Using Hysteresis to Reduce Adaptation Cost of a Dynamic Quorum Assignment*

Mark Bearden
Bell Labs, Lucent Technologies, Murray Hill, New Jersey, USA
mbearden@research.bell-labs.com
http://www.bell-labs.com/~mbearden

Abstract
One classic technique for coordinating distributed computations is to require each processor to get permission for certain actions from some quorum of processors, such that every processor’s quorum overlaps every other processor’s quorum. A dynamic quorum assignment allows the processor-to-quorum mapping to adapt during system execution, to improve performance and availability when processors fail or change load. This work considers how the run-time quorum adaptation cost is impacted by the selection of a quorum mapping function. An effective quorum mapping function exhibits not only desirable quorum size and load properties, but also a type of hysteresis that minimizes the changes made to the processor-to-quorum mapping whenever the mapping is recomputed. A new quorum mapping function called MEMRING is given that exhibits hysteresis by identifying quorums that are similar to previously chosen quorums. This behavior reduces the number of modifications to dynamic quorum assignment data structures, and can consequently reduce the amount of interprocessor communication needed for distributed control of quorum adaptation. The expected cost of distributed quorum adaptation using MEMRING is shown to be less than the expected cost of using other quorum mapping functions that have similar quorum size and load properties.

1. Introduction
Quorum systems have long been investigated as a decentralized way to enforce distributed system-level safety properties. Well-known examples include the use of quorums for mutually exclusive access to a shared resource [19,25] and consistency protocols for replicated data [10,18,26]. More recently, quorum systems have been considered for masking the effects of malicious network intruders [20] and for replication of process state [9]. In a quorum system, each processor that wishes to execute some potentially unsafe action \(A\), such as an update to replicated data, is first required to obtain permission for \(A\) from a subset of processors called a quorum. For any processing actions \(A_1\) and \(A_2\) that could lead to a safety violation if performed unsafely (e.g., in the wrong order), the quorums \(Q_1\) and \(Q_2\) that grant permission for \(A_1\) and \(A_2\) to occur are required to have an intersection that contains at least one processor. This quorum intersection property ensures it is possible to prevent processors from independently choosing sets of actions that are unsafe in conjunction. For example, some processor in the intersection of \(Q_1\) and \(Q_2\) could enforce a safe ordering of events \(A_1\) and \(A_2\), or could refuse to give permission for an unsafe event. Because quorum systems enable decentralized control, their advantages include increased concurrency and fault tolerance. This paper considers the operation of quorum systems in general, without reference to any particular safety property that can be enforced using quorums. The correctness of general quorum-based control is proved in [7], and surveys of previous quorum systems research are available in [11,14,24,27].

A quorum assignment is the mapping from each processor to the one or more quorums that can grant the processor permission to take a desired action. A dynamic quorum assignment can adapt when the state of a distributed system changes, to improve performance. For example, processors that are failed or overloaded can be removed from quorums to improve availability, and processors added to the system or recovered from failures can be included in quorums to improve load distribution. Quorum systems typically identify the quorum assignment using distributed data structures that are maintained in the local storage of multiple processors. Certain inconsistencies among the distributed data structures can violate the quorum intersection property. Thus safely adapting a dynamic quorum assignment necessarily requires a non-trivial amount of interprocessor coordination as analyzed in [5], especially in the presence of processor failures and distributed system asynchrony.

The dynamic quorum assignments described in [4,12,13,18,25] give fault-tolerant quorum adaptation protocols which ensure that all quorums intersect at all times. However, each of these protocols imposes at least one of two limitations: (1) quorum adaptation cannot occur when certain sets of processors have simultaneously failed or been removed from the system, or (2) restrictions are...
placed on the size and distribution of quorums that may be mapped to processors, in addition to the restrictions that are necessary for quorum intersection. In contrast, this work considers the model known as generalized on-line quorum adaptation, or generalized OQA, which decouples the two run-time tasks of (1) updating the processor-to-quorum mapping when the system state changes, and (2) making the needed changes to distributed data structures that allow processors to identify quorums. Since these two tasks are performed separately, there is no restriction on what method is used to map quorums to processors (other than the quorum intersection requirement). Thus quorums can be mapped to processors in a way that achieves any satisfiable performance objective, such as minimizing quorum size. A solution to generalized OQA enables quorum adaptation when any subset of processors can fail, recover from failure, or leave or join the system. It is assumed that a quorum adaptation mechanism exists to ensure the progress of quorum adaptation under these assumptions, without unnecessary restrictions on the set of quorums that can be chosen. The work in [6] describes how such a quorum adaptation mechanism can be implemented, but does not consider the question of how to effectively select the quorums for a generalized OQA solution. The present research addresses this question.

