Optimal Register Construction in M&M Systems

Vassos Hadzilacos
Department of Computer Science, University of Toronto, Canada
vassos@cs.toronto.edu

Xing Hu
Department of Computer Science, University of Toronto, Canada
xing@cs.toronto.edu

Sam Toueg
Department of Computer Science, University of Toronto, Canada
sam@cs.toronto.edu

Abstract

Motivated by recent distributed systems technology, Aguilera *et al.* introduced a hybrid model of distributed computing, called *message-and-memory model* or *m&m model* for short [1]. In this model, processes can communicate by message passing and also by accessing some shared memory. We consider the basic problem of implementing an atomic single-writer multi-reader (SWMR) register shared by *all* the processes in m&m systems. Specifically, we give an algorithm that implements such a register in m&m systems and show that it is optimal in the number of process crashes that it can tolerate. This generalizes the well-known implementation of an atomic SWMR register in a pure message-passing system [4].

2012 ACM Subject Classification Theory of computation → Concurrency; Theory of computation → Parallel computing models; Theory of computation → Distributed computing models

Keywords and phrases asynchronous distributed system, shared memory, message passing

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2019.28

Funding This research was partially funded by the Natural Sciences and Engineering Research Council of Canada.

1 Introduction

Motivated by recent distributed systems technology [9, 12, 13, 19, 22], Aguilera *et al.* introduced a hybrid model of distributed computing, called *message-and-memory model* or *m&m model* for short [1]. In this model processes can communicate by message passing and also by accessing some shared memory. Since it is impractical to share memory among *all* processes in large distributed systems [8, 14, 15, 24], the m&m model allows us to specify which subsets of processes share which sets of registers. Among other results, Aguilera *et al.* show that it is possible to leverage the advantages of the two communication mechanisms (message-passing and share-memory) to improve the fault-tolerance of randomized consensus algorithms compared to a pure message-passing system [1].

In this paper, we consider the more basic problem of implementing an atomic single-writer multi-reader (SWMR) register shared by *all* the processes in m&m systems, and we give an algorithm that is optimal in the number of process crashes that it can tolerate. This generalizes the well-known implementation of an atomic SWMR register in a pure message-passing system [4]. We now describe our results in more detail.

A *general* m&m system $\mathcal{S}_L$ is specified by a set of n asynchronous processes that can send messages to each other over asynchronous reliable links, and by a collection $L = \{S_1, S_2, \ldots, S_m\}$ where each S_i is a subset of processes: for each S_i, there is a set of atomic registers that can be shared by processes in S_i and only by them. Even though

the m&m model allows the collection L to be arbitrary, in practice hardware technology imposes a structure on L [8, 14]: for processes to share memory, they must establish a connection between them (e.g., an RDMA connection). These connections are naturally modelled by an undirected *shared-memory graph* G whose nodes are the processes and whose edges are shared-memory connections [1]. Such a graph G defines what Aguilera *et al.* call a *uniform* m&m system $\mathcal{S}_G$, where each process has atomic registers that it can share with its neighbours in G (and only with them). Note that $\mathcal{S}_G$ is the instance of the general m&m system $\mathcal{S}_L$ with $L = \{S_1, S_2, \ldots, S_n\}$ where each S_i consists of a process and its neighbours in G. Furthermore, if G is the trivial graph with n nodes but no edges, the m&m system $\mathcal{S}_G$ that G induces is just a pure message-passing asynchronous system with n processes.

We consider the implementation of an atomic SWMR register $\mathbf{R}$, shared by all the processes, in both general and uniform m&m systems. For each general m&m system $\mathcal{S}_L$, we determine the maximum number of crashes t_L for which it is possible to implement $\mathbf{R}$ in $\mathcal{S}_L$: we give an algorithm that tolerates t_L crashes and prove that no algorithm can tolerate more than t_L crashes. Similarly, for each shared-memory graph G and its corresponding uniform m&m system $\mathcal{S}_G$, we use the topology of G to determine the maximum number of crashes t_G for which it is possible to implement $\mathbf{R}$ in $\mathcal{S}_G$. By specifying t_G in terms of the topology of G, one can leverage results from graph theory to design m&m systems that can implement $\mathbf{R}$ with high fault tolerance and relatively few RDMA connections per process. For example, it allows us to design an m&m system with 50 processes that can implement a *wait-free* $\mathbf{R}$ (i.e., this implementation can tolerate *any* number of process crashes) with only 7 RDMA connections per process; as explained in Section 4, this is optimal in some precise sense.

An important remark is now in order. In this paper we consider RDMA systems where process crashes do not affect the accessibility of the shared registers of that system. This is the case in systems where the CPU, the DRAM (main memory), and the NIC (Network Interface Controller) are separate entities: for example, in the InfiniBand cluster evaluated in [21], the crash of a CPU, and of the processes that it hosts, does not prevent other processes from accessing its DRAM because it can use the NIC without involving the CPU; see also [8, 10, 26].

2 Model outline

We consider m&m systems with a set of n asynchronous processes $\Pi = \{p_1, p_2, \ldots p_n\}$ that may *crash*. To define these systems we first recall the definition of atomic SWMR registers.

2.1 Atomic SWMR registers

A SWMR register R shared by a set S of processes is a register that can be written (sequentially) by exactly one process $w \in S$ and can be read by all processes in S; we say that w is the *writer* of R [18].

We now define an *atomic* SWMR register [4, 18] in terms of two simple properties that they must satisfy. To do so, we first define what it means for a (read or write) operation to precede another operation, and for two operations to be concurrent. We say that an operation o *precedes* another operation o' if and only if o completes before o' starts. A write operation o *immediately precedes a read operation* r if and only if o precedes r, and there is no write operation o' such that o precedes o' and o' precedes r. Operations o and o' are *concurrent* if and only if neither precedes the other.

We assume, without loss of generality, that the values successively written by the single writer w of a SWMR register R are distinct, and different from the initial value of R.[1] Let v_0 denote the *initial value* of R, and v_k denote the value written by the k-th write operation of w. A SWMR register R is *atomic* if and only if it satisfies the following two properties:

▶ **Property 1.** *If a read operation r returns the value v then:*

- *there is a write v operation that immediately precedes r or is concurrent with r, or*
- *no write operation precedes r and $v = v_0$.*

▶ **Property 2.** *If two read operations r and r' return values v_k and $v_{k'}$, respectively, and r precedes r', then $k \leq k'$.*

2.2 General m&m systems

Let $L = \{S_1, S_2, \ldots, S_m\}$ be any bag of non-empty subsets of $\Pi = \{p_1, p_2, \ldots p_n\}$.

▶ **Definition 3.** *$\mathcal{M}_L$ is the class of m&m systems (induced by L), each consisting of:*

1. *The processes in Π.*
2. *Reliable asynchronous communication links between every pair of processes in Π.*
3. *The following set of registers: For each subset of processes S_i in L, a non-empty set of atomic registers that are shared by the processes in S_i (and only by them).*

Note that $\mathcal{M}_L$ includes m&m systems that differ by the type and number of registers shared by the processes in each S_i; for example they could be sharing multi-writer multi-reader atomic registers.

