Randomized Sets and Multisets

Hyunyoung Lee[*]
Department of Computer Science
Unversity of Denver, 2360 S. Gaylord St.
Denver, CO 80208, USA
hlee@cs.du.edu

Andreas Klappenecker
Department of Computer Science
Texas A&M University
TX 77843-3112, TX, USA
klappi@cs.tamu.edu

Abstract

Sets and multisets are fundamental data structures which are used in numerous distributed algorithms. We investigate randomized versions of sets and multisets and show that our probabilistic approach can improve the performance of systems using our distributed shared data structures. We give algorithms for basic set and multiset operations and prove a number of fundamental properties. We illustrate the use of randomized multisets in a location tracking application of mobile ad hoc networks.

1 Introduction

Recall that a *multiset* is an unordered collection of elements that can occur multiple times; whereas a *set* is an unordered collection of distinct elements. Many algorithms use these basic data structures. In fact, they are so common that many programming languages support sets and multisets either directly or through standard libraries (for example, the STL of C++).

This paper is concerned with distributed shared data structures for sets and multisets. Assume that we have a set of n servers that we number, for simplicity, from 1 to n. We represent a set (or multiset) M by replicas M_r on server r for $r \in \{1, \ldots, n\}$. To give an idea how this is traditionally done, let's have a look at a small example with five servers:

$$\begin{array}{ll}
\text{Server } 1: & M_1 = \{\text{⚀}, \text{⚄}\}, \\
\text{Server } 2: & M_2 = \{\text{⚁}, \text{⚄}, \text{⚅}\}, \\
\text{Server } 3: & M_3 = \{\text{⚀}, \text{⚅}\}, \\
\text{Server } 4: & M_4 = \{\text{⚁}, \text{⚄}, \text{⚅}\}, \\
\text{Server } 5: & M_5 = \{\text{⚀}, \text{⚁}\}.
\end{array}$$

The five replicas represent the set

$$M = \bigcup_{r=1}^{5} M_r = \{\text{⚀}, \text{⚁}, \text{⚄}, \text{⚅}\}.$$

You will notice that each element of M is contained in exactly three replicas. This means that whenever a client process requests the replicas from 3 servers, then the client can reconstruct M by taking the union of the three sets. This scheme is based on a traditional quorum system where each client process is supposed to accesses $k > n/2$ replica servers to read a set or write an element.

The drawback of this scheme is that it is fairly rigid and it might even be undesirable for a client process to contact such a large number of servers. Imagine for instance a mobile ad-hoc network, where some servers are not available at all times. A large quorum Q of $k > n/2$ replica servers is likely to contain some servers that are not reachable, and this can be a considerable bottleneck.

A possible remedy was suggested by Malkhi, Reiter, Wool, and Wright [7], who proposed to replace traditional quorum systems by probabilistic quorum systems. In this case, the clients select quorum of, say, size $k = \Omega(n^{1/2+\varepsilon})$. Clearly, we can find two quorums that are disjoint. However,

[*]Contact author. Phone (303) 871–7732. Fax (303) 871–3010.

if the clients choose the two quorums uniformly at random, then the two quorums are likely to share at least one server thanks to the birthday paradox.

A concrete numerical example might help the reader to appreciate this property. Suppose that we have $n = 50$ servers. A traditional quorum system chooses quorums of size $k \geq 26$. If we assume that the clients select merely $k = 16$ servers per quorum, then two quorums that are chosen uniformly at random will be disjoint with probability less than $\varepsilon < 0.000448$.

We explore the possibility to build *randomized* set and multiset operations using probabilistic quorum constructions. An advantage of the smaller quorum size is a significantly reduced message complexity – a client has to communicate with considerably fewer servers to execute the operations. On the downside, all operations on randomized sets or multisets involve a certain (controllable) amount of error ε. The situation is comparable with floating point or fixed-point arithmetic where the operations also introduce inaccuracies.

We introduce in Section 2 read and write operations for randomized sets and multisets and analyze the behavior of these operations. We derive in Section 3 other set and multiset operations that are mostly based on these primitives. We find that the price that we have to pay for the significantly reduced quorum sizes is a small loss of values in most primitive operations. In Section 4, we give an application of randomized multisets to the mobile location tracking problem.

Notations. If n is a positive integer, then we denote by $[n]$ the set $\{1, 2, \ldots, n\}$. The family of all k-subsets of the set $[n]$ is denoted by $\binom{[n]}{k}$.

2 Read and Write Operations

We describe in this section the most fundamental operations to create, write, and read a randomized set or multiset. We analyze several key properties of these operations; in particular, we determine the expected size of the set returned by the read operation. The first operation concerns the genesis of sets:

• **Create.** We can create a set or multiset M by

$$\texttt{create}(M).$$

This operation creates the replica $M_i = \emptyset$ on each server i in the range $1 \leq i \leq n$.

The read and write operations have the following behavior:

• **Add.** The write operation

$$\texttt{add}(\texttt{x}, M)$$

chooses uniformly at random a k-subset W of $[n] := \{1, \ldots, n\}$ and adds the element $\texttt{x}$ to the replicas M_w for all w in W. In other words, the operation has the effect

$$M_w := M_w \cup \{x\} \quad \text{for all } w \text{ in } W.$$

In other words, the multiplicity of $\texttt{x}$ is increased by one in the case of multisets; and the replica contains $\texttt{x}$ in the case of sets.

• **Read.** Similarly, the read operation

$$\texttt{read}(M)$$

chooses uniformly at random a k-subset R of $[n]$ and returns the union of the replicas M_r with r in R; in other words, the read operation returns the set $\bigcup_{r \in R} M_r$.

The replicas M_r with $1 \leq r \leq n$ represent a set M if and only if $M = \bigcup_{r \in [n]} M_r$.

Lemma 1. *Suppose that we choose two quorums R and W in $\binom{[n]}{k}$ uniformly at random. Then*

$$\Pr[R \cap W = \emptyset] = \binom{n-k}{k} \Big/ \binom{n}{k}.$$

In the terminology of Malkhi, Reiter, Wool and Wright [7], our operations are based on a probabilistic ε-intersecting quorum system $\mathcal{Q}_k = \binom{[n]}{k}$ of all k-subsets of the set $[n] = \{1, \ldots, n\}$ with a uniform access strategy. The previous lemma simply records the fact that two quorums in $\mathcal{Q}_k$ intersect with probability $1 - \varepsilon$, where $\varepsilon = \binom{n-k}{k}\binom{n}{k}^{-1}$. If we are conservative and choose

$k \geq n/2$, then we recover a traditional, deterministic quorum system in which any two quorums intersect. An advantage of probabilistic quorum systems is that one can choose considerably smaller quorum sizes [7]. In our application, the benefit is a considerably reduced message complexity to access the distributed data structures.

If we choose quorums of size $k < n/2$, then a read quorum can fail to intersect with a previous write quorum, and thus a set returned