f server failures, the total storage cost of the system is at least $(f+1)\log_2 |\mathcal{V}|$ bits. In contrast, if we use erasure coding, the total storage cost of the system is $\frac{N \log_2 |\mathcal{V}|}{N-f}$, which approaches $\log_2 |\mathcal{V}|$ as N increases. If N is sufficiently large, the storage cost of replication is approximately $f+1$ times the storage cost of erasure coding.

2.2 Motivation and Summary - Theorems 4.1 and 5.1

Motivation: The classical coding-theoretic model does not model clients or channels, and therefore differs significantly from the shared memory emulation model. However, the storage cost lower bound of $\frac{\log_2 |\mathcal{V}|}{N-f}$ described by the Singleton bound is, in fact, applicable in the context of shared memory emulation as well. We provide the first formal proof of the lower bound in the extended paper [7]; in particular, we show that for any SWSR regular shared memory emulation algorithm that implements a read write data object whose values come from a set $\mathcal{V}$, the total storage cost is at least $\frac{N \log_2 |\mathcal{V}|}{N-f}$. The natural lower bound of $\frac{N \log_2 |\mathcal{V}|}{N-f}$ inspires the following question.

Question 1: Does there exist an atomic shared memory emulation algorithm whose storage cost is equal to $\frac{N}{N-f}\log_2 |\mathcal{V}|$?

Summary of Theorems 4.1 and 5.1: In this paper, we answer the above question in the negative by proving storage cost lower bounds that are stronger than $\frac{N}{N-f}\log_2 |\mathcal{V}|$. In Theorems 4.1 and 5.1 we show that the total-storage cost of single-writer-single-reader (SWSR) *regular* shared memory emulation algorithm is at least $\frac{2N}{N-f+2}\log_2 |\mathcal{V}| -$

[2]For the sake of the discussion here, we assume that $|\mathcal{V}|$ is a power of 2. We refer the reader to [18, 6] for more details about erasure coding.

$o(\log_2 |\mathcal{V}|)$. In particular, if f is fixed and N is chosen freely, the total-storage cost lower bound of Theorems 4.1 and 5.1 approach $\frac{2N}{N-f} \log_2 |\mathcal{V}| - o(|\log_2 |\mathcal{V}|)$ as N increases; therefore the bounds of Theorems 4.1 and 5.1 are twice as large as the previously known lower bound. Recall that regularity [17] is a weaker consistency model as compared with atomicity. Since Theorems 4.1 and 5.1 apply for regular SWSR shared memory emulation algorithm, they automatically apply for atomic MWSR shared memory emulation algorithms. Theorem 4.1 describe a storage cost lower bound for algorithms which do not use server gossip. Theorem 5.1 describes a lower bound for *any* shared memory emulation algorithm, including algorithms that use server gossip. Our storage cost lower bounds are universal in nature, that is, we make no assumption on the structure of the protocols or the method of data storage. Because we answer Question 1 in the negative, an important implication is that there is an unavoidable price, in terms of storage cost, to ensure regularity in a shared memory emulation system. We next discuss the tightness of Theorems 4.1 and 5.1 in the context of previously derived storage cost upper bounds.

2.3 Motivation and Summary - Theorem 6.5

In the sequel, we define the number of active write operations at point P of an execution as the number of write operations which have begun before the point P but not yet terminated or failed at point P. The number of active write operations of an execution is the supremum, over all points of the execution, of the number of active write operations at the points of the execution.

Motivation: There is a growing body of literature related to erasure coding based shared memory emulation algorithms [15, 13, 1, 4, 21, 12, 6, 5, 2, 22]. These algorithms differ in their structure, liveness conditions on operation termination, and their communication costs. A common insight that applies to all the algorithms of [15, 13, 1, 4, 21, 12, 6, 5, 2, 22] is that, among the class of all executions with at most ν active write operations, the worst case storage cost of implementing an atomic shared memory object whose values come from a finite set $\mathcal{V}$ is *at least* $\nu \frac{N \log_2 |\mathcal{V}|}{N-f}$. In fact, references [13, 5, 2, 4] conduct a formal analysis of the incurred storage cost and show that the storage cost incurred[3] is approximately $\nu \frac{N \log_2 |\mathcal{V}|}{N-f}$. While the prior works highlight the benefit of erasure coding when the number of active writes is small, the storage cost benefits of erasure coding vanish as the number of active writes increases. In particular, for a sufficiently large value of ν, erasure coding based algorithms can even have a higher storage cost as compared to replication based algorithms [3, 14], which incur a storage cost of $\Theta(f) \log_2 |\mathcal{V}|$ irrespective of the number of active writes.

In contrast with the storage cost upper bounds in literature, our lower bounds of Theorem 4.1 and 5.1 do not depend on the number of active writes. Furthermore, if f is proportional to N, then storage cost lower bounds of Theorem 4.1 and 5.1 are both $o(f) \log_2 |\mathcal{V}| + o(\log_2 |\mathcal{V}|)$. Prior literature in conjunction with our results of Theorems 4.1 and 5.1 motivates the following question:

[3]There are subtle differences in the storage cost incurred by the algorithms of [6, 5, 2, 4, 13]. Nonetheless, $\nu \frac{N}{N-f} \log_2 |\mathcal{V}|$ is a lower bound on the cost incurred by these algorithms in the *worst case*, among executions where the number of active writes is bounded by ν.

Question 2: Can we develop an algorithm whose storage cost, when f is proportional to N, is as small as $o(f) \log_2 |\mathcal{V}|$ and does not grow with the number of active writes?

Summary of Theorem 6.5: We provide partial answer to Question 2 in our lower bound presented in Theorem 6.5. The lower bound states that the answer to Question 2 is negative, if the write protocol of the algorithm satisfies certain technical conditions described in Section 6. Informally speaking, the technical conditions in Section 6 imply that the write operation is executed in phases, and a message containing information about the value is sent to the servers in at most one phase per write operation. For any atomic MWSR algorithm that ensures that all operations terminate in every execution where the active number of write operations is at most ν and the number of server failures is at most f, Theorem 6.5 shows that if the write protocol satisfies the conditions stated in Section 6, then the storage cost cannot be smaller than $\nu^* \frac{\log_2 |\mathcal{V}|}{N-f+\nu^*-1} - o(\log |\mathcal{V}|)$, where ν^* is the minimum of $\{f+1, \nu\}$.

Theorem 6.5 is interesting from a conceptual viewpoint since it captures the dependence of the storage cost on the degree of concurrency that has been noticed in the upper bounds of [13, 6, 5, 2, 4]. In particular, the bound of Theorem 6.5 can be much larger than the bounds of Theorems 4.1 and 5.1, if the parameters ν and f are sufficiently large. If the number of active write operations exceeds $(f+1)$, then our storage cost lower bound of Theorem 6.5, which equals $(f+1) \log_2 |\mathcal{V}| - o(\log_2 |\mathcal{V}|)$, implies that replication based algorithms are approximately optimal in the class of algorithms described in the theorem.

The class of algorithms that satisfy the conditions stated in Section 6 include a majority of the algorithms in literature [13, 1, 4, 21, 12, 5]. We refer the reader to Section 6 for a more detailed justification. Theorem 6.5, in the stated form, does not apply to a few algorithms [2, 15] because these protocols send messages related to the value of the write operation in two phases; one phase is used to send a hash of the value for client verification purposes, and a second phase is used to send codeword symbols corresponding to the value.

We provide sketches of proofs in this paper. Full proofs of our results are provided in the extended version of our paper [7].