Since we are interested in implementing atomic SWMR registers (shared by *all* processes in the system), here we focus on an m&m system of $\mathcal{M}_L$ in which the set of registers shared by the processes in each set S_i are atomic SWMR registers. More precisely, we focus on the m&m system $\mathcal{S}_L$ defined below:

▶ **Definition 4.** *The general m&m system $\mathcal{S}_L$ (induced by L) consists of:*

1. *The processes in Π.*
2. *Reliable asynchronous communication links between every pair of processes in Π.*
3. *The following set of registers: For each subset of processes S_i in L and each process $p \in S_i$, an atomic SWMR register, denoted $R_i[p]$, that can be written by p and read by all processes in S_i (and only by them).*

In this paper, for every L, we give an algorithm that implements atomic SWMR registers shared by all processes in the m&m system $\mathcal{S}_L$, and we show that this algorithm is optimal in the number of process crashes that can be tolerated. In fact we prove that any algorithm that implements such registers in *any* m&m system in $\mathcal{M}_L$, (not only in $\mathcal{S}_L$) cannot tolerate more crashes. This justifies our focus on $\mathcal{S}_L$: considering members of $\mathcal{M}_L$ with more or stronger registers than $\mathcal{S}_L$ does not improve the fault tolerance of implementing atomic SWMR registers shared by all.

Without loss of generality we assume the following:

▶ **Assumption 5.** *The bag $L = \{S_1, S_2, \ldots, S_m\}$ of subsets of $\Pi = \{p_1, p_2, \ldots p_n\}$ is such that every process in Π is in at least one of the subsets S_j of L.*

[1] This can be ensured by the writer w writing values of the form $\langle sn, v\rangle$ where sn is the value of a counter that w increments before each write.

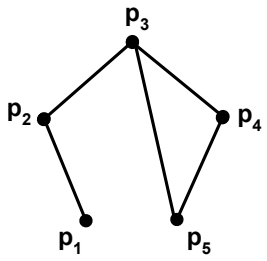

Figure 1 A graph G.

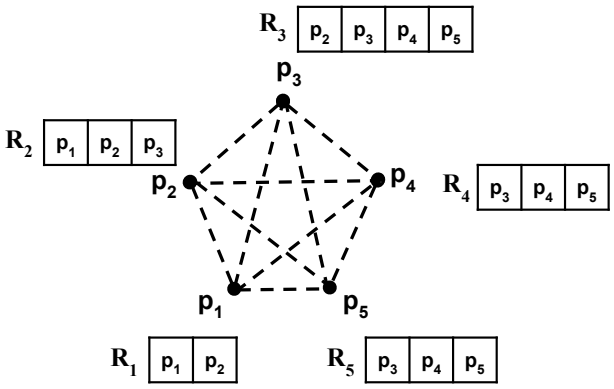

Figure 2 The uniform m&m system $\mathcal{S}_G$ induced by G.

This assumption can be made without loss of generality because it does not strengthen the system $\mathcal{S}_L$ induced by L. In fact, given a bag L that does *not* satisfy the above assumption, we can construct a bag that satisfies the assumption as follows: for every process p_i in Π that is *not* contained in any S_j of L, we can add the singleton set $\{p_i\}$ to L. Let L' be the resulting bag. By Definition 4(3) above, adding $\{p_i\}$ to L results in adding only a *local* register to the induced system $\mathcal{S}_L$, namely, an atomic register that p_i (trivially) shares only with itself. So $\mathcal{S}_{L'}$ is just $\mathcal{S}_L$ with some additional local registers. Note that a pure message-passing system (with no shared memory) with n processes $p_1, p_2, \ldots p_n$ is modelled by the system $\mathcal{S}_L$ where $L = \{\{p_1\}, \{p_2\}, \ldots \{p_n\}\}$.

2.3 Uniform m&m systems

Let $G = (V, E)$ be an undirected graph such that $V = \Pi$, i.e., the nodes of G are the n processes $p_1, p_2, \ldots, p_n$ of the system. For each $p_i \in V$, let $N(p_i) = \{p_j \mid (p_i, p_j) \in E\}$ be the *neighbours* of p_i in G, and let $N^+(p_i) = N(p_i) \cup \{p_i\}$.

▶ **Definition 6.** *The uniform m&m system $\mathcal{S}_G$ (induced by G) is the m&m system $\mathcal{S}_L$ where $L = \{S_1, S_2, \ldots, S_n\}$ with $S_i = N^+(p_i)$.*[2]

The graph G induces the uniform m&m system $\mathcal{S}_G$ where processes can communicate by message passing (via reliable asynchronous communication links), and also by shared memory as follows: for each process p_i, and every neighbour p of p_i in G (including p_i) there is an atomic SWMR register $R_i[p]$ that can be written by p and read by every neighbour of p_i in G (including p_i). We can think of the registers $R_i[*]$ as being physically located in the DRAM of the host of p_i, and every neighbour of p_i accessing these registers over its RDMA connection to p_i (which is modelled by an edge of G).[3]

For example, in Figures 1 and 2 we show a graph G and the uniform m&m system $\mathcal{S}_G$ induced by G. Here G has five nodes representing processes p_1, p_2, p_3, p_4, p_5; the edges of G represent the RDMA connections that allow these processes to share registers. The uniform m&m system $\mathcal{S}_G$ induced by G is the system $\mathcal{S}_L$ for $L = \{S_1, S_2, S_3, S_4, S_5\}$ where

[2] Note that L satisfies Assumption 5 because each $S_i = N^+(p_i)$ contains p_i.

[3] As we mentioned in the introduction, we assume that the crash of p_i does not prevent the neighbours of p_i from accessing the shared registers $R_i[*]$.

each S_i consists of p_i and its neighbours in G: specifically, $S_1 = \{p_1, p_2\}, S_2 = \{p_1, p_2, p_3\}$, $S_3 = \{p_2, p_3, p_4, p_5\}$, and $S_4 = S_5 = \{p_3, p_4, p_5\}$. The box adjacent to each process p_i in $\mathcal{S}_G$ represents the atomic SWMR registers that are shared among p_i and its neighbours in G (intuitively these registers are located at p_i). For example, in the box adjacent to process p_2, the component labelled p_1 represents the register $R_2[p_1]$ that can be written by p_1 and read by all the neighbours of p_2 in G, namely p_1, p_2, and p_3. Similarly, registers $R_2[p_2]$ and $R_2[p_3]$ can be written by p_2 and p_3, respectively, and read by p_1, p_2, and p_3. The dashed lines in Figure 2 represent the asynchronous message-passing links between the processes of $\mathcal{S}_G$.

3 Atomic SWMR register implementation in general m&m systems

Let $\mathcal{S}_L$ be the general m&m system induced by a bag $L = \{S_1, \ldots, S_m\}$ of subsets of $\Pi = \{p_1, p_2, \ldots p_n\}$. Recall that in system $\mathcal{S}_L$, for every S_i in L, the processes in S_i share some atomic SWMR registers that can be read *only* by the processes in S_i (recall that it is impractical to share registers among *all* processes in large distributed systems [8, 14, 15, 24]). In the rest of the paper, we determine the maximum number of process crashes t_L that may occur in $\mathcal{S}_L$ such that it is possible to *implement* in $\mathcal{S}_L$ a shared atomic SWMR register readable by *all* processes in $\mathcal{S}_L$. Intuitively, if $t \leq t_L$ processes may crash, then any two subsets of processes of size $n - t$ either intersect, or they each contain a process that can communicate with the other via a shared SWMR register that it can write and the other can read. If $t > t_L$ processes may crash, then there are two subsets of processes of size $n - t$ that are disjoint and cannot communicate via shared SWMR register.