A quorum mapping function computes the set of quorums that are mapped to each processor. A large number of quorum mapping functions have been proposed by previous research; the work in [1,2,8,15,17,19] gives a few examples. Different quorum mapping functions result in quorum assignments that impose varying communication and processing costs and have varying vulnerability to processor failures. Generalized OQA leaves the quorum system designer free to choose a quorum mapping function that is best for a given application of quorums. This choice can be based on a number of quorum system metrics that predict the cost and performance of quorum assignments. Traditional metrics include the quorum size, quorum load, and quorum availability metrics discussed in [21,27]. Quorum size is the number of processors in each quorum; load is the number of processors a processor is contained in; and availability is the probability that all members of a quorum are simultaneously not faulty.

The quorum size, load, and availability metrics are not alone sufficient for predicting the cost of using different quorum mapping functions for generalized OQA, because none of these metrics indicates the expected costs of modifying the quorum assignment. The present research introduces a change sensitivity metric, formally defined in Section 3, which is the first known metric for predicting the quorum adaptation cost that results from using a given quorum mapping function. The change sensitivity metric quantifies how many changes to distributed quorum data structures are required to install new quorum assignments, and can be extended to predict the necessary message passing cost of controlling quorum adaptation. This work presents a new quorum mapping function called MEMRING, which is the first quorum mapping function designed to minimize change sensitivity as well as quorum size and load. MEMRING searches for a new quorum assignment that is similar to the most previously used quorum assignment. Thus MEMRING utilizes what is “remembered” about its previous output to optimize its present output. This behavior is referred to as hysteresis.1

This paper is organized as follows. The remainder of this section gives a formal definition of generalized OQA and a brief review of related research. Section 2 describes the MEMRING mapping function. Section 3 compares measurements of the change sensitivity of the MEMRING function and four other mapping functions. Section 4 gives conclusions and discussion of future work.

1.1 System Model and Notation

A distributed processing system contains a set \( S \) of processors that have unique identifiers. Within \( S \), there is a subset of client processors and a set of controller processors. The actions taken by clients must satisfy safety. A quorum is some subset of the controller processors. Consistent with most previous quorums research, we make the simplifying assumption that the client and controller sets are both equal to \( S \). A processor-to-quorum mapping, or quorum assignment, is determined by a quorum mapping function \( M(S) \) that returns, for each processor \( p \in S \), a set of sets called \( Qset_p \). Each \( Qset_p \) is a set of quorums selected such that the quorum intersection property, formally stated as

\[
\forall p, q, Q', Q'' \left| \left( p, q \in S \land Q' \in Qset_p \land Q'' \in Qset_q \right) \Rightarrow Q' \cap Q'' \neq \emptyset \right.
\]

is satisfied. Before executing an action \( A \) that must be coordinated with other processors’ actions, processor \( p \) must obtain permission to execute \( A \) from all of the members of at least one quorum in \( Qset_p \). A simplex quorum assignment is an assignment that maps exactly one quorum, denoted \( Q^p \), to each \( p \in S \). Given a mapping function \( M \) that computes a simplex quorum assignment, the quorum assigned to \( p \) by \( M \) is denoted \( M(S):Q^p \). To simplify the presentation of concepts in this paper, all quorum assignments are assumed to be simplex assignments. Extension of this work to non-simplex quorum assignments is straightforward2.

A dynamic quorum assignment allows the value of \( S \) and the mapping between processors and quorums to change repeatedly during system execution. Quorum adaptation is the process of generating a sequence of quorum mappings that is a function of the sequence of values taken by \( S \) during system execution. Initially, and after any change to \( S \), function \( M \) is invoked to obtain a desired quorum assignment that is optimal3 given the current value of \( S \). If the desired quorum assignment differs from the

1The term “hysteresis” is borrowed from the physical sciences, where it is used to describe a system whose output is a function of the input history rather than a function only of the current input.
2To further simplify the presentation, distinct quorums are not identified for purposes of reading and writing, as is done in some data replication protocols [12,18,26]. These extensions are also straightforward.
3The \( M \) selected by the system designer determines what is optimized. In practice, \( M \) can be a function not only of the membership of \( S \) but also of current processing loads, or any other detected state of \( S \).
quorum assignment currently in use, then the new assignment is installed, by making any necessary changes to the distributed data structures that identify the quorum assignment to processors in the system. Over time the system acquires a set of states that are identified by the value of \( S \), such that \( S \) takes the sequence of values \( S_0, S_1, ... \). After each state transition from \( S = S_i \) to \( S = S_{i+1} \), quorum mapping function \( M \) is invoked to compute the next element in the sequence of desired quorum assignments \( M(S_0), M(S_1), ... \). There are no a priori restrictions on the sequence of values taken by \( S \); thus if \( S \) models the set of non-faulty processors in the system, any number of processors may simultaneously fail or recover, or leave or join the system. Each processor \( p \in S \) is required to modify its local data structures to indicate that the processors in \( M(S_{i+1}) \cup Q^p - M(S_i) \cup Q^p \) are added to its quorum, and that the processors in \( M(S_i) \cup Q^p - M(S_{i+1}) \cup Q^p \) are removed from its quorum. For a processor joining \( S \) such that \( p \notin S \), \( M(S_i) \cup Q^p = \emptyset \) by definition. For a processor \( p \) that leaves \( S \), \( p \) is not required to modify its quorum, since \( p \) is no longer a client after leaving \( S \).