2.4 Comparison with Prior Storage Cost Lower Bounds

We refer the reader to the extended version of the paper [7] for a comparison with some related works [9, 10, 14, 23, 24]. Here we discuss reference [22], which describes interesting, non-trivial lower bounds on shared memory emulation algorithms where the server and client storage schemes satisfy certain restrictions. Reference [22] assumed that every bit stored in the system is associated uniquely with a write operation, and showed that under such a storage scheme, the worst case total storage cost of the system is at least $\Omega(\min(f, \nu) \log |\mathcal{V}|)$. The implication of [22] is that in the worst case, if the degree of concurrency is infinite and the server storage scheme is restricted in a particular manner, then the replication based algorithms of [3, 14] are approximately optimal.

The assumption of [22] that every bit stored is associated with a unique write operation is restrictive and does not apply to all possible storage methods. To see this, consider a scenario where $\mathcal{V}$ is a finite field. Let $v_1, v_2 \in \mathcal{V}$ be values corresponding to two different write operations. Suppose in some algorithm A, at some point of an execution, a server stores $v_1 + v_2$, where $+$ denotes the addition operator over the field. Then a bit stored by the server cannot be uniquely associated with any of the write operations in an unambiguous way. Therefore, the proof technique and the result of [22] fails to provide any insight on the storage cost of the algorithm A. Put differently, the storage method of [22] does not allow for server storage techniques that potentially compress the values of different versions together (See extended paper [7] for more details). In contrast, the results of Theorems 4.1, 5.1 are universal and would automatically apply to algorithm A. The result of Theorem 6.5 does not impose any structure on the storage method, and could also apply to algorithm A if its write protocol satisfies the appropriate restrictions. In our concluding section, Section 7, we provide a summary of the state of the art based on the main results of our paper and of [22].

3. PRELIMINARIES

We study the emulation of a shared atomic memory in an asynchronous message passing network. Our setting consists of a set of N server nodes and a possibly infinite set of client nodes. Without loss of generality, we let the set of server nodes be $\{1, 2, \ldots, N\}$. We denote the set of client nodes as $\mathcal{C}$. We assume a multi-writer single-reader (MWSR) setting, that is, we assume that $\mathcal{C}$ has a single read client; the remaining clients in $\mathcal{C}$ are write clients, denoted by $\mathcal{C}_w$. If $\mathcal{C}$ has exactly one write client and one read client, we refer to the setting as a single-writer-single-reader (SWSR) setting. Each client node is connected to all the server nodes, and the servers are connected to each other via point-to-point reliable asynchronous channels. Read clients receive read requests (invocations) from some external source and respond with object values. Write clients receive write-requests and respond with acknowledgements. Every new invocation at a client waits for a response of a preceding invocation at the same client. The goal of a shared memory emulation algorithm A studied in this paper is to design the client and server protocols that implement a read-write register of a data object which can take values from a finite set $\mathcal{V}$, with the following safety and liveness properties.

Safety Properties: The algorithms must emulate a SWSR regular registers [17]. Informally a regular shared memory object requires that every read operation returns either the value written by the latest write operation that terminates before the invocation of the read operation, or the value of a write operation that overlaps with the read operation. In Section 6 we further consider multi-writer single-reader algorithms, and we require the algorithm to be atomic [17]. Informally, in an atomic algorithm, the observed external behavior of every execution looks like the execution of a serial variable type. Recall that SWSR execution of an atomic shared memory emulation is also regular, so our lower bounds for regular algorithms in Theorems 4.1 and 5.1 also apply to atomic emulation algorithms.

Liveness Properties: An operation of a non-failed client must terminate in a fair execution, so long as the number of server failures in the execution is bounded by a parameter f. We consider algorithms with weaker liveness properties as well in Section 6, where we ensure termination of operations in executions if the number of *active* write operations is bounded. A formal statement of the weaker liveness properties is provided in Section 6.

Storage Cost Definition: Informally speaking, the *storage cost* of an algorithm is the total number of bits stored by the servers. In general, for an algorithm where the state of server node $i \in \{1, 2, \ldots, N\}$ can take values from a set $\mathcal{S}_i$, we define the storage cost of the server to be equal to $\log_2 |\mathcal{S}_i|$ bits. The *total-storage cost* of the algorithm is defined to be $TotalStorage(A) = \sum_{i=1}^{N} \log_2 |\mathcal{S}_i|$ bits.

4. STORAGE COST LOWER BOUND FOR ALGORITHMS WITHOUT GOSSIP

Our main result of this section is a storage cost lower bound, assuming that servers do not gossip. Specifically, in this section, we assume that every message is sent from a client to a server, or from a server to a client. We state Theorem 4.1, provide an informal description of the proof, followed by a more detailed description.

THEOREM 4.1. *Let A be a single-writer-single-reader shared memory emulation algorithm that implements a regular read-write object whose values come from a finite set $\mathcal{V}$. Suppose that in A, every message is sent from a server to a client, or from a client to a server. Suppose that the algorithm A satisfies the following liveness property: In a fair execution of A, if the number of server failures is no bigger than f, $f \geq 2$, then every operation invoked at a non-failing client terminates. Then, for every subset $\mathcal{N} \subset \{1, 2, \ldots, N\}$ where $|\mathcal{N}| = N - f$,*

$$\sum_{n \in \mathcal{N}} \log_2 |\mathcal{S}_i| + \max_{n \in \mathcal{N}} \log_2 |\mathcal{S}_i| \geq \log_2 |\mathcal{V}| + \log_2(|\mathcal{V}| - 1)) - \log_2(N - f). \tag{1}$$

$$TotalStorage(A) \geq \frac{N(\log_2 |\mathcal{V}| + \log_2(|\mathcal{V}| - 1) - \log_2(N - f))}{N - f + 1}. \tag{2}$$

Informal Proof Sketch: We show in the extended paper [7] that inequality (2) is a simple consequence of (1). Here we prove (1). Informally speaking, our lower bound argument is as follows. For every subset $\mathcal{N} \subset \{1, 2, \ldots, N\}$ where $|\mathcal{N}| = N - f$, we construct an execution where the servers in $\{1, 2, \ldots, N\} - \mathcal{N}$ fail at the beginning of the execution. The execution has two write operations for values v_1 and v_2, where $v_1 \neq v_2 \in \mathcal{V}$. The second write operation which writes value v_2 begins after the termination of the first write operation, which has value v_1.

In this execution, after the point of termination of the first write, a reader can return v_1 because of regularity. Similarly, after the termination of the second write operation, a reader can return v_2. Therefore, the value v_1 is returnable from the servers at a point before the invocation of the second write operation and v_2 is returnable from the servers at a point after the completion of second write operation. Furthermore, at every point in the interval of the second write operation, at least one of v_1 or v_2 are returnable. This implies that, in the interval of the second write operation, there are two consecutive points P and P' such that v_1 must be returnable from the non-failing servers at point P and v_2 must be returnable from the non-failing servers at point P'. Since (v_1, v_2) can be any ordered pair of distinct values from $\mathcal{V}$, there must be a one-to-one mapping between the set $\{(v_1, v_2) : (v_1, v_2) \in \mathcal{V} \times \mathcal{V}, v_1 \neq v_2\}$ and the set of possible configurations of server states at points P and P'. This implies that the number of possible server states at points P and P' is at least $(|\mathcal{V}|)(|\mathcal{V}|-1)$. Since P and P' are consecutive, at most one non-failing server changes its state between these two points. At least $N - f - 1$ non-failing servers have the same state at point P as at point P'. We use this fact to show that the number of elements in the set of possible server states at two consecutive points is at most $\prod_{n\in\mathcal{N}} |\mathcal{S}_i| \times \max_{n\in\mathcal{N}} |\mathcal{S}_n| \times (N-f)$. Therefore, we get $\prod_{n\in\mathcal{N}} |\mathcal{S}_i| \times \max_{n\in\mathcal{N}} |\mathcal{S}_n| \times (N-f) \geq (|\mathcal{V}|)(|\mathcal{V}|-1)$, which implies the lower bound.