▶ **Definition 7.** *Given a bag $L = \{S_1, \ldots, S_m\}$ of subsets of $\Pi = \{p_1, p_2, \ldots p_n\}$, t_L is the maximum integer t such that the following condition holds: For all disjoint subsets P and P' of Π of size $n - t$ each, some set S_i in L contains both a process in P and a process in P'.*

Note that if $t \leq \lfloor (n-1)/2 \rfloor$ then there are *no* disjoint subsets P and P' of Π of size $n-t$ each, and so the above condition is vacuously true. Therefore $t_L \geq \lfloor (n-1)/2 \rfloor$. Recall that for a pure message-passing system, $L = \{\{p_1\}, \{p_2\}, \ldots \{p_n\}\}$, so in this system $t_L = \lfloor (n-1)/2 \rfloor$.

To illustrate Definition 7, suppose $\Pi = \{p_1, p_2, p_3, p_4, p_5\}$ and $L = \{S_1, S_2, S_3\}$ where $S_1 = \{p_1, p_2\}$, $S_2 = \{p_4, p_5\}$, and $S_3 = \{p_2, p_4, p_3\}$. By the definition of t_L: (1) $t_L \geq 3$ because for any two disjoint subsets of Π of size $5 - 3 = 2$ each, there exists a set S_i in L that intersects both subsets; e.g., for subsets $\{p_1, p_5\}$ and $\{p_3, p_4\}$, the set $S_2 = \{p_4, p_5\}$ intersects both of them. (2) $t_L < 4$ because there are two disjoint subsets $\{p_1\}, \{p_5\}$ of size $5 - 4 = 1$ each, such that no set S_i in L contains both p_1 and p_5. So in this example $n = 5$ and $t_L = 3 > \lfloor (5-1)/2 \rfloor = 2$.

We now prove that in the general m&m system $\mathcal{S}_L$, it is possible to implement an atomic SWMR register readable by *all* processes *if and only if* at most t_L processes may crash in $\mathcal{S}_L$. More precisely:

▶ **Theorem 8.** *Let $\mathcal{S}_L$ be the general m&m system induced by a bag $L = \{S_1, \ldots, S_m\}$ of subsets of $\Pi = \{p_1, p_2, \ldots p_n\}$.*

- *If at most t_L processes crash in $\mathcal{S}_L$, then for every process w in $\mathcal{S}_L$, it is possible to implement an atomic SWMR register writable by w and readable by **all** processes in $\mathcal{S}_L$.*
- *If more than t_L processes crash in $\mathcal{S}_L$, then for some process w in $\mathcal{S}_L$, it is impossible to implement an atomic SWMR register writable by w and readable by **all** processes in $\mathcal{S}_L$.*

The above theorem is a direct corollary of Theorem 18 (Section 3.1) and Theorem 19 (Section 3.2).

3.1 Algorithm

We now show how to implement an atomic SWMR register $\mathbf{R}$, that can be written by an arbitrary fixed process w and read by *all* processes, in an m&m system $\mathcal{S}_L$ where at most t_L processes may crash. This implementation is given in terms of the procedures WRITE() and READ() shown in Algorithm 1.

Without loss of generality we assume that for all $i \geq 1$, the i-th value that the writer writes is of the form $\langle i, val \rangle$, and the initial value of the register $\mathbf{R}$ is $\langle 0, u_0 \rangle$. To write $\langle i, val \rangle$ into $\mathbf{R}$, the writer w calls the procedure WRITE($\langle i, val \rangle$). To read $\mathbf{R}$, a process q calls the procedure READ() which returns a value of the form $\langle i, val \rangle$. The sequence number i makes the values written to $\mathbf{R}$ unique.

Algorithm 1 Implementation of an atomic SWMR register writeable by process w and readable by all processes in $\mathcal{S}_L$, provided that at most t_L processes crash.

SHARED VARIABLES

For all S_i in L and all p in S_i:

$R_i[p]$: atomic SWMR register writeable by p and readable by every process in $S_i \in L$; initialized to $\langle 0, u_0 \rangle$.

WRITE($\langle sn_w, u \rangle$): ▷ executed by the writer w

1: **send** $\langle \text{W}, \langle sn_w, u \rangle \rangle$ **to every process** p in $\mathcal{S}_L$
2: **wait for** $\langle \text{ACK-W}, sn_w \rangle$ **messages from** $n - t_L$ **distinct processes**
3: **return**

▷ executed by every process p in $\mathcal{S}_L$

4: **upon receipt of a** $\langle \text{W}, \langle sn_w, u \rangle \rangle$ **message from process** w:
5: **for every** i in $\{1, \ldots, m\}$ such that $p \in S_i$ **do**
6: $\langle sn, - \rangle \leftarrow R_i[p]$
7: **if** $sn_w > sn$ **then**
8: $R_i[p] \leftarrow \langle sn_w, u \rangle$
9: **send** $\langle \text{ACK-W}, sn_w \rangle$ **to process** w

READ(): ▷ executed by any process q

10: $sn_r \leftarrow sn_r + 1$
11: **send** $\langle \text{R}, sn_r \rangle$ **to every process** p in $\mathcal{S}_L$
12: **wait for** $\langle \text{ACK-R}, sn_r, \langle -, - \rangle \rangle$ **messages from** $n - t_L$ **distinct processes**
13: $\langle seq, val \rangle \leftarrow \max\{\langle r\_sn, r\_u \rangle \mid$ received a $\langle \text{ACK-R}, sn_r, \langle r\_sn, r\_u \rangle \rangle$ message$\}$
14: WRITE($\langle seq, val \rangle$)
15: **return** $\langle seq, val \rangle$

▷ executed by every process p in $\mathcal{S}_L$

16: **upon receipt of a** $\langle \text{R}, sn_r \rangle$ **message from a process** q:
17: $\langle r\_sn, r\_u \rangle \leftarrow \max\{\langle sn, u \rangle \mid \exists i \in \{1, \ldots, m\}, p \in S_i$ and $\exists p' \in S_i, R_i[p'] = \langle sn, u \rangle\}$
18: **send** $\langle \text{ACK-R}, sn_r, \langle r\_sn, r\_u \rangle \rangle$ **to process** q

Algorithm 1 generalizes the well-known implementation of an atomic SWMR register in pure message-passing systems [4]. To write a new value into $\mathbf{R}$, the writer w sends messages to all processes asking them to write the new value into all the shared SWMR registers that they can write in $\mathcal{S}_L$. The writer then waits for acknowledgments from $n - t_L$ processes indicating that they have done so. To read $\mathbf{R}$, a process sends messages to all processes asking them for the most up-to-date value that they can find in all the shared SWMR registers that they can read. The reader waits for $n - t_L$ responses, selects the most up-to-date value among them, writes back that value (using the same procedure that the writer uses), and returns it. From the definition of t_L it follows that every write of $\mathbf{R}$ "intersects" with every read of $\mathbf{R}$ at some shared SWMR register of $\mathcal{S}_L$. Note that since at most t_L processes crash, the waiting mentioned above does not block any process.