### 1.2 Comparison to Previous Work

This section briefly describes how this work differs from previously proposed quorum adaptation approaches. With the exception of the work in [6] that introduces the generalized OQA model, previous dynamic quorum assignments have not assumed that the choice of quorum mapping function and the set of allowed adaptations are unrestricted. Previous approaches fall into two categories, those that use coordinated updates to distributed quorum data structures, and those that enable processors to autonomously adapt their quorum data structures.

Examples of coordinated quorum adaptation are given in [4,12,13]. These protocols identify quorums implicitly by gathering votes whose assignment can be adjusted dynamically. The coordinated quorum adaptation approaches in [16,18] enable changes to distributed data structures that explicitly identify quorums. In each of these coordinated approaches, multi-phase updates ensure proper system-wide ordering of communication with quorums in older and newer installed quorum assignments. Permission to install a new quorum assignment must be given by all members of some existing quorum in the old quorum assignment before the new quorum assignment can be installed. If there is not at least one quorum that contains no failed or removed processors, quorum adaptation is prevented from progressing. In contrast, the present work assumes a generalized OQA model that allows the progress of quorum adaptation despite any subset of unavailable processors.

Other dynamic quorum assignments allow autonomous changes to local quorum data structures, ensuring correct adaptation without coordination even when processors have inconsistent local knowledge about the system state. Sanders’ quorums’ adaptation algorithm [25], as extended in [22], allows any set of failed processors to be removed from quorums, and any recovered processors to be added back to quorums. The algorithms of [1,8] also allow local decisions that guarantee quorum intersection is preserved, but they tolerate failure of only a minority of the processors in the system. A common limitation of these approaches is the requirement that the set of processors in the system is static and known a priori; no provision is made for adding or removing processors to/from the system. Each of these uncoordinated adaptation approaches also restricts the quorum size and load that can result after a quorum assignment is adapted. In contrast, the present work assumes that the number of processors in the system can change over time, and assumes no restriction on quorum size and load.

### 2. New Quorum Mapping Functions

This section gives two new quorum mapping functions called SHORTRING and MEMRING. SHORTRING is an incremental improvement of the balanced quorum mapping function given by Lien and Yuan in [15]. (A balanced quorum mapping function computes quorums that have uniform size, and results in uniform load across processors, where the load of \( p \in S \) is defined as the number of quorums in which \( p \) is included.) The MEMRING function is an extension of the SHORTRING function that reduces the function’s change sensitivity.

Lien and Yuan’s original quorum mapping function, referred to here as RING, sorts the processors in \( S \) by logical identifier and maps them to the nodes of a logical ring, moving (let us say) in a clockwise direction. The function computes a total of \( |S| \) quorums, one quorum per processor, by including in \( p \)’s quorum the first \( \lceil |S|/|S| \rceil \) consecutive processors in the ring, starting with \( p \) and moving clockwise, and each \( \lceil |S|/|S| \rceil \)th processor among the remaining \( |S| - \lceil |S|/|S| \rceil \) processors in the ring. The resulting quorums have uniform size and load of approximately \( 2/|S| \).

#### 2.1 The SHORTRING Function

The SHORTRING function is a simple improvement of Lien and Yuan’s RING that reduces quorum size to approximately \( 1.5/|S| \) as follows. Each quorum consists of a head set and a tail set. The head set for processor \( p \) contains the first \( \lceil |S|/|S| \rceil \) consecutive processors in the ring, starting with \( p \) and moving around the ring in the (clockwise) direction of increasing processor identifiers. The tail set for \( p \) consists of every \( \lceil |S|/|S| \rceil \)th processor among the first \( \lceil |S|/2 \rceil \) processors that lie clockwise of \( p \) on the ring and are not included in the head set for \( p \). The quorum SHORTRING\((S)\) is the union of the head and tail sets for \( p \). An example quorum obtained for processor \( f \in S \) when \( |S| = 15 \) is shown in Figure 5(a). Let \( S[\cdot] \) refer to the ordered list of elements in \( S \) sorted by increasing processor identifier. Let \( \hat{S}(p) \) be the index value that indicates \( p \)’s position in the list \( S[\cdot] \), such that \( p \) is the \( \hat{S}(p) \)th element of \( S[\cdot] \). Given that all math is modulo \( |S[\cdot]| \), let the set \( Arc(i,j) \) define the set of \( j \) neighbors in the ring, starting with \( i \) and moving clockwise: \( \{ i, S[(\hat{S}(i)+1)],...,S[(\hat{S}(i)+j-1)] \} \).
head^p \quad \text{tail}^p

\text{SHORTRING}(S): Q^p

head^q \quad \text{tail}^q

\text{SHORTRING}(S): Q^q

q > p

0 \leq \$q - \$p < \lfloor \sqrt{|S|} \rfloor

\lfloor \sqrt{|S|} \rfloor \leq \$q - \$p < \lfloor |S|/2 \rfloor

\lfloor |S|/2 \rfloor \leq \$q - \$p < |S|

Figure 1: Proof of SHORTRING and MEMRING Correctness.