4.1 Formal Proof of Theorem 4.1

Consider an arbitrary subset $\mathcal{N} \subset \{1, 2, \ldots, N\}$ such that $|\mathcal{N}| = N - f$. We construct $|\mathcal{V}| \times (|\mathcal{V}|-1)$ executions of the algorithm A. In particular, for every tuple $(v_1, v_2) \in \mathcal{V} \times \mathcal{V}$ where $v_1 \neq v_2$, we create an execution $\alpha^{(v_1,v_2)}$ of algorithm A. In our proof, we first describe execution $\alpha^{(v_1,v_2)}$ in Section 4.1.1. Then we present some properties of execution $\alpha^{(v_1,v_2)}$ in Section 4.1.2. We use the results of Section 4.1.2 to prove Theorem 4.1 in Section 4.1.3.

4.1.1 Execution $\alpha^{(v_1,v_2)}$

In execution $\alpha^{(v_1,v_2)}$ the reader and the channels from and to the reader do not perform any actions. Among the set of write clients $\mathcal{C}_w$, only one write client takes actions. The f servers in $\{1, 2, \ldots, N\} - \mathcal{N}$ fail at the beginning of the execution. No further server failures occur in the execution. A write π_1 is invoked at a write client with value v_1. All the components of the system except the reader, and the channels from and to the reader, take turns in a fair manner until the completion of π_1. Immediately after the termination of π_1, a write operation π_2 with value v_2 begins. All the components of the system except the reader and the channels from and to the reader take turns in a fair manner until the completion of π_2. The execution $\alpha^{(v_1,v_2)}$ ends after the termination of π_2.

4.1.2 Properties of Execution $\alpha^{(v_1,v_2)}$

Let $P_0^{(v_1,v_2)}, P_1^{(v_1,v_1)}, P_2^{(v_1,v_2)}, \ldots, P_M^{(v_1,v_2)}$ be the adjacent points (or points after successive steps) of the constructed execution $\alpha^{(v_1,v_2)}$, where $P_0^{(v_1,v_2)}$ is an arbitrary point after the termination of π_1 and before the invocation of π_2 and $P_M^{(v_1,v_2)}$ is an arbitrary point after the point of termination of π_2, and M is a positive integer. By the liveness property, we can choose M to be finite. For $i \in \{0, 1, 2, \ldots, M\}$, we denote by $\alpha_i^{(v_1,v_2)}$, the execution between the initial point of $\alpha^{(v_1,v_2)}$ and point $P_i^{(v_1,v_2)}$. We next define the notions of 1-valent and 2-valent points.

Definition 4.2 (k-valent, $k \in \{1, 2\}$). *For $i \in \{0, 1, 2, \ldots, M\}$, a point $P_i^{(v_1,v_2)}$ in execution $\alpha_i^{(v_1,v_2)}$ is said to be k-valent if we can extend $\alpha_i^{(v_1,v_2)}$ to an execution β as follows: After $P_i^{(v_1,v_2)}$ in β, all the messages from and to the writer are delayed indefinitely. A read operation starts at point $P_i^{(v_1,v_2)}$ and all the components, except the writer and the channels from and to the writer, perform actions until the read operation terminates. The read operation returns v_k.*

It should be noted that a point of an execution can be both 1-valent and 2-valent.

Lemma 4.3. *For $i \in \{0, 1, 2, \ldots, M\}$, a point $P_i^{(v_1,v_2)}$ that is not 1-valent is 2-valent.*

Lemma 4.3 is a natural consequence of Lemma 4.4 which informally states that a reader that begins after the termination of the write π_1 should return v_1 or v_2.

Lemma 4.4. *Consider an execution β which is an extension of $\alpha_i^{(v_1,v_2)}$. In β, after point $P_i^{(v_1,v_2)}$, the writer stops taking steps and all messages from and to the writer are delayed indefinitely. A read operation begins at some point after point $P_i^{(v_1,v_2)}$ and terminates in β. Then, the read operation returns either v_1 or v_2.*

The lemma is a natural consequence of the regularity of algorithm A.

Lemma 4.5. *There exists some integer $i \in \{0, 2, \ldots, M-1\}$ such that $P_i^{(v_1,v_2)}$ is 1-valent and $P_{i+1}^{(v_1,v_2)}$ is not 1-valent.*

We can show by regularity that (i) Point $P_0^{(v_1,v_2)}$ is 1-valent, and (ii) point $P_M^{(v_1,v_2)}$ is not 1-valent. Among all the numbers in $\{0, 1, 2, \ldots, M\}$, let i denote the largest number such that $P_i^{(v_1,v_2)}$ is 1-valent. If (i) is true, we note that the number i exists. If (ii) is true, then $i < M$. Since i is the largest number such that $P_i^{(v_1,v_2)}$ is 1-valent, we infer that $P_i^{(v_1,v_2)}$ is 1-valent, but $P_{i+1}^{(v_1,v_2)}$ is not 1-valent. The point $P_i^{(v_1,v_2)}$ therefore satisfies the statement of the lemma. Next, we present the definition of a *pair of critical points* of execution $\alpha^{(v_1,v_2)}$.

Definition 4.6 (Critical points). *Let Q_1, Q_2 be two points in execution $\alpha^{(v_1,v_2)}$. The pair of points (Q_1, Q_2) is defined to be a pair of critical points if there exists a number i in $\{0, 2, \ldots, M-1\}$ such that $Q_1 = P_i^{(v_1,v_2)}$, $Q_2 = P_{i+1}^{(v_1,v_2)}$, Q_1 is 1-valent, and Q_2 is not 1-valent.*

Lemma 4.5 implies that every execution $\alpha^{(v_1,v_2)}$ has at least one pair of critical points. Lemma 4.3 implies that if (Q_1, Q_2) is a pair of critical points in $\alpha^{(v_1,v_2)}$, then point Q_2 is 2-valent in $\alpha^{(v_1,v_2)}$. We need the following lemma before proceeding to prove Theorem 4.1.

Lemma 4.7. *Let (Q_1, Q_2) be a pair of critical points of execution $\alpha^{(v_1,v_2)}$. Then, the reader, and the channels between the reader and the servers, are all in the same state at point Q_2 as at point Q_1. Also, there is at most one non-failing server s such that its state at Q_1 is different from its state at Q_2.*

4.1.3 Proof of Theorem 4.1

For given (v_1, v_2), fix $(Q_1^{(v_1,v_2)}, Q_2^{(v_1,v_2)})$ to be a pair of critical points in execution $\alpha^{(v_1,v_2)}$. From Lemma 4.7, we note that there is at most one non-failing server that changes state between $Q_1^{(v_1,v_2)}$ and $Q_2^{(v_1,v_2)}$. Let s denote the server which changes state between points $Q_1^{(v_1,v_2)}$ and $Q_2^{(v_1,v_2)}$, if there is one; if not, let s denote an arbitrary non-failing server. If $s' \in \mathcal{N}, s' \neq s$, the state of the server s' is the same at points $Q_1^{(v_1,v_2)}$ and $Q_2^{(v_1,v_2)}$.