We now show that the procedure WRITE(), called by the writer w, and the procedure READ(), called by any process q in $\mathcal{S}_L$, implement an atomic SWMR register $\mathbf{R}$. To do so, we show that the calls of WRITE() *by* w and of READ() by any process satisfy Properties 1 and 2 of atomic SWMR registers given in Section 2.1.

▶ **Definition 9.** *The operation write*(v) *is the execution of* WRITE(v) ***by the writer*** w *for some tuple* $v = \langle sn_w, u\rangle$*: this operation starts when* w *calls* WRITE(v) *and it completes if and when this call returns. An operation read*(v) *is an execution of* READ() *that returns* v *to some process* q*: this operation starts when* q *calls* READ() *and it completes when this call returns* v *to* q*.*

Let $v_0 = \langle 0, u_0\rangle$ be the initial value of the implemented register $\mathbf{R}$, and, for $k \geq 1$, let $v_k = \langle k, -\rangle$ denote the k-th value written by the writer w on $\mathbf{R}$. Note that all v_k's are distinct: for all $i \neq j \geq 0, v_i \neq v_j$.

Let $\mathcal{S}_L$ be the general m&m system induced by a bag $L = \{S_1, \ldots, S_m\}$ of subsets of $\Pi = \{p_1, p_2, \ldots p_n\}$. To prove the correctness of the SWMR implementation shown in Algorithm 1, we now consider an arbitrary execution of this implementation in $\mathcal{S}_L$ under the assumption that *at most* t_L *processes crash.*

▶ **Lemma 10.** *Any read*$(-)$ *or write*$(-)$ *operation executed by a process that does not crash completes.*

Proof. The only statements that could prevent the completion of a *read*$(-)$ or *write*$(-)$ operation are the **wait** statements of line 2 and line 12. But since communication links are reliable, these wait statements are for $n - t_L$ acknowledgements, and at most t_L processes out of the n processes of $\mathcal{S}_L$ may crash, it is clear that these wait statements cannot block. ◀

We first note that every read operation returns some v_k for $k \geq 0$.

▶ **Lemma 11.** *If* r *is a read*(v) *operation in the execution, then* $v = v_k$ *for some* $k \geq 0$*.*

Proof. This proof is straightforward and omitted here. ◀

The next lemma says that no read operation can read a "future" value, i.e., a value that is written *after* the read completes.

▶ **Lemma 12.** *If* r *is a read*(v) *operation in the execution, then either* $v = v_0$*, or* $v = v_k$ *such that the operation write*(v_k) *precedes* r *or is concurrent with* r*.*

Proof. This proof is straightforward and omitted here. ◀

Note that the guard in lines 6-8 (which is the only place where the shared SWMR registers are updated) ensures that the content of each shared SWMR register in $\mathcal{S}_L$ is non-decreasing in the following sense:

▶ **Observation 13.** *[Register monotonicity] For all $1 \leq i \leq m$ and all $p \in S_i$, if $R_i[p] = \langle k, - \rangle$ at some time t and $R_i[p] = \langle k', - \rangle$ at some time $t' \geq t$ then $k' \geq k$.*

▶ **Lemma 14.** *For all $k \geq 1$, if a call to the procedure* WRITE(v_k) *returns before a read(v) operation starts, then $v = v_\ell$ for some $\ell \geq k$.*

Proof. Suppose a call to WRITE(v_k) returns before a *read*(v) operation starts; we must show that $v = v_\ell$ with $\ell \geq k$. Note that before this call of WRITE(v_k) returns, $\langle$ACK-W$, k\rangle$ messages are received from a set P of $n - t_L$ distinct processes (see line 2 of the WRITE() procedure). From lines 5-8, which are executed before these $\langle$ACK-W$, k\rangle$ messages are sent, and by Observation 13, it is now clear that the following holds:

▶ **Claim 14.1.** *By the time* WRITE(v_k) *returns, every shared SWMR register in $\mathcal{S}_L$ that can be written by a process in P contains a tuple $\langle k', - \rangle$ with $k' \geq k$.*

Now consider the *read*(v) operation, say it is by process q. Recall that *read*(v) is an execution of the READ() procedure that returns v to q. When q calls READ(), it increments a local counter sn_r and asks every process p in $\mathcal{S}_L$ to do the following: (a) read every SWMR register that p can read, and (b) reply to q with a $\langle$ACK-R$, sn_r, \langle r_sn, r_u\rangle\rangle$ message such that $\langle r\_sn, r\_u\rangle$ is the tuple with the maximum r_sn that p read. By line 12 of the READ() procedure, q waits to receive such $\langle$ACK-R$, sn_r, \langle -, -\rangle\rangle$ messages from a set P' of $n - t_L$ distinct processes, and q uses these messages to select the value v as follows:

$$v \leftarrow \max\{(r\_sn, r\_u) \mid q \text{ received some } \langle \text{ACK-R}, sn_r, \langle r\_sn, r\_u\rangle\rangle \text{ from a process in } P'\}$$

Thus, by Lemma 11, it is clear that:

▶ **Claim 14.2.** *$v = v_\ell$ where $\ell = \max\{j \mid q$ received a $\langle$ACK-R$, sn_r, \langle j, -\rangle\rangle$ message from a process in $P'\}$.*

▶ **Claim 14.3.** *Some set S_i in L contains both a process in P and a process in P'.*

Proof of Claim 14.3. If P and P' are disjoint, the claim follows directly from Definition 7 of t_L. If P and P' intersect, let p be a process in both P and P'. By Assumption 5, p is in some set S_i in L, and the claim follows. ◀

By the above claim, some set S_i in L contains a process p in P and a process p' in P'. Since $p \in S_i$ and $p' \in S_i$, $R_i[p]$ is one of the SWMR registers that can be written by p and read by p'. From Claim 14.1, by the time the call to WRITE(v_k) returns, $R_i[p]$ contains a tuple $\langle k', - \rangle$ such that $k' \geq k$ (*). Since $p' \in P'$, during the execution of *read*(v) by q, p' reads all the shared SWMR registers that it can read, including $R_i[p]$. Since *read*(v) starts after WRITE(v_k) returns, p' reads $R_i[p]$ after WRITE(v_k) returns. Thus, by (*) and the monotonicity of $R_i[p]$ (Observation 13), p' reads from $R_i[p]$ a tuple $\langle r\_sn, - \rangle$ with $r\_sn \geq k' \geq k$. Then p' selects the tuple $\langle j, - \rangle$ with the maximum sn among all the $\langle sn, - \rangle$ tuples that it read (see line 17); note that $j \geq k$. So the $\langle$ACK-R$, sn_r, \langle j, -\rangle\rangle$ message that p' sends to q, and q uses to select v, is such that $j \geq k$. So, by Claim 14.2, $v = v_\ell$ such that $\ell \geq j \geq k$. ◀

Lemma 14 immediately implies the following:

▶ **Corollary 15.** *For all $k \geq 1$, if a write(v_k) operation precedes a read(v) operation then $v = v_\ell$ with $\ell \geq k$.*

We now show that Algorithm 1 satisfies Property 1 and 2 of atomic SWMR registers.