Then the quorum for processor \( p \in S \) is computed as

\[\text{head}^p = \text{Arc}(p, \lfloor \sqrt{|S|} \rfloor)\]

(1)

\[\text{tail}^p = \{S[\$p] + 2\lfloor \sqrt{|S|} \rfloor - 1, S[\$p] + 3\lfloor \sqrt{|S|} \rfloor - 1\},\]

(2)

\[\ldots S[\$p] + (\lfloor |S|/(2\lfloor \sqrt{|S|} \rfloor) \rfloor + 1)\lfloor \sqrt{|S|} \rfloor - 1\}

\text{SHORTRING(S): } Q^p = \text{head}^p \cup \text{tail}^p

(3)

The proof that SHORTRING computes intersecting sets of quorums can be given as a simple counting argument similar to the RING proof given by Lien and Yuan in [15]. The proof is given here in a graphically intuitive manner.

**Theorem 1.** The quorums computed by SHORTRING for any \( S \) satisfy the quorum intersection property.

**Proof:** Let \( \text{start}[^{\text{tail}}] \) be the processor that immediately follows the last processor included in \( \text{head}^p \), such that \( \text{start}[^{\text{tail}}] = S[\$p] + \lfloor \sqrt{|S|} \rfloor \). For any two processors \( p \) and \( q \) such that \( q > p \), either the head sets of \( p \) and \( q \) intersect, as shown in Case I of Figure 1; or \( \text{head}^q \) is a subset of \( \text{Arc}(\text{start}[^{\text{tail}}]), \lfloor |S|/2 \rfloor \), as shown in Case II of Figure 1; or else \( \text{head}^q \) is a subset of \( \text{Arc}(\text{start}[^{\text{tail}}]), \lfloor |S|/2 \rfloor \), as shown in Case III of Figure 1. Without loss of generality, if \( \text{head}^q \) is a subset of \( \text{Arc}(\text{start}[^{\text{tail}}]), \lfloor |S|/2 \rfloor \), then the quorums for \( p \) and \( q \) must intersect since by equation (2), there can exist no processor \( i \in S \) such that \( \text{Arc}(i, \lfloor \sqrt{|S|} \rfloor) \) is a subset of \( \text{Arc}(\text{start}[^{\text{tail}}]), \lfloor |S|/2 \rfloor \) and the two sets \( \text{Arc}(\text{start}[^{\text{tail}}]), \lfloor |S|/2 \rfloor \) and \( \text{Arc}(i, \lfloor \sqrt{|S|} \rfloor) \) have an empty intersection.

**2.2 The MEMRING Function**

MEMRING is an extension of SHORTRING that is designed to reduce change sensitivity. MEMRING incorporates a hysteresis heuristic that uses the current value of \( S \) in addition to the most recently computed quorum assignment in attempting to reduce the needed number of modifications to quorum data structures. Like RING and SHORTRING, MEMRING maps the processors in \( S \) to the nodes of a logical ring. The head set for \( p \) is the same head set computed by SHORTRING. Then the tail set of the quorum computed for \( p \) is defined to be any subset of \( S \) such that

\[|\text{tail}^p| \leq \left\lfloor \frac{|S|}{(2\lfloor \sqrt{|S|} \rfloor)} \right\rfloor + g, \quad g \geq 0\]

(4)

\[\text{TAILSPAN}^p = \text{Arc}(S[\$p] + \lfloor \sqrt{|S|} \rfloor, \lfloor |S|/2 \rfloor)\]

(5)

\[\text{tail}^p \subseteq \text{TAILSPAN}^p\]

(6)

\[\forall i \in S \quad (\text{Arc}(i, \lfloor \sqrt{|S|} \rfloor) \subseteq \text{TAILSPAN}^p)\]

(7)

The constant \( g \geq 0 \) in equation (4) is a quorum size growth constant whose significance is explained later. Equations (5) and (6) state that \( \text{tail}^p \) contains only processors among the first \( \lfloor |S|/2 \rfloor \) that follow those in \( \text{head}^p \). By equation (7), \( \text{tail}^p \) must be chosen from the processors in the ring such that \( \text{TAILSPAN}^p \) does not contain any set of \( \lfloor \sqrt{|S|} \rfloor \) neighboring processors that are not included in \( \text{tail}^p \). The tail set construction for MEMRING is a generalization of the tail set construction for SHORTRING. The quorum of \( p \) can be formed using any set \( \text{tail}^p \) among a number of legitimate tail sets that satisfy equations (4) through (7).