Let $\vec{S}^{(v_1,v_2)}$ be an element of $\prod_{n\in\mathcal{N}} \mathcal{S}_n \times \mathcal{N} \times \cup_{n\in\mathcal{N}} \mathcal{S}_n$ as follows. The first $N-f$ components of $\vec{S}^{(v_1,v_2)}$ denote the states of the $N-f$ servers in $\mathcal{N}$ at point $Q_1^{(v_1,v_2)}$. The $(N-f+1)$st component of $\vec{S}^{(v_1,v_2)}$ denote the server index s, and the $(N-f+2)$nd component is the state of server s at point $Q_2^{(v_1,v_2)}$. Note that the number of elements in the set $\bigcup_{(v_1,v_2)\in\mathcal{V}\times\mathcal{V}, v_1\neq v_2} \{\vec{S}^{(v_1,v_2)}\}$ is at most $\prod_{i\in\mathcal{N}} |\mathcal{S}_i| \times (N-f) \times \max_{i\in\mathcal{N}} |\mathcal{S}_i|$.

To prove Theorem 4.1, we show that, if (v_1,v_2) and (v_1',v_2') are two *distinct* elements of the set $\{(x,y) : (x,y) \in \mathcal{V}\times\mathcal{V}, x\neq y\}$, then $\vec{S}^{(v_1,v_2)} \neq \vec{S}^{(v_1',v_2')}$. If we show this, then it implies that the number of elements in the set

$$\bigcup_{(v_1,v_2)\in\mathcal{V}\times\mathcal{V}, v_1\neq v_2} \{\vec{S}^{(v_1,v_2)}\}$$

is at least equal to the number of elements in the set $\{(x,y) : (x,y) \in \mathcal{V}\times\mathcal{V}, x\neq y\}$, which is equal to $(|\mathcal{V}|)\times(|\mathcal{V}|-1)$. This implies the following inequality:

$$\prod_{n\in\mathcal{N}} |\mathcal{S}_n| \times (N-f) \times \max_{n\in\mathcal{N}} |\mathcal{S}_n| \geq (|\mathcal{V}|)\times(|\mathcal{V}|-1)$$

which implies the theorem. Suppose, for contradiction, there are two distinct tuples (v_1,v_2) and (v_1',v_2') in $\{(x,y) : (x,y) \in \mathcal{V}\times\mathcal{V}, x\neq y\}$ and $\vec{S}^{(v_1,v_2)} = \vec{S}^{(v_1',v_2')}$.

Let i denote an integer such that $Q_1^{(v_1,v_2)}$ is the point $P_i^{(v_1,v_2)}$ in $\alpha^{(v_1,v_2)}$. Because the point $Q_1^{(v_1,v_2)}$ is 1-valent, there is an execution $\beta_1^{(v_1,v_2)}$ that extends $\alpha_i^{(v_1,v_2)}$ such that, after point $Q_1^{(v_1,v_2)}$ in $\beta_1^{(v_1,v_2)}$, all the messages from and to the writer are delayed indefinitely, and a read operation begins and returns v_1. Similarly, because the point $Q_2^{(v_1,v_2)}$ is 2-valent, there is an execution $\beta_2^{(v_1,v_2)}$ that extends $\alpha_{i+1}^{(v_1,v_2)}$ such that, after point $Q_2^{(v_1,v_2)}$, all the messages from and to the writer are delayed indefinitely, and a read operation begins and returns v_2. The following lemma describes a useful property of executions $\beta_1^{(v_1,v_2)}$ and $\beta_2^{(v_1,v_2)}$.

Lemma 4.8. *Let $\vec{S}^{(v_1,v_2)} = \vec{S}^{(v_1',v_2')}$. Consider the composite automaton $\mathcal{A}$ formed by the servers, the reader and the channels between the reader and servers. For $k \in \{1,2\}$, every component of the automaton $\mathcal{A}$ has the same state at point $Q_k^{(v_1,v_2)}$ in $\beta_k^{(v_1,v_2)}$ as at point $Q_k^{(v_1',v_2')}$ in execution $\beta_k^{(v_1',v_2')}$.*

We now use Lemma 4.8 to obtain a contradiction. Because (v_1,v_2) and (v_1',v_2') are distinct ordered pairs, there are only two possibilities: (I) $v_1' \neq v_1, v_1' \neq v_2$, (II) $v_2' \neq v_1, v_2' \neq v_2$, or $v_2' = v_1, v_1' = v_2$, both of which imply that $v_2' \neq v_2$. We handle these possibilities separately.

Case (I): $v_1' \neq v_1, v_1' \neq v_2$. We create an execution γ of the algorithm A which contradicts Lemma 4.4. Let i be an integer such that $Q_1^{(v_1,v_2)} = P_i^{(v_1,v_2)}$ in execution $\alpha^{(v_1,v_2)}$. The execution γ extends execution $\alpha_i^{(v_1,v_2)}$, that is, it follows execution $\alpha^{(v_1,v_2)}$ until point $Q_1^{(v_1,v_2)}$. After point $Q_1^{(v_1,v_2)}$, the writer, and the channels from and to the writers do not perform any actions, After point $Q_1^{(v_1,v_2)}$, the servers, reader and the channels between the servers and reader in γ follow the same steps as the corresponding components in $\beta_1^{(v_1',v_2')}$. Lemma 4.8 implies that γ is an execution of algorithm A. In particular, γ is an extension of $\alpha_i^{(v_1,v_2)}$, where, after point $P_i^{(v_1,v_2)}$, the writer and channels from and to the writer do not perform any actions, a read begins and terminates returning v_1'. However, according to Lemma 4.4, the read operation in γ should return either v_1 or v_2,. Since $v_1 \neq v_1', v_2 \neq v_1'$, this is a contradiction.

Case (II): $v_2' \neq v_2$. We create an execution γ of the algorithm A that leads to a contradiction as follows. The execution γ extends execution $\alpha_{i+1}^{(v_1,v_2)}$, that is, it follows execution $\alpha^{(v_1,v_2)}$ until point $Q_2^{(v_1,v_2)}$. After point $Q_2^{(v_1,v_2)}$, the writer does not take any steps, and the messages from and to the writer are delayed indefinitely. After point $Q_2^{(v_1,v_2)}$, the servers, reader and the channels between the servers and reader in γ follow the same steps as the corresponding components in $\beta_2^{(v_1',v_2')}$. Lemma 4.8 implies that γ is an execution of algorithm A. Since γ follows $\beta_2^{(v_1',v_2')}$ from point $Q_2^{(v_1,v_2)}$, a read operation begins in γ and returns v_2'. However, according to Lemma 4.3 and the fact that $Q_2^{(v_1,v_2)}$ is not 1-valent, $Q_2^{(v_1,v_2)}$ is 2-valent, and the read operation in γ should return v_2. This is a contradiction. □

5. UNIVERSAL STORAGE COST LOWER BOUND

Our main result of this section is a storage cost lower bound that is applicable to any regular SWSR shared memory emulation algorithm, even if it uses server gossip.