▶ **Lemma 16.** *The write$(-)$ and read$(-)$ operations satisfy Property 1.*

Proof. Suppose for contradiction that Property 1 does not hold. Thus there is a read operation $r = read(v)$ such that:

(a) there is no $write(v)$ operation that immediately precedes r or is concurrent with r, and

(b) some $write(-)$ operation precedes r, or $v \neq v_0$.

There are two cases.

1. $v = v_0$. By (b) above, some $write(-)$ operation, say $write(v_k)$, precedes r. Thus $write(v_k)$ precedes $read(v_0)$. Since $k \geq 1$ this contradicts Corollary 15.
2. $v \neq v_0$. By Lemma 12, $v = v_k$ such that the operation $write(v_k)$ precedes r, or $write(v_k)$ is concurrent with r. By (a) above, $write(v_k)$ does not *immediately* precede r, and $write(v_k)$ is not concurrent with r. Thus, $write(v_k)$ precedes, but not *immediately*, r. Let $write(v_{k'})$ be the write operation that immediately precedes r. Note that $write(v_k)$ precedes $write(v_{k'})$, so $k < k'$. Since $write(v_{k'})$ precedes $r = read(v)$, by Corollary 15, $v = v_\ell$ with $\ell \geq k'$, so $\ell > k$. This contradicts that $v = v_k$.

Since both cases lead to a contradiction, Property 1 holds. ◀

▶ **Lemma 17.** *The write$(-)$ and read$(-)$ operations satisfy Property 2.*

Proof. We have to show that if a $read(v_k)$ operation precedes a $read(v_{k'})$ operation then $k \leq k'$. Suppose $read(v_k)$ precedes $read(v_{k'})$. Note that during the $read(v_k)$ operation, namely in line 14, there is a call to the procedure Write(v_k) which returns before the $read(v_k)$ operation completes. So this call to Write(v_k) returns before the $read(v_{k'})$ operation starts. By Lemma 12, $k \leq k'$. ◀

Lemmas 10, 16 and 17 immediately imply:

▶ **Theorem 18.** *Let $\mathcal{S}_L$ be the general m&m system induced by a bag $L = \{S_1, \ldots, S_m\}$ of subsets of $\Pi = \{p_1, p_2, \ldots p_n\}$. If at most t_L processes crash in $\mathcal{S}_L$, for every process w in $\mathcal{S}_L$, Algorithm 1 implements an atomic SWMR register writable by w and readable by **all** processes in $\mathcal{S}_L$.*

3.2 Lower bound

▶ **Theorem 19.** *Let $\mathcal{S}_L$ be the general m&m system induced by a bag $L = \{S_1, \ldots, S_m\}$ of subsets of $\Pi = \{p_1, p_2, \ldots p_n\}$. If more than t_L processes crash in $\mathcal{S}_L$, for some process w in $\mathcal{S}_L$, there is no algorithm that implements an atomic SWMR register writable by w and readable by **all** processes in $\mathcal{S}_L$.*

Proof. Let $\mathcal{S}_L$ be the general m&m system induced by a bag $L = \{S_1, \ldots, S_m\}$ of subsets of $\Pi = \{p_1, p_2, \ldots p_n\}$. Suppose for contradiction that $t > t_L$ processes can crash in $\mathcal{S}_L$, but for every process w in $\mathcal{S}_L$, there is an algorithm $\mathcal{A}_w$ that implements an atomic SWMR register writable by w and readable by **all** processes in $\mathcal{S}_L$ (*).

Since $t > t_L$, by the Definition 7 of t_L there are two disjoint subsets P and P' of Π, of size $n - t$ each, such that: no set S_i in L contains both a process in P and a process in P' (**). Since P and P' are disjoint, the sets P, P', and $Q = \Pi - (P \cup P')$ form a partition of Π.

Let the writer w be any process in P. Let $\mathcal{A}$ be an algorithm that tolerates $t > t_L$ process crashes in $\mathcal{S}_L$ and implements an atomic SWMR register $\mathbf{R}$ that is writable by w and readable by **all** processes in $\mathcal{S}_L$; this algorithm exists by our initial assumption (*).

Since $|P \cup Q \cup P'| = n$, clearly $|P \cup Q| = |P' \cup Q| = n - (n - t) = t$. Since algorithm $\mathcal{A}$ tolerates t crashes, it works correctly in every execution in which all the processes in $P \cup Q$ or in $P' \cup Q$ crash.

We now define three executions E_1, E_2, and E_3 of algorithm $\mathcal{A}$. These are illustrated in Figure 3.

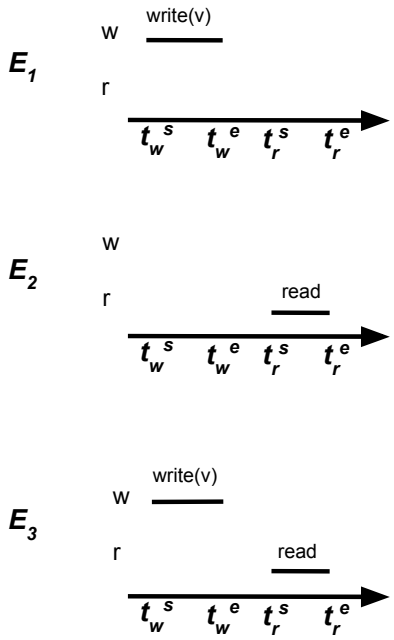

Figure 3 Scenarios for Theorem 19.

Execution E_1 of algorithm $\mathcal{A}$ is defined as follows:

- The processes in $P' \cup Q$ crash from the beginning of the execution; they take no steps in E_1.
- At some time t_w^s the writer w starts an operation to write the value v into the implemented register $\mathbf{R}$, for some $v \neq v_0$, where v_0 is the initial value of $\mathbf{R}$. Since the number of processes that crash in E_1 is $|P' \cup Q| = t$, and the algorithm $\mathcal{A}$ tolerates t crashes, this write operation eventually terminates, say at time t_w^e.
- After this write terminates, no process takes a step up to and including some time $t_r^s > t_w^e$.

Note that in E_1, processes in P are the only ones that take steps up to time t_r^s.

Execution E_2 of algorithm $\mathcal{A}$ is defined as follows:

- The processes in $P \cup Q$ crash from the beginning of the execution; they take no steps in E_2.
- At time t_r^s, some process $r \in P'$ starts a read operation on the implemented register $\mathbf{R}$. Since the number of processes that crash in E_2 is $|P \cup Q| = t$, and the algorithm $\mathcal{A}$ tolerates t crashes, this read operation terminates, say at time t_r^e.

Since no write operation precedes the read operation in E_2, Property 1 of atomic SWMR registers implies:

▶ **Claim 19.1.** *At time t_r^e in E_2 the read operation returns the initial value v_0 of $\mathbf{R}$.*

We now construct an execution E_3 of the algorithm $\mathcal{A}$ that merges E_1 and E_2, and contradicts the atomicity of the implemented $\mathbf{R}$. E_3 is identical to E_1 up to time t_r^s, and it is identical to E_2 from time t_r^s to t_r^e (note that in E_3 processes in Q can only take steps *after* time t_r^e). To obtain this merged run E_3, intuitively we delay the messages sent by processes in P to processes in P' to after time t_r^e, and we also use the fact that processes in P' cannot read any of the shared registers in $\mathcal{S}_L$ that processes in P may have written by time t_r^s (this is because of (**)).