Figure 2: Number of possible MEMRING tail sets.
equals zero and only one tail set is possible. Among the set of legitimate tail sets for the quorum of processor \( p \), some tail sets will be more similar than others to the previous quorum computed for \( p \). When invoked with the current value of \( S \), MEMRING seeks to identify a new tail set for \( p \) that is similar to \( p \)'s previous quorum, by trying to include in \( \text{tail}^p \) processors that were previously included in \( \text{MEMRING}(S_{\langle j \rangle}):Q^p \). The MEMRING algorithm is formally specified in Figure 4. The algorithm is invoked with parameters \( S[\] and \( g \). Line 21 commits the new quorum computed for \( p \) to local storage, so that it can be accessed in line 13 the next time the MEMRING function is invoked.

The change sensitivity of MEMRING is reduced if quorums are allowed to grow slightly larger than the minimum size, by selecting a value for \( g \) that is greater than zero. For \( g > 0 \), the number of choices for \( \text{tail}^p \) increases significantly, making it easier for MEMRING to identify a tail set that contains a larger number of the processors previously included in \( p \)'s quorum. The expected cost of larger \( g \) is an increase in the average quorum size and increased variance in the load of different processors. If MEMRING is invoked with \( g = 0 \), quorums are computed that have minimum and uniform size, although the processor load values may not remain uniform as MEMRING is repeatedly invoked. The value chosen for \( g \) determines the magnitude of the trade-off between reduced change sensitivity and increased quorum size and load values.

A quorum adaptation example is given in Figure 5. Consider the quorum represented by the shaded processors in 5(a) and 5(d) to be the quorum initially computed for processor \( p \) by the SHORTRING and MEMRING functions for \( |S| = 15 \). Figure 5(b) shows the quorum subsequently

**Figure 3:** Example: multiple tail set choices for MEMRING.

**Figure 4:** MEMRING algorithm.
computed for \( f \) by SHORTRING after one processor \( p \) is added to \( S \). Figure 5(e) shows the quorum computed for \( f \) by MEMRING (\( g = 0 \)) after processor \( p \) is added to \( S \). Likewise, Figures 5(c) and 5(d) show the quorums for \( f \) computed by SHORTRING and MEMRING respectively after one processor \( j \) is removed from \( S \). A comparison of the quorums for \( f \) before and after the given changes to \( S \) indicates that MEMRING requires fewer adjustments to \( f \)'s quorum than SHORTRING requires. When processor \( p \) is added to \( S \), SHORTRING requires that three processors \( \{i,m,a\} \) are added to \( f \)'s quorum, while MEMRING requires only the addition of one processor \( i \). SHORTRING requires that two processors, \( k \) and \( n \), be removed from \( f \)'s quorum; MEMRING requires the removal of no processors. After the removal of \( j \) from \( S \), MEMRING requires no changes at all to \( f \)'s quorum, while SHORTRING requires the removal of \( \{k,n\} \) and the addition of \( \{l,o\} \) to \( f \)'s quorum. The experimental results given in the following section indicate that MEMRING consistently requires fewer changes to quorums than SHORTRING requires.

In a decentralized implementation of generalized OQA, every processor executes MEMRING locally. In such a case, it must be considered that quorums computed by MEMRING are a function of the order in which the processor locally observes changes that occur to \( S \). Allowing divergent local beliefs about the order of changes that occur to \( S \) is desirable in a distributed system, since the alternative requires an expensive synchronization protocol. To ensure liveness, it is sufficient that all processors eventually agree on the value of \( S \).\(^1\) However, even after agreement is reached, the quorum mappings computed by MEMRING at different processors may not agree, if all processors have not observed a total order on the changes to \( S \). We state the correctness of MEMRING in a manner that makes clear its correctness in a decentralized implementation without agreement on a total ordering of changes to \( S \). Although processors may never reach exact agreement on the quorum mapping, the quorum intersection property is dependent only on the cardinality of the set of processors in \( S \), and not on the previous history of changes to \( S \) or on the value of \( g \) used in the computations.\(^3\)

**Theorem 2.** Let \( \text{history}^p(S) \equiv \{S[p,0], S[p,1], ... \} \) represent \( p \)'s local view of the sequence of values taken by \( S \), and let \( \text{MEMRING}(K,g,p) \) represent the result of \( p \)'s invocation of MEMRING when \( p \)'s local view of \( S \) is \( S[p,t] \). Given any set of processors \( K \), any set of histories \( H = \{\text{history}^p(S) \mid x \in K\} \), and any set of constants \( \{T^x \mid x \in K\} \) such that for each \( \text{history}^p(S) \in H \), we have \( \text{history}^p(S) = \{...,S[p, T_p], ...\} \) and \( S[p, T_p] = K \); then for any \( g_p, g_{p'} \), \( T_p, T_{p'} \) and \( p, q, j, k \in K \):

\[
\text{MEMRING}(K,g,p)[p', T_p; Q' \cap \text{MEMRING}(K,g,p') \mid q', t_{q'}; Q' \neq \emptyset].
\]