Theorem 5.1. *Let A be a single-writer-single-reader shared memory emulation algorithm that implements a regular read-write object whose values come from a finite set $\mathcal{V}$. Suppose that the algorithm A satisfies the following liveness property: In a fair execution of A, if the number of server failures is no bigger than f, then every operation invoked at a non-failing client terminates. Then, for every subset $\mathcal{N} \subset \{1,2,\ldots,N\}, |\mathcal{N}| = N-f$,*

$$\begin{aligned} &2\max_{n\in\mathcal{N}} \log_2 |\mathcal{S}_i| + \sum_{n\in\mathcal{N}} \log_2 |\mathcal{S}_i| \\ &\geq \log_2 |\mathcal{V}| + \log_2 (|\mathcal{V}|-1) - 2\log_2(N-f), \end{aligned} \tag{3}$$

$$\begin{aligned} &TotalStorage(A) \\ &\geq \frac{N(\log_2 |\mathcal{V}| + \log_2 |\mathcal{V}-1| - 2\log_2(N-f))}{N-f+2}. \end{aligned} \tag{4}$$

Informal Proof Sketch: The proof of Theorem 5.1 shares many common elements with the proof of Theorem 4.1. The main difference is that, here, we need to carefully handle the actions performed by the channels between servers. Like our proof of Theorem 4.1, for every subset $\mathcal{N} \subset \{1,2,\ldots,N\}$ where $|\mathcal{N}| = N-f$, we construct an execution $\alpha^{(v_1,v_2)}$ where the servers in $\{1,2,\ldots,N\} - \mathcal{N}$ fail at the beginning of the execution. The execution has two write operations for distinct values v_1 and v_2. The second write operation which writes value v_2 begins after the termination of the first write operation, which has value v_1.

After the point of termination of the first write, if we let the channels between servers deliver all the gossip messages, and begin a read operation after the delivery of these messages, the read can return v_1. Similarly, after the termination of the second write operation, if we let the channels between servers deliver all the gossip messages, a read can return v_2. This implies that, in the interval of the second write operation, there are two consecutive points Q_1 and Q_2 as follows: for $i \in \{1, 2, \}$, if at point Q_i we stop the writers and the channels from the writers and let the channels between servers deliver all the gossip messages to arrive at point Q_i, then v_i must be returnable by a read operation that begins at point Q_i. By ensuring that gossip messages are delivered in the same order after Q_i, we can ensure that after the delivery of the messages, at most 2 servers differ in their states. We use this fact to show that the number of elements in the set of possible server states after the delivery of the gossip messages is at most $\prod_{n\in\mathcal{N}} |\mathcal{S}_i| \times \max_{n\in\mathcal{N}} |\mathcal{S}_n| \times \max_{n\in\mathcal{N}} |\mathcal{S}_n| \times (N-f)^2$. Therefore, we get $\prod_{n\in\mathcal{N}} |\mathcal{S}_i| \times (\max_{n\in\mathcal{N}} |\mathcal{S}_n|)^2 \times (N-f)^2 \geq (|\mathcal{V}|)(|\mathcal{V}|-1)$, which implies the lower bound.

5.1 Detailed Proof Sketch of Theorem 5.1

We provide a proof sketch of Theorem 5.1 here. Omitted details can be found in [7].

5.1.1 Execution $\alpha^{(v_1,v_2)}$

Let $\mathcal{N}$ be an arbitrary subset of $\{1, 2, \ldots, N\}$ with $N-f$ elements. Like the proof of Theorem 4.1, for every tuple $(v_1, v_2) \in \mathcal{V} \times \mathcal{V}$ where $v_1 \neq v_2$, we create an execution $\alpha^{(v_1,v_2)}$ of algorithm A. The f servers in $\{1, 2, \ldots, N\} - \mathcal{N}$ fail at the beginning of $\alpha^{(v_1,v_2)}$. The execution $\alpha^{(v_1,v_2)}$ has two complete write operations π_1 and π_2 with values v_1 and v_2, with π_2 being invoked after the termination of π_1.

Let $P_0^{(v_1,v_2)}, P_1^{(v_1,v_1)}, P_2^{(v_1,v_2)}, \ldots, P_M^{(v_1,v_2)}$ be a sequence of consecutive points in execution $\alpha^{(v_1,v_2)}$, where $P_0^{(v_1,v_2)}$ is an arbitrary point after the termination of π_1 and before the invocation of π_2, and $P_M^{(v_1,v_2)}$ is an arbitrary point after the point of termination of π_2. We denote by $\alpha_i^{(v_1,v_2)}$, the execution between the initial point of $\alpha^{(v_1,v_2)}$ and point $P_i^{(v_1,v_2)}$.

5.1.2 Properties of Execution $\alpha^{(v_1,v_2)}$

The definition of k-valent points are similar to Definitions 4.2, $k = 1, 2$, with the exception that we allow the channels the servers to deliver their messages before the invocation of the read operation.

Definition 5.2 (k-valent, $k \in \{1, 2\}$). *For $i \in \{0, 1, 2, \ldots, M\}$, a point $P_i^{(v_1,v_2)}$ in the constructed $\alpha_i^{(v_1,v_2)}$ is said to be k-valent if we can extend $\alpha_i^{(v_1,v_2)}$ to an execution β as follows: After $P_i^{(v_1,v_2)}$ all the messages from and to the writer are delayed indefinitely. At $P_i^{(v_1,v_2)}$ all the channels between the servers act, delivering all their messages. After the delivery of the messages in the channels between the servers, a read operation starts and all the components, except the writer and the channels from and to the writer, execute their protocols until the read operation terminates. The read operation returns v_k.*

We inherit the definition of critical points from Definition 4.6. The only change is that the terms k-valent points use the modified definition. Results analogous to Lemmas 4.3, 4.4 and 4.5 in the Section 4 hold, with the modified definition of k-valent points. Lemma 4.7 requires some minor modifications as follows.

Lemma 5.3 (Analogous to Lemma 4.7). *Let (Q_1, Q_2) be a pair of critical points of execution $\alpha^{(v_1,v_2)}$. Then, (i) the readers, and the channels between the readers and the servers, are all in the same state at point Q_2 as at point Q_1; (ii) there is at most one non-failing server such that its state at Q_1 is different from its state at Q_2; (iii) among all the channels between the servers, there is at most one channel whose state at Q_1 is different from its state at Q_2.*

5.1.3 Proof of Theorem 5.1

From Lemma 5.3, we note that there is at most one non-failing server and one channel that changes its state between a pair of critical points $Q_1^{(v_1,v_2)}$ and $Q_2^{(v_1,v_2)}$. Let s denote the non-failing server which changes state between points $Q_1^{(v_1,v_2)}$ and $Q_2^{(v_1,v_2)}$, if there is one; if not, let s denote an arbitrary non-failing server. Let s' be a non-failing server such that the channel from s'' to s' change its state between $Q_1^{(v_1,v_2)}$ and $Q_2^{(v_1,v_2)}$, for some server s'', if there is such a server [4]; let s' be an arbitrary non-failing server if there is not.

We next present constructions of two executions, $\beta_1^{(v_1,v_2)}$ and $\beta_2^{(v_1,v_2)}$. We know that $Q_1^{(v_1,v_2)}$ is the point $P_i^{(v_1,v_2)}$ for some $i \in \{0, 1, \ldots, M-1\}$. Create $\beta_1^{(v_1,v_2)}$ as an extension of $\alpha_i^{(v_1,v_2)}$. At point $P_i^{(v_1,v_2)}$, the writer and channels from the writer stop performing actions, the channels between the servers deliver all their messages. Denote this point by $R_1^{(v_1,v_2)}$. A read operation begins after $R_1^{(v_1,v_2)}$ and returns v_1. Such an execution exits because $Q_1^{(v_1,v_2)}$ is 1-valent.