▶ **Claim 19.2.** *There is an execution E_3 of algorithm $\mathcal{A}$ such that*
(a) *up to and including time t_w^e, E_3 is indistinguishable from E_1 to all processes.*
(b) *up to and including time t_r^e, E_3 is indistinguishable from E_2 to all processes in P'.*
(c) *No process crashes in E_3.*

Proof of Claim 19.2. Until time t_r^s, E_3 is identical to E_1. We now show that it is possible to extend E_3 in the time interval $[t_r^s, t_r^e]$ with the sequence of steps that the processes in P' executed during the same time interval in E_2.[4] More precisely, let $s^1, s^2, \ldots, s^\ell$ be the sequence of steps executed during the time interval $[t_r^s, t_r^e]$ in E_2. Since only processes in P' take steps in E_2, $s^1, s^2, \ldots, s^\ell$ are all steps of processes in P'. Let C_2^0 be the configuration of the system $\mathcal{S}_L$ at time t_r^s in E_2,[5] and let C_2^i be the configuration that results from applying step s^i to configuration C_2^{i-1}, for all i such that $1 \leq i \leq \ell$. We will prove that there are configurations $C_3^0, C_3^1, \ldots, C_3^\ell$ of $\mathcal{S}_L$ extending E_3 at time t_r^s such that:

(i) every process in P' has the same state in C_3^i as in C_2^i;
(ii) the set of messages sent by processes in P' to processes in P', but not yet received, is the same in C_3^i as in C_2^i;
(iii) every shared register readable by processes in P' has the same value in C_3^i as in C_2^i; and
(iv) if $i \neq 0$, C_3^i is the result of applying step s^i to configuration C_3^{i-1}.

This is shown by induction on i.

For the basis of the induction, $i = 0$, we take C_3^0 to be the configuration of the system just before time t_r^s in E_3. Since no process in P' takes a step before time t_r^s in either E_2 or E_3, C_3^0 satisfies properties (i) and (ii).

▶ **Claim 19.3.** *At time t_r^s in E_3 the shared registers that can be read by processes in P' have their initial values.*

Proof of Claim 19.3. Suppose, for contradiction, that at time t_r^s in E_3, some shared register R that can be read by a process p' in P' does not have its initial value. By construction, E_3 is identical to E_1 until time t_r^s, and so only processes in P take steps before time t_r^s in E_3. Thus, register R was written by some process p in P by time t_r^s in E_3. Since R is readable by $p' \in P'$ and is written by $p \in P$, R is shared by both p and p'. Thus, there must be a set S_i in L that contains both p and p' – a contradiction to (**). ◀

By Claim 19.3, the shared registers readable by processes in P' have the same value (namely, their initial value) in C_3^0 as in C_2^0. So, C_3^0 also satisfies property (iii). Property (iv) is vacuously true for $i = 0$.

[4] A *step* of $\mathcal{A}$ executed by process p is one of the following: p sending or receiving a message, or p applying a write or a read operation to a shared register in $\mathcal{S}_L$.
[5] The *configuration* of $\mathcal{S}_L$ at time t in execution E consists of the state of each process, the set of messages sent but not yet received, and the value of each shared register in $\mathcal{S}_L$ at time t in E.

For the induction step, for each i such that $1 \leq i \leq \ell$, we consider separately the cases of s^i being a step to send a message, receive a message, write a shared register, and read a shared register. In each case, it is easy to verify that, assuming (inductively) that C_3^{i-1} has properties (i)–(iv), step s^i is applicable to C_3^{i-1}, and the resulting configuration C_3^i has properties (i)–(iv).

To complete the definition of E_3, after time t_r^e we let processes take steps in round-robin fashion. Whenever a process's step is to receive a message, it receives the oldest one sent to it; this ensures that all messages are eventually received. Processes continue taking steps in this fashion according to algorithm $\mathcal{A}$.

Since E_3 is identical to E_1 up to and including time t_w^e, E_3 is indistinguishable from E_1 up to and including time t_w^e to all processes in P. This proves part (a) of the claim.

Note that in E_3 and E_2, the processes in P': (a) take no steps before time t_r^s, and (b) during the time interval $[t_r^s, t_r^e]$, they execute exactly the same sequence of steps, and go through the same sequence of states. Thus, up to and including time t_r^e, E_3 is indistinguishable from E_2 to all processes in P'. This proves part (b) of the claim.

Finally, every process takes steps as required by the algorithm in E_3, so no process crashes. This proves part (c) of the claim. ◀

By Claim 19.2(a), up to and including time t_w^e, E_3 is indistinguishable from E_1 to the writer $w \in P$. So E_3 contains the write operation that writes $v \neq v_0$ into $\mathbf{R}$, which starts at time t_w^s and ends at time t_w^e. By Claim 19.2(b), up to and including time t_r^e, E_3 is indistinguishable from E_2 to $r \in P'$. So E_3 contains the read operation that returns v_0, which starts at time t_r^s and ends at time t_r^e. Since $t_w^e < t_r^s$, this read operation violates Property 1 of atomic SWMR registers. As there are no process crashes in E_3 (by Claim 19.2(c)), this contradicts the assumption that $\mathcal{A}$ is an implementation of an atomic SWMR register $\mathbf{R}$ that tolerates $t > t_L$ crashes. ◀

Note that the proof of Theorem 19 does not depend on the *type* or *number* of registers shared by the processes in each set S_i of the bag L. So the result of this theorem applies not only to $\mathcal{S}_L$ but also to every m&m system in $\mathcal{M}_L$. In fact, the proof of Theorem 19 does not even depend on the type of objects that are shared by the processes in each set S_i; for example these objects could include queues, stacks, and consensus objects. Hence we have the following stronger result:

▶ **Theorem 20.** *Consider any m&m system $\mathcal{S}$ induced by a bag $L = \{S_1, \ldots, S_m\}$ of subsets of $\Pi = \{p_1, p_2, \ldots p_n\}$, where the processes in each S_i share any number of arbitrary objects among themselves (and only among themselves). If more than t_L processes crash in $\mathcal{S}$, then for some process w in $\mathcal{S}$, there is no algorithm that implements an atomic SWMR register writable by w and readable by **all** processes in $\mathcal{S}$.*

4 Atomic SWMR register implementation in uniform m&m systems

Let $G = (V, E)$ be an undirected graph where $V = \{p_1, p_2, \ldots p_n\}$, i.e., the nodes of G are the processes $p_1, p_2, \ldots p_n$. Let $\mathcal{S}_G$ be the uniform m&m system induced by G. Recall that in $\mathcal{S}_G$, each process p_i and its neighbours in G share some atomic SWMR registers that can be read by (and only by) them.

We now use G to determine the maximum number of process crashes that may occur in $\mathcal{S}_G$ such that it is possible to implement a shared atomic SWMR register readable by *all* processes in $\mathcal{S}_G$. To do so, we first recall the definition of the *square of the graph* G: $G^2 = (V, E')$ where $E' = \{(u, v) \mid (u, v) \in E$ or $\exists k \in V$ such that $(u, k) \in E$ and $(k, v) \in E\}$.