Theorem 2 states that no matter when in the sequence of \( p \)'s local computations \( \text{MEMRING}(K,g,p) \) is computed, any quorum returned by invocation of \( \text{MEMRING}(K,g,p) \) intersects each other quorum returned by \( \text{MEMRING}(K,g,p') \), and also intersects any quorum computed at any other processor that also uses \( K \) as input to MEMRING. Despite the greater complexity of MEMRING, the proof of Theorem 2 follows the same pattern as the proof of Theorem 1 for SHORTRING. Any two quorums computed for processors \( j,k \in K \) must contain a common processor since \( |\text{head}| = |\text{head}| = |\sqrt{|K|}| \) and, by equations (4) through (7), neither TAILSPAN\(^j\) nor TAILSPAN\(^k\) can contain \( |\sqrt{|K|}| \) neighboring processors that are not contained in tail\(^j\) and tail\(^k\) respectively.

3. **Experimental Evaluation of Change Sensitivity**

To measure the change sensitivity of MEMRING and other quorum mapping functions, two metrics for change sensitivity are given. The first metric is based on the cardinality of the minimum set of changes that must be made to quo-

---

\(^1\)See the quorum adaptation algorithm in [6] that allows processors to have temporarily divergent and partially incorrect beliefs about the value of \( S \), but ensures safety and liveness given that the processors will eventually reach agreement on the actual value of \( S \).
rum data structures when a change occurs to $S$. We consider simple atomic events that remove (add) some processor from (to) some quorum or $S$. Let Grow be a generic step that increases by one the cardinality of exactly one processor’s quorum. Let Shrink be a step that decreases by one the cardinality of a single processor’s quorum. As an example, if $M(S_1):Q_P = \{a,b,c\}$ and $M(S_{i+1}):Q_P = \{a,b,d,e\}$, then at least one Shrink step is needed to remove processor $c$ from $p_i$’s quorum, and at least two Grow steps are required to add $d$ and $e$ to $p_i$’s quorum. A quorum mapping function that requires relatively many Grow and Shrink steps when a small change occurs to $S$ has a high degree of change sensitivity. The change sensitivity function defined below indicates the ratio of required Grow and Shrink steps to the cardinality of changes to $S$.

**Definition** (change sensitivity function $\delta$): Given a change in system configuration from $S = S'$ to $S = S''$ and dynamic quorum assignment values $M(S')$ and $M(S'')$. The function $\delta(M, S', S'')$ is defined by

$$\delta(M, S', S'') = \sum_{p \in S} \left( \frac{M(S'):Q_P \oplus M(S''):Q_P}{S' \oplus S'' \setminus [M(S'):Q_P]} \right)$$  \hspace{1cm} (8)

The notation $A \oplus B$ refers to the disjunction of sets $A$ and $B$, which is the minimum number of processors that must be added to or removed from $A$ to obtain $B$. The expression $[M(S'):Q_P]$ appears in the divisor of the summation expression as a normalization factor, to make the value of the change sensitivity function independent of the average quorum size computed by $M$. Let $p(S' \rightarrow S'')$ be the probability that a change from $S = S'$ to $S = S''$ occurs, according to some probability distribution $\Pi$.

**Definition** (expected change sensitivity $E_\delta$): Function $M$’s expected change sensitivity, denoted $E_\delta(M, \Pi)$, is given thus:

$$E_\delta(M, \Pi) = \sum_{S', S''} \left( \delta(M, S', S'') \cdot p(S' \rightarrow S'') \right)$$  \hspace{1cm} (9)

Consider the operation of an ideal quorum mapping function $M_{\text{perfect}}$ that has minimum change sensitivity. It would be expected that $M_{\text{perfect}}$ tends to remove from quorums only the processors that are removed from $S$, and tends to add to quorums the minimum number of processors that ensures every pair of quorums keeps a nonempty intersection. The value of $E_\delta(M_{\text{perfect}}, \Pi)$ would take a value near zero. Consider the possible values taken by $E_\delta$ for the class of balanced quorum mapping functions. By inspection of equation (8) the value of $\delta$ falls in the range $(0, 2((|S'| + 1)/|S'|))$ for any adaptation of the quorum assignment. The highest expected value for $\delta$, approximately two, corresponds to the most inefficient quorum adaptation, in which only one processor is added to $S'$ and every processor’s new quorum $M(S''):Q_P$ and old quorum $M(S'):Q_P$ are disjoint sets.