The execution $\beta_2^{(v_1,v_2)}$ follows $\alpha^{(v_1,v_2)}$ until point $Q_2^{(v_1,v_2)}$. At point $Q_2^{(v_1,v_2)}$, all the channels between the servers act delivering all their messages. For a server j in $\{1, 2, \ldots, N-f\} - \{s, s'\}$, the channels with destination j are at the same state at $Q_2^{(v_1,v_2)}$ as they are at $Q_1^{(v_1,v_2)}$; these channels act and deliver their messages in the same order as they do after point $Q_1^{(v_1,v_2)}$ in $\beta_1(v_1, v_2)$. At point $Q_2^{(v_1,v_2)}$, server $j \in \mathcal{N} - \{s, s''\}$ is at the same state as it was at point $Q_1^{(v_1,v_2)}$. Also, at point $Q_2^{(v_1,v_2)}$, server j receives messages in the same order as it does at point $Q_1^{(v_1,v_2)}$ in $\beta_1^{(v_1,v_2)}$; on receiving each message, server j takes the same action in $\beta_2^{(v_1,v_2)}$ as it does in $\beta_1^{(v_1,v_2)}$. The channels with destinations s or s' deliver messages in some arbitrary order, and servers s and s' perform actions based on the protocol specified by algorithm A. We denote this point as $R_2^{(v_1,v_2)}$. It is worth noting that at point $R_2^{(v_1,v_2)}$, all the channels are empty, and every server in $\mathcal{N} - \{s, s'\}$ is at the same state as it is at point $R_1^{(v_1,v_2)}$ in $\beta_1^{(v_1,v_2)}$. At $R_2^{(v_1,v_2)}$, a read operation begins, all the components except the writer and the channels from and to the writer act in a fair manner until the read returns. Because the point $Q_2^{(v_1,v_2)}$ is 2-valent but not 1-valent, the read returns v_2 in $\beta_2^{(v_1,v_2)}$.

[4]Between $Q_1^{(v_1,v_2)}, Q_2^{(v_1,v_2)}$, if there is a server s that changes its state, and a channel between two servers s'' and s' that changes its state, it is easy to show that $s \in \{s', s''\}$. We nonetheless use distinct notation for servers s, s', s'' since it simplifies presentation.

Let $\vec{S}^{(v_1,v_2)}$ be an element of $\prod_{n\in\mathcal{N}}\mathcal{S}_n \times \mathcal{N} \times \cup_{n\in\mathcal{N}}\mathcal{S}_n \times \mathcal{N} \times \cup_{n\in\mathcal{N}}\mathcal{S}_n$ as follows. The first $N-f$ components of $\vec{S}^{(v_1,v_2)}$ denote the states of the $N-f$ servers in $\mathcal{N}$ at point $R_1^{(v_1,v_2)}$. The $(N-f+1)$st component of $\vec{S}^{(v_1,v_2)}$ denotes the server index s and $(N-f+2)$nd component denotes the state of server s at point $R_2^{(v_1,v_2)}$ in $\alpha^{(v_1,v_2)}$. The $(N-f+3)$nd component denotes the server index s' and the $(N-f+4)$th component denotes the state of server s' at point $R_2^{(v_1,v_2)}$ in execution $\beta_2^{(v_1,v_2)}$. Note that the number of elements in the set $\bigcup_{(v_1,v_2)\in\mathcal{V}\times\mathcal{V},v_1\neq v_2}\{\vec{S}^{(v_1,v_2)}\}$ is at most $\prod_{i\in\mathcal{N}}|\mathcal{S}_i| \times (N-f)^2 \times (\max_{i\in\mathcal{N}}|\mathcal{S}_i|)^2$.

To prove Theorem 5.1, we show that, if (v_1,v_2) and (v_1',v_2') are two *distinct* elements of the set $\{(x,y) : (x,y)\in\mathcal{V}\times\mathcal{V}, x\neq y\}$, then $\vec{S}^{(v_1,v_2)} \neq \vec{S}^{(v_1',v_2')}$. If we show this, then it implies that the number of elements in the set

$$\bigcup_{(v_1,v_2)\in\mathcal{V}\times\mathcal{V},v_1\neq v_2}\{\vec{S}^{(v_1,v_2)}\}$$

is at least equal to the number of elements in the set $\{(x,y) : (x,y)\in\mathcal{V}\times\mathcal{V}, x\neq y\}$, which is equal to $(|\mathcal{V}|)\times(|\mathcal{V}|-1)$. This leads to the following inequality:

$$\prod_{n\in\mathcal{N}}|\mathcal{S}_n| \times (N-f)^2 \times (\max_{n\in\mathcal{N}}|\mathcal{S}_n|)^2 \geq (|\mathcal{V}|)\times(|\mathcal{V}|-1)$$

which implies the theorem.

The following lemma is analogous to Lemma 4.8.

LEMMA 5.4. *Let $\vec{S}^{(v_1,v_2)} = \vec{S}^{(v_1',v_2')}$. Consider the composite automaton formed by the servers, the readers, the channels between the servers, and the channels between the readers and servers. For $k\in\{1,2\}$, every component of this system at point $R_k^{(v_1,v_2)}$ in $\beta_k^{(v_1,v_2)}$ is identical to the state of the corresponding component at point $R_k^{(v_1',v_2')}$ in execution $\beta_k^{(v_1',v_2')}$.*

Using the above lemma and similar arguments as in Theorem 4.1, we can prove that for two distinct tuples (v_1,v_2) and (v_1',v_2') in $\{(x,y) : (x,y)\in\mathcal{V}\times\mathcal{V}, x\neq y\}$, we have $\vec{S}^{(v_1,v_2)} \neq \vec{S}^{(v_1',v_2')}$. Therefore, the proof of Theorem 5.1 is completed.

6. STORAGE LOWER BOUND FOR A RESTRICTED CLASS OF ALGORITHMS

In this section, we study a restricted class of MWSR algorithms where the write protocols have specific structure. In our restricted class, the write protocols consist of a fixed number of phases. During an operation the writer performs *black-box* actions that handle objects obliviously, without regard to the actual value. We assume that there is only one phase where a message containing information about the actual value is sent to the servers. The formal statement of our assumptions on the write protocol in Section 6.1 is somewhat technically involved. However the write protocols of most previous algorithms [13, 1, 4, 21, 12, 6, 5] satisfy our assumptions. After stating our assumptions, we state in Theorem 6.5 in Section 6.2, a storage cost lower bound that applies to the class of algorithms that we study. The lower bound of Theorem 6.5 is much larger than the bound of Theorems 4.1 and 5.1, and is close to the costs of previously developed algorithms.

6.1 Protocol Assumptions

Assumption 1: *The state of a write client during a write operation is of the form $(v, m, h(v,m))$ where $v\in\mathcal{V}$ is the value of the write operation, m is an element of a set $\mathcal{M}$, and $h(m,v)$ is the value of function h whose domain is $\mathcal{V}\times\mathcal{M}$ and range is a finite set.*

The set $\mathcal{M}$ is referred to as the metadata set of the write protocol of the algorithm. The function $h(m,v)$ can contain components of the send buffers that depend on the value, and hashed values used for verification to handle Byzantine adversaries [2, 15, 12]. To describe Assumption 2, we first define the notion of a quorum system and a phase. A quorum system $\mathcal{Q}$ is a collection of subsets of $\{1,2,\ldots,N\}$.