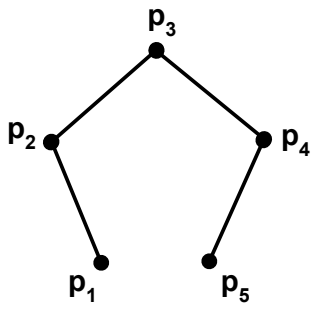

Figure 4 A graph G.

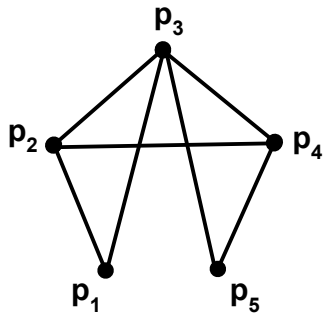

Figure 5 The square of graph G.

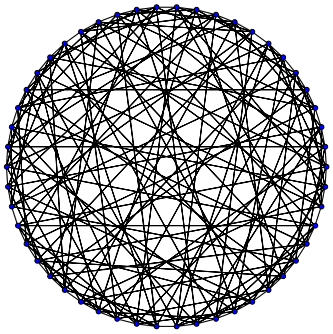

Figure 6 The Hoffman-Singleton graph.

▶ **Definition 21.** *Given an undirected graph $G = (V, E)$ such that $V = \{p_1, p_2, \ldots p_n\}$, t_G is the maximum integer t such that the following condition holds: For all disjoint subsets P and P' of V of size $n - t$ each, some edge in G^2 connects a node in P with a node in P'; i.e., G^2 has an edge (u, v) such that $u \in P$ and $v \in P'$.*

Note that $t_G \geq \lfloor (n-1)/2 \rfloor$. Moreover, in a pure message-passing system (where G and G^2 have no edges) $t_G = \lfloor (n-1)/2 \rfloor$.

In Theorem 22 stated below, we prove that in the uniform m&m system $\mathcal{S}_G$ induced by a graph G, it is possible to implement an atomic SWMR register readable by all processes *if and only if* at most t_G processes may crash in $\mathcal{S}_G$.

For example, consider the graph G in Figure 4 where $V = \{p_1, p_2, p_3, p_4, p_5\}$. Figure 5 shows the corresponding G^2 graph. By the above definition of t_G: (a) $t_G \geq 3$ because for any two disjoint subsets of V of size $5 - 3 = 2$ each, G^2 has an edge that "connects" these two subsets (e.g., for subsets $P = \{p_1, p_2\}$ and $P' = \{p_4, p_5\}$, the edge (p_2, p_4) of G^2 connects a node of P to a node of P'), and (b) $t_G < 4$ because there are two disjoint subsets $\{p_1\}, \{p_5\}$ of size $5 - 4 = 1$ each, such that no edge in G^2 connects p_1 and p_5. So in this example $n = 5$ and $t_G = 3 > \lfloor (5-1)/2 \rfloor = 2$.

Now consider the uniform m&m system $\mathcal{S}_G$ of 5 processes induced by this graph G. In addition to message-passing links, $\mathcal{S}_G$ has 4 pairwise RDMA connections. Since $t_G = 3$, by Theorem 22: (1) we can implement an atomic SWMR register readable by *all* 5 processes of $\mathcal{S}_G$ even if 3 of them (i.e., more than the majority) may crash, and (2) no algorithm can implement such a register in $\mathcal{S}_G$ if more than 3 processes may crash.

As another example, consider a pure message-passing system $\mathcal{S}$ with 50 nodes. In $\mathcal{S}$, one can implement an atomic SWMR register $\mathbf{R}$ (readable by all the processes) only if *at most* 24 process crashes may occur. But if we allow each process of $\mathcal{S}$ to establish 7 pairwise RDMA connections, one can implement $\mathbf{R}$ in a way that tolerates *any number* of process crashes (i.e., $\mathbf{R}$ is wait-free). This is because there is an undirected graph G with $n = 50$ nodes, each with degree 7, such that G^2 has an edge between *every* pair of nodes (G is the well-known Hoffman-Singleton graph [11] shown in Figure 6 [25]); so G has $t_G = n - 1 = 49$, and thus by Theorem 22 it is possible to implement $\mathbf{R}$ in the uniform m&m system $\mathcal{S}_G$ in a way that tolerates up to 49 process crashes. Some simple graph theory arguments show that this is optimal in two ways: (a) one cannot implement a wait-free register $\mathbf{R}$ that is shared by 50 processes with fewer than 7 RDMA connections per process (more precisely, with any such implementation, if a process has fewer than 7 RDMA connections there must be another process with more than 7 RDMA connections), and (b) with at most 7 RDMA connections per process, one cannot implement a wait-free register $\mathbf{R}$ that is shared by more than 50 processes.

▶ **Theorem 22.** *Let $\mathcal{S}_G$ be the uniform m&m system induced by an undirected graph $G = (V, E)$ where $V = \{p_1, p_2, \ldots p_n\}$.*

- *If at most t_G processes crash in $\mathcal{S}_G$, then for every process w in $\mathcal{S}_G$, it is possible to implement an atomic SWMR register writable by w and readable by **all** processes in $\mathcal{S}_G$.*
- *If more than t_G processes crash in $\mathcal{S}_G$, then for some process w in $\mathcal{S}_G$, it is impossible to implement an atomic SWMR register writable by w and readable by **all** processes in $\mathcal{S}_G$.*

Proof. By Definition 6, $\mathcal{S}_G$ is the m&m system $\mathcal{S}_L$ where $L = \{S_1, S_2, \ldots, S_n\}$ such that $S_i = N^+(p_i)$, i.e., for all i, $1 \leq i \leq n$, S_i is the set of neighbours of p_i (including p_i) in the graph G. Recall that t_L is the maximum t such that for all disjoint subsets P and P' of V of size $n - t$ each, some set S_i in L contains both a node in P and a node in P'.

▶ **Claim 22.1.** $t_G = t_L$.

Proof of Claim 22.1. From the definitions of t_G and t_L, it is clear that to prove the claim it suffices to show that for all disjoint subsets P and P' of V of size $n - t$ each, the following holds: some edge in G^2 connects a node in P with a node in P' if and only if some set S_i in L contains both a node in P and a node in P'.

[Only If] Suppose G^2 has an edge (p_i, p_j) such that $p_i \in P$ and $p_j \in P'$; since P and P' are disjoint, p_i and p_j are distinct. By definition of G^2, there are two cases:

1. $(p_i, p_j) \in E$. In this case, $p_j \in N^+(p_i)$ and $p_i \in N^+(p_i)$. So the set $S_i = N^+(p_i)$ in L contains both node $p_i \in P$ and node $p_j \in P'$.
2. There is a node $p_k \in V$ such that $(p_i, p_k) \in E$ and $(p_k, p_j) \in E$. In this case, $p_i \in N^+(p_k)$ and $p_j \in N^+(p_k)$. So the set $S_k = N^+(p_k)$ in L contains both $p_i \in P$ and $p_j \in P'$.

So in both cases, some set S_ℓ in L contains both a node in P and a node in P'.

[If] Suppose set S_k in L contains both a node p_i in P and a node p_j in P'; since P and P' are disjoint, p_i and p_j are distinct. Recall that $S_k = N^+(p_k)$ for node $p_k \in V$.