Intuitively, the $E_\delta$ metric predicts the number of read and write operations that must be performed on the data structures that identify quorums, but it provides little information about the costs of coordinating the changes to data structures. A second change sensitivity metric is given that predicts coordination cost. The expected message cost for $M$ is determined as follows, given the assumption of a particular decentralized implementation of generalized OQA. The metric uses the predicted costs of the decentralized algorithm for generalized OQA, called QADAPT, that is given in [6]. QADAPT enables processors to exchange messages that are used to correctly order the changes to quorum data structures, ensuring that the quorum intersection property is not violated during quorum adaptation. QADAPT predicts the minimum cost of decentralized quorum adaptation, since the number of messages required by QADAPT is shown analytically in [5,6] to be close to a proven lower bound on the number of messages required for any decentralized quorum adaptation algorithm. In equation (10), QADAPT_COST($S' \rightarrow S''$) is the theoretical minimum number of unordered point-to-point interprocessor messages required by execution of QADAPT in order to replace quorum assignment $M(S')$ with $M(S'')$.

**Definition** (expected message cost $E_M$): The expected message passing cost of quorum adaptation using $M$ is

$$E_M(M, \Pi) = \sum_{S', S''} \left( \text{QADAPT}_\text{COST}(S' \rightarrow S'') \cdot p(S' \rightarrow S'') \right)$$  \hspace{1cm} (10)

The expected value for QADAPT_COST is a function of the change sensitivity of $M$ and also of the sizes of quorums computed by $M$. These two factors determine the number and distribution of necessary Grow and Shrink steps. The value for QADAPT_COST is computed using a simulation of QADAPT execution. Experiments described in [6] using an implementation of QADAPT in a network of workstations confirm that the simulation of QADAPT correctly predicts actual message passing costs of QADAPT execution. Thus the $E_M$ metric gives a meaningful approximation of the real communication cost of decentralized quorum adaptation using function $M$.

### 3.1 Other Balanced Quorum Mapping Functions

Change sensitivity for the RING, SHORTRING and MEMRING functions is compared to that of the following two quorum mapping functions that have quorum size and load values similar to the values for SHORTRING and MEMRING, that is, between $\sqrt{|S|}$ and $1.5 \sqrt{|S|}$.

#### 3.1.1 STR Function

In [2] Agrawal and Jalote give a mapping function called STR that generates the $|S|$ subsets of $S$ given by all combinations of two processors from the set of processor identifiers $(1,...,H)$, where $|S| = H(H-1)/2$. A one to one mapping is performed from the set of processors in $S$ to the set of combinations generated. If a processor $p$ is mapped into the combination $(i,j)$, it is included in the $i$th and $j$th quorums and in no other quorum. If $i + j$ is odd, then max$(i,j)$ is the quorum computed for $p$, otherwise min$(i,j)$ is the quorum computed for $p$.

#### 3.1.2 CYCLIC Function

Luk and Wong give in [17] a quorum mapping function adapted from cyclic difference set theory. The quorum computed for each $p \in S$ contains
the processors \( \{p, p + a_1, p + a_2, \ldots, p + a_k\} \) modulo \( |S| \), where \( \{a_1, a_2, \ldots, a_k\} \) is a relaxed cyclic \( (|S|,k) \)-difference set. A table containing relaxed cyclic difference sets determined by exhaustive search is given in [17], that can be used to compute quorum assignments for any \( S \) such that \( 4 \leq |S| \leq 111 \).

### 3.2 Change Sensitivity Measurements

This section gives sensitivity measurements for five quorum mapping functions, based on simulations of changes to \( S \). The following probability distribution applies to the simulated changes to \( S \).

**Definition** (probability distribution \( \Pi \)): Let \( \Pi_{x,d} \) indicate a probability distribution for changes that occur to \( S \), such that

\[
p(S' \rightarrow S^{'*}) = \Phi_{x,d} \quad \text{if } |S'| = x \quad \text{and} \quad |S^{'*}| = x + d
\]

and \( \Phi_{x,d} \) is a function of \( x \) and \( d \).

If \( d \) is positive, \( \Pi_{x,d} \) represents that the only possible changes to \( S \) are those that occur when \( |S| = x \), such that exactly \( d \) processors are added to \( S \). All such events are equally likely, occurring with probability \( \Phi_{x,d} > 0 \). If \( d \) is negative, then \( \Pi_{x,d} \) represents that the only possible changes to \( S \) occur when \( |S| = x \) and exactly \( d \) processors are removed from \( S \). In each simulation, changes to \( S \) were selected randomly, in accordance with the indicated probability distributions. The metrics given in equations (8) and (9) were evaluated for each simulated change to \( S \) and the average values are presented in the plots given below. For reference, the quorum size and load for each quorum function are given in Figure 6, as a function of the size of \( S \).

### 3.3 Expected Sensitivity \( E_\delta \)

Figure 7 shows expected change sensitivity as a function of the size of \( S \), when only single processor removals from \( S \) are considered. Measurements were collected with two values of parameter \( g \) for MEMRING, \( g = 0 \) and \( g = 1 \). The plots in Figure 8 show the expected change sensitivity when single processors are added at random to the system. The sensitivity for the STR and CYCLIC functions is higher than the sensitivity for any of the logical-ring-based functions, including Lien and Yuan’s original RING function. For most values of \( |S| \), MEMRING has significantly lower change sensitivity than each of the other functions, as expected. Simulations were also carried out for other cases \( \Pi_{x,d} \), \( |d| > 1 \). The resulting sensitivity plots (not shown) are similar to those given in Figures 7 and 8.