DEFINITION 6.1 (PHASE). *For an arbitrary subset $\mathcal{N}\subseteq\{1,2,\ldots,N\}$ and a quorum system $\mathcal{Q}$, a $(\mathcal{N},\mathcal{Q})$-phase consists of a sequence of actions at a write client as follows: (i) Send message m_n to server node n for* every *$n\in\mathcal{N}$. (ii) Wait for responses from least one subset of servers in the collection $\mathcal{Q}$. (iii) Perform internal actions, and finish the phase.*

DEFINITION 6.2 (DECOMPOSABLE INTO PHASES). *We say a write protocol is* decomposable into phases *if, on the invocation of a write operation, it invokes a phase, and on the termination of a phase, it either invokes another phase, or terminates the write operation.*

Assumption 2: *The write protocol is decomposable into phases.*

Before we state Assumption 3, we state the notion of the black-box action. Informally, actions treat the data object obliviously without regard to the actual value of the object. Recall that the write protocol is specified as a set of transitions (old-state, action, new-state).

DEFINITION 6.3 (BLACK-BOX ACTION). *An internal or output action σ performed by a write client is said to be a black-box action if the following holds:* ***if****, for some value $v\in\mathcal{V}$, the action σ is enabled when the client's state is $(m, v, h(m,v))$ for some $m\in\mathcal{M}$, and can result in the transition of the client's state from $(m, v, h(m,v))$ to $(m', v, h(m,v))$ for some $m'\in\mathcal{M}$,* ***then****, for every value $v'\in\mathcal{V}$, the action σ is enabled when the client's state is $(m, v', h(m,v'))$, and can result in the transition of the client's state from $(m, v', h(m,v'))$ to $(m', v', h(m',v'))$.*

A write client's output actions are send and return. Send actions are categorized as below.

DEFINITION 6.4 (VALUE-DEPENDENT SEND ACTIONS). *A black-box send action σ that is enabled during a write operation is said to be value-independent if the message sent does not depend on the value of the operation. A send action that is not a value-independent send action is referred to as a value-dependent send action.*

Messages sent by value-dependent and value-independent send actions are respectively referred to as value-dependent and value-independent messages.

Assumption 3: (a) *All write client actions are black-box actions,* and **(b)** *in a write operation π in an execution α, if there is a phase where at least one value-dependent send action is performed, then every send action in every subsequent phase of the write operation π is a value-independent send action.*

In particular, Assumption 3(b) implies that there is at most one phase where the writer sends value-dependent messages on behalf of a write operation in any execution.

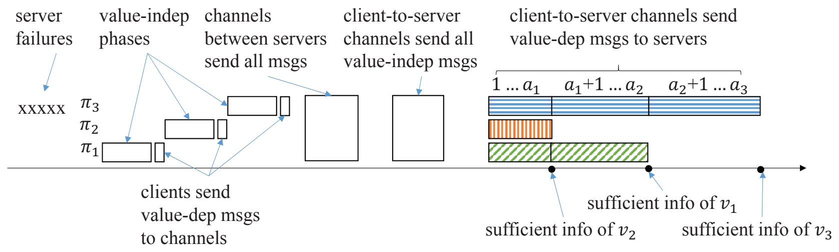

Figure 1: Pictorial description of the execution for $\nu = 3$. $b_1 = 2, b_2 = 1, b_3 = 3$.

6.2 Main Result

In an execution α, a write operation π is said to be active at point P if the point P is after the point of invocation but before the point of termination of π.

THEOREM 6.5. *Let A be a multi-writer-single-reader shared memory emulation algorithm that implements an atomic read-write object whose values come from a finite set $\mathcal{V}$. Suppose algorithm A satisfies Assumptions 1, 2, and 3 stated in Section 6.1, and following liveness property: In a fair execution of A, if the number of server failures is no bigger than f and the number of active write operations is no bigger than ν, then every operation invoked at a non-failing client terminates. Let $\nu^* = \min(\nu, f+1)$. Then, for every subset $\mathcal{N} \subseteq \{1, 2, \ldots, N\}, |\mathcal{N}| = \min(N - f + \nu - 1, N)$,*

$$\sum_{n \in \mathcal{N}} \log_2 |\mathcal{S}_n|$$

$$\geq \log_2 \binom{|\mathcal{V}| - 1}{\nu^*} - \log_2(\nu^*!) - \nu^* \log_2(N - f + \nu^* - 1),$$

$$TotalStorage(A) \geq \frac{\nu^* N}{N - f + \nu^* - 1} \log |\mathcal{V}| - o(\log |\mathcal{V}|).$$

Informal Proof Sketch: For simplicity of exposition, assume $\mathcal{N} = \{1, 2, \ldots, N - f + 2\}$. For our informal description, we set the parameter $\nu = 3$, and $f \geq \nu - 1 = 2$. Our proof of Theorem 6.5 constructs an execution where the servers in $\{N - f + 3, N - f + 4, \ldots, N\}$ fail at the beginning of the execution. The execution has $\nu = 3$ write operations π_1, π_2, π_3 with distinct values v_1, v_2, v_3 respectively invoked at distinct clients C_1, C_2, C_3. We assume that at the beginning of the execution before the invocation of any write operation, a default initial value v_0 can be returned by any read operation, and that values v_1, v_2, v_3 are distinct from the default initial value v_0.

Operations π_1, π_2, π_3 execute their protocols in a fair manner until they reach their respective phases where they send the value-dependent messages (refer to Assumption 3). The 3 clients send the value-dependent messages onto the channels, but the channels do not yet deliver the value-dependent messages.

Now allow the channels from the clients to the servers to deliver all the value-dependent messages to the first $N - f$ servers. It must be the case that the first $N - f$ servers store "sufficient information" to return at least one of the values v_1, v_2 or v_3. This is because there are no additional phases where value-dependent messages are sent, so the servers cannot receive any additional information related to v_1, v_2 or v_3. Furthermore, if we let at least one of the operations π_1, π_2, π_3 complete by performing the remaining phases, then, by safety property, at least one of the values v_1, v_2 or v_3 must be returnable from the first $N - f$ servers after the completion of the operation. So it must be the case that the servers store sufficient information to return one of v_1, v_2 or v_3. Let a_1 be the smallest number such that, if the channels between the clients and the first a_1 servers deliver all their messages, then the first a_1 servers store sufficient information of value v_{b_1} for some $b_1 \in \{1, 2, 3\}$. Note that $1 \leq a_1 \leq N - f$, and sufficient information of any *one* of v_1, v_2 or v_3 is not contained from any of the first $a_1 - 1$ servers.

Now allow the channels from clients $\{C_1, C_2, C_3\} - \{C_{b_1}\}$ deliver their value-dependent messages to servers in $\{a_1 + 1, a_1 + 2, \ldots, N - f + 1\}$. After the delivery of the messages, sufficient information of one of the values v_{b_2} must be available in the first $N - f + 1$ servers for some $b_2 \in \{1, 2, 3\} - \{b_1\}$. This is because, if, after the delivery of the value-dependent messages, server a_1 stops taking actions, and clients $\{C_1, C_2, C_3\} - \{C_{b_1}\}$, and the first $N - f + 1$ servers apart from server a_1 take actions in a fair manner, then one of the operations π_1, π_2, π_3 completes; safety property implies that one of the values v_1, v_2, v_3 is returnable from the first $N - f + 1$ servers. However, note that sufficient information related to v_{b_1} is not contained in the first $a_1 - 1$ servers. As a consequence, v_{b_1} cannot be returnable from the first $N - f + 1$ servers if server a_1 does not take actions and we do not allow the value-dependent messages from client C_{b_1} to be delivered to any one of the servers $\{a_1 + 1, a_1 + 2, \ldots, N - f + 1\}$; therefore a value $v_{b_2} \neq v_{b_1}$ must be returnable from the first $N - f + 1$ servers. Let a_2 be a number with $a_1 < a_2 \leq N - f + 1$ such that, if all the channels deliver their value-dependent messages to the first a_1 servers and the channels from clients in $\{C_1, C_2, C_3\} - \{C_{b_1}\}$ deliver their value-dependent messages to the servers in $\{a_1 + 1, a_1 + 2, \ldots, a_2\}$, then sufficient information about value v_{b_2} is contained in the first a_2 servers for some $b_2 \neq b_1, b_2 \in \{1, 2, 3\}$.