There are two cases:

1. p_i, p_j and p_k are distinct. In this case, since p_i and p_j are in $S_k = N^+(p_k)$, (p_i, p_k) and (p_k, p_j) are edges of G, i.e., $(p_i, p_k) \in E$ and $(p_k, p_j) \in E$. Thus, by definition of G^2, (p_i, p_j) is an edge of G^2; this edge connects $p_i \in P$ and $p_j \in P'$.
2. $p_k = p_i$ or $p_k = p_j$. Without loss of generality, assume that $p_k = p_i$. Since p_i and p_j are in $N^+(p_k) = N^+(p_i)$, (p_i, p_j) must be an edge of G, i.e., $(p_i, p_j) \in E$. Thus, by definition of G^2, (p_i, p_j) is an edge of G^2; this edge connects $p_i \in P$ and $p_j \in P'$.

So in both cases, some edge in G^2 connects a node in P with a node in P'. ◀

The result now follows immediately from Claim 22.1 and Theorem 8. ◀

5 Concluding remarks

Hybrid systems that combine message passing and shared memory have long been a subject of study in the systems community [3, 5, 6, 7, 16, 17, 20, 23]. To the best of our knowledge, however, such systems have only recently been examined from a theoretical point of view. Aguilera *et al.* gave a rigorous model for hybrid systems, and studied how the combination of message passing and shared memory can be harnessed to improve solutions to certain fundamental problems: In particular, they show that, compared to a pure message-passing system, a hybrid system can improve the fault tolerance of randomized consensus algorithms and reduce the synchrony necessary to elect a leader [1]. A more recent paper by Aguilera *et al.* extends the hybrid model to Byzantine failures, and shows how to improve the inherent trade-off between fault tolerance and performance for consensus, for both Byzantine and

crash failures [2]. The present paper is another contribution to the theoretical study of hybrid systems: whereas the highly cited paper by Attiya *et al.* shows how to implement an atomic SWMR register with optimal fault tolerance in a pure message-passing system [4], here we solve the corresponding problem in hybrid systems. Extending our results to hybrid systems with Byzantine failures is a subject for future research.

References

1. Marcos K. Aguilera, Naama Ben-David, Irina Calciu, Rachid Guerraoui, Erez Petrank, and Sam Toueg. Passing Messages while Sharing Memory. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 51–60, 2018. doi:10.1145/3212734.3212741.
2. Marcos K. Aguilera, Naama Ben-David, Rachid Guerraoui, Virendra Marathe, and Igor Zablotchi. The Impact of RDMA on Agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019.*, pages 409–418, 2019. doi:10.1145/3293611.3331601.
3. Cristiana Amza, Alan L. Cox, Shandya Dwarkadas, Pete Keleher, Honghui Lu, Ramakrishnan Rajamony, Weimin Yu, and Willy Zwaenepoel. TreadMarks: Shared memory computing on networks of workstations. *IEEE Computer*, 29(2):18–28, February 1996.
4. Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. Sharing memory robustly in message-passing systems. *Journal of the ACM*, 42(1):124–142, January 1995.
5. Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania. The Multikernel: A New OS Architecture for Scalable Multicore Systems. In *ACM Symposium on Operating Systems Principles*, pages 29–44, October 2009.
6. John K. Bennett, John B. Carter, and Willy Zwaenepoel. Munin: Distributed Shared Memory Based on Type-specific Memory Coherence. In *ACM Symposium on Principles and Practice of Parallel Programming*, pages 168–176, March 1990.
7. Tudor David, Rachid Guerraoui, and Maysam Yabandeh. Consensus Inside. In *International Middleware Conference*, pages 145–156, December 2014.
8. Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. FaRM: Fast remote memory. In *Symposium on Networked Systems Design and Implementation*, pages 401–414, April 2014.
9. Gen-Z Draft Core Specification—December 2016. URL: http://genzconsortium.org/draft-core-specification-december-2016.
10. Gen-Z DRAM and Persistent Memory Theory of Operation. URL: https://genzconsortium.org/wp-content/uploads/2019/03/Gen-Z-DRAM-PM-Theory-of-Operation-WP.pdf.
11. Alan J. Hoffman and Robert R. Singleton. On Moore Graphs with Diameters 2 and 3. *IBM Journal of Research and Development*, 4(5):497–504, 1960. doi:10.1147/rd.45.0497.
12. InfiniBand. http://www.infinibandta.org/content/pages.php?pg=about_us_infiniband.
13. iWARP. https://en.wikipedia.org/wiki/IWARP.
14. Anuj Kalia, Michael Kaminsky, and David G. Andersen. Using RDMA Efficiently for Key-value Services. In *ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 295–306, August 2014.
15. Anuj Kalia, Michael Kaminsky, and David G. Andersen. FaSST: Fast, scalable and simple distributed transactions with two-sided (RDMA) datagram RPCs. In *Symposium on Operating Systems Design and Implementation*, pages 185–201, November 2016.
16. Stefanos Kaxiras, David Klaftenegger, Magnus Norgren, Alberto Ros, and Konstantinos Sagonas. Turning Centralized Coherence and Distributed Critical-Section Execution on Their Head: A New Approach for Scalable Distributed Shared Memory. In *IEEE International Symposium on High Performance Distributed Computing*, pages 3–14, June 2015.

17 David Kranz, Kirk Johnson, Anant Agarwal, John Kubiatowicz, and Beng-Hong Lim. Integrating Message-passing and Shared-memory: Early Experience. In *ACM Symposium on Principles and Practice of Parallel Programming*, pages 54–63, 1993.

18 Leslie Lamport. On interprocess communication Part I–II. *Distributed Computing*, 1(2):77–101, May 1986.

19 Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K. Reinhardt, and Thomas F. Wenisch. Disaggregated Memory for Expansion and Sharing in Blade Servers. In *International Symposium on Computer Architecture*, pages 267–278, June 2009.

20 Jacob Nelson, Brandon Holt, Brandon Myers, Preston Briggs, Luis Ceze, Simon Kahan, and Mark Oskin. Latency-tolerant Software Distributed Shared Memory. In *USENIX Annual Technical Conference*, pages 291–305, July 2015.

21 Marius Poke and Torsten Hoefler. DARE: High-Performance State Machine Replication on RDMA Networks. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '15, pages 107–118, New York, NY, USA, 2015. ACM. doi:10.1145/2749246.2749267.

22 RDMA over converged ethernet. https://en.wikipedia.org/wiki/RDMA_over_Converged_Ethernet.

23 Daniel J. Scales, Kourosh Gharachorloo, and Chandramohan A. Thekkath. Shasta: A Low Overhead, Software-only Approach for Supporting Fine-grain Shared Memory. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 174–185, October 1996.

24 Shin-Yeh Tsai and Yiying Zhang. LITE kernel RDMA support for datacenter applications. In *ACM Symposium on Operating Systems Principles*, pages 306–324, October 2017.

25 Figure by Uzyel - Own work, CC BY-SA 3.0. https://commons.wikimedia.org/w/index.php?curid=10378641.

26 Jian Yang, Joseph Izraelevitz, and Steven Swanson. Orion: A Distributed File System for Non-Volatile Main Memory and RDMA-Capable Networks. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*, pages 221–234, Boston, MA, February 2019. USENIX Association. URL: https://www.usenix.org/conference/fast19/presentation/yang.