The plots in Figure 9 give a more detailed look at the change sensitivity measurements for \( 22 \leq |S| \leq 25 \). The measured values for \( \delta \) are given for an arbitrarily selected sequence of 75 simulated changes to \( S \), applied using each of the five functions. Each change to \( S \) is either the removal of one processor, indicated by an “F” on the timeline below the graphs, or the addition of one processor, indicated by an “R” below the graphs.

### 3.4 Expected Quorum Adaptation Message Cost

Figure 10 gives the expected message cost for different system sizes, when a single processor is removed from \( S \). Expected message cost for all functions other than MEMRING is \( O(|S|) \), while expected message cost for MEMRING is \( O(1) \) for \( \Pi_{x,1} \), except for the “spikes” that can be observed for each \( x \) that is an integral square. (A linear increase in \( g \) as \( |S| \) increases will eliminate the spikes. It was observed that using \( g = \lceil |S|/20 \rceil \) instead of holding \( g \).
constant results in a plot almost identical to Figure 10, but without the spikes.) Although the change sensitivity for RING is shown in Figure 7 to be less than the sensitivity for STR or CYCLIC, the expected message costs of these three functions are about the same. This can be explained by noting that RING's lower change sensitivity is offset by its larger computed quorum size (cf. Figure 6). Although the quorum size for MEMRING increases with larger $S$, the trend for the expected message cost of MEMRING is not increasing for larger $S$, because change sensitivity for MEMRING is decreasing for larger $S$ (cf. Figures 7 and 8), offsetting the effect increasing quorum size has on message cost.

3.5 Effect of Varying Parameter $g$

The price that is paid for the reduced sensitivity of MEMRING is slightly larger quorum size and load. The bottom plot in Figure 11 show the effects of choosing different values for $g$ (see equation 4) on the operation of MEMRING. The plot that shows the values computed for $\delta$ was generated using the same sequence of 75 changes to $S$ shown in Figure 9. Sensitivity of the SHORTRING function is also shown. The greatest incremental reduction in change sensitivity occurs when $g = 1$; selecting $g > 1$ achieves little additional reduction in change sensitivity for $22 \leq |S| \leq 25$. Figure 11 also shows the evolution of the average quorum size in each system, and the maximum quorum load for any processor. For $g > 0$, as the MEMRING function is repeatedly executed the average quorum size grows slightly larger than the minimum quorum size achieved for $g = 0$. The result of selecting $g > 0$ is a dynamic quorum assignment that is on average less balanced, with greater variation in the load of different processors.

4. Summary and Discussion

This research considers the generalized on-line quorum adaptation method, that allows remapping of processors to
quorums in order to increase availability or performance. An advantage of generalized OQA is the ability to select any quorum mapping function that delivers desirable trade-offs between quorum size, load, availability, or other traditional quorum system metrics. This work considers, in addition to the traditional metrics, the costs of coordinating changes to distributed data structures that identify the quorum assignment. The concept of change sensitivity is introduced, and new change sensitivity metrics are given for comparing the expected change sensitivity and communication costs of different quorum mapping functions. A new quorum mapping function called MEMRING is given that reduces change sensitivity in return for a small increase in quorum size and load. MEMRING exhibits hysteresis, computing a quorum assignment that is based not only on current system state but also on the previously installed quorum assignment. This allows MEMRING to reduce the needed modifications to distributed quorum data structures, and to consequently reduce the amount of communication needed for installing a new quorum assignment in a decentralized system. Simulation-based measurements are given to show the expected performance of MEMRING. The change sensitivity and expected message costs of MEMRING are shown to be less than the costs for four other quorum mapping functions that compute quorums with comparable size and load.

The experimentation illustrates the importance of the dynamic change sensitivity metrics for selecting a quorum mapping function. The two previously designed quorum functions STR and CYCLIC considered in Section 3 both compute balanced quorum assignments with similar quorum size, load, and availability. MEMRING also has size, load, and availability similar to STR and CYCLIC; yet the expected dynamic costs of MEMRING are significantly less than the expected costs for STR or CYCLIC. Without an analysis of dynamic costs using metrics like those given in this work, it would be difficult to determine what quorum mapping function should be used in an implementation of generalized OQA. The idea of using hysteresis to reduce change sensitivity of dynamic quorum assignments should be applicable to other quorum mapping functions. Further research includes investigation of the dynamic costs of a wide variety of previously derived quorum map-

Figure 10: Expected message cost, single processor removed from $S$.

Figure 11: MEMRING and SHORTRING quorum size, load, and sensitivity, $|S|=25$
ping functions to determine their inherent change sensitivity, including functions that do not compute balanced quorum assignments.

References