Similarly, if we let the channels from remaining client in $\{C_1, C_2, C_3\} - \{C_{b_1}, C_{b_2}\}$ deliver their value-dependent messages to the servers in $\{a_2 + 1, a_2 + 2, \ldots, N - f + 2\}$, then sufficient information about the value in $\{v_1, v_2, v_3\} - \{v_{b_1}, v_{b_2}\}$ is contained from the first $N - f + 2$ servers after the delivery of the messages. At this point, sufficient information about all 3 values is contained in the first $N - f + 2$ servers. We can show that this implies that there is a one-to-one mapping from the states of the first $N - f + 2$ servers to the values in $(\mathcal{V} - \{v_0\})^3$, where v_0 is the initial value. This implies that the storage cost must be at least $\frac{3}{N-f+2} \log_2 |\mathcal{V}| + o(\log_2 |\mathcal{V}|)$.

Our proof involves developing an appropriate notion of *sufficient information of a value* that is applicable even when each server stores some arbitrary function of the values of the different versions it receives. In particular, we cannot directly borrow from other work [22], whose notion of sufficient information of a value is tied to the storage scheme imposed by the model studied.

7. CONCLUSION

This paper was motivated by open questions, Question 1 and Question 2 in Section 2. We resolved Question 1 in the paper in the negative. Although Question 2 remains open, we obtain the following insights via our results in conjunction with the result of [22]: If there is an algorithm whose storage cost is $g(\nu, N, f) \log_2 |\mathcal{V}| + o(\log_2 |\mathcal{V}|)$, where $g(\nu, N, f)$ is some real-valued function of parameters ν, N, f then $g(\nu, N, f) \geq \frac{2N}{N-f+2}$, and

- if $g(\nu, N, f) < \frac{\min(\nu, f+1)N}{N-f+\min(\nu, f+1)-1}$, then a write operation sends information about the value of the operation in multiple phases to the servers;
- if, for given values of parameters N, f, we have $g(\nu, N, f) < f+1$ for all values of ν, then, in certain executions, the servers store symbols which somehow compress values across different versions.

8. ACKNOWLEDGMENTS

This work is supported by the National Science Foundation (NSF) under grant numbers CCF-1553248, CCF-1464336, CCF-0939370, CCF-1217506, CCF-1566587, and by the Air Force Office of Scientific Research (AFOSR) under grant number FA9550-14-1-0403.

9. REFERENCES

[1] M. K. Aguilera, R. Janakiraman, and L. Xu. Using erasure codes efficiently for storage in a distributed system. In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pages 336–345. IEEE, 2005.

[2] E. Androulaki, C. Cachin, D. Dobre, and M. Vukolić. Erasure-coded Byzantine storage with separate metadata. In *Principles of Distributed Systems*, pages 76–90. Springer, 2014.

[3] H. Attiya, A. Bar-Noy, and D. Dolev. Sharing memory robustly in message-passing systems. In *Proceedings of the ninth annual ACM symposium on principles of distributed computing*, PODC '90, pages 363–375. ACM, 1990.

[4] C. Cachin and S. 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.

[5] V. Cadambe, N. Lynch, M. Medard, and P. Musial. A coded shared atomic memory algorithm for message passing architectures. In *2014 IEEE 13th International Symposium on Network Computing and Applications (NCA),*, pages 253–260, 2014.

[6] V. R. Cadambe, N. Lynch, M. Medard, and P. Musial. A coded shared atomic memory algorithm for message passing architectures. 2014. MIT CSAIL Technical Report MIT-CSAIL-TR-2014-015, http://hdl.handle.net/1721.1/88551; Also available on http://arxiv.org/abs/1407.4167.

[7] V. R. Cadambe, Z. Wang, and N. Lynch. Information-theoretic lower bounds on the storage cost of shared memory emulation. 2016. arxiv pre-print, available on http://arxiv.org/abs/1605.06844.

[8] Y. Cassuto. What can coding theory do for storage systems? *ACM SIGACT News*, 44(1):80–88, 2013.

[9] G. Chockler, D. Dobre, A. Shraer, and A. Spiegelman. Space bounds for reliable multi-writer data store: Inherent cost of read/write primitives. 2015. arXiv pre-print, available on http://arxiv.org/abs/1508.03762.

[10] G. Chockler, R. Guerraoui, and I. Keidar. Amnesic distributed storage. In *Distributed Computing*, pages 139–151. Springer, 2007.

[11] A. G. Dimakis, P. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. *IEEE Transactions on Information Theory*, 56(9):4539–4551, 2010.

[12] D. Dobre, G. Karame, W. Li, M. Majuntke, N. Suri, and M. 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.

[13] P. Dutta, R. Guerraoui, and R. R. Levy. Optimistic erasure-coded distributed storage. In *Distributed Computing*, pages 182–196. Springer, 2008.

[14] R. Fan and N. Lynch. Efficient replication of large data objects. In *In Proceedings of the 17th International Symposium on Distributed Computing (DISC)*, pages 75–91, 2003.

[15] J. Hendricks, G. R. Ganger, and M. K. Reiter. Low-overhead Byzantine fault-tolerant storage. *ACM SIGOPS Operating Systems Review*, 41(6):73–86, 2007.

[16] M. P. Herlihy and J. M. Wing. Linearizability: a correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12:463–492, July 1990.

[17] L. Lamport. On interprocess communication. Part I: basic formalism. *Distributed Computing*, 2(1):77–85, 1986.

[18] S. Lin and D. J. Costello. *Error control coding, Second edition*. Prentice-Hall, Inc., 2004.

[19] D. A. Patterson, G. Gibson, and R. H. Katz. *A case for redundant arrays of inexpensive disks (RAID)*, volume 17. ACM, 1988.

[20] R. Roth. *Introduction to coding theory*. Cambridge University Press, 2006.

[21] Y. Saito, S. Frølund, A. Veitch, A. Merchant, and S. Spence. FAB: building distributed enterprise disk arrays from commodity components. *ACM SIGOPS Operating Systems Review*, 38(5):48–58, 2004.

[22] A. Spiegelman, Y. Cassuto, G. Chockler, and I. Keidar. Space bounds for reliable storage: Fundamental limits of coding. In *Proceedings of the 2016 ACM symposium on principles of distributed computing*, PODC '16. ACM, 2016.

[23] Z. Wang and V. R. Cadambe. Multi-version coding - an information theoretic perspective of consistent distributed storage. arxiv pre-print, available on http://arxiv.org/abs/1506.00684.

[24] Z. Wang and V. R. Cadambe. Multi-version coding in distributed storage. In *2014 IEEE International Symposium on Information Theory (ISIT)*, 2014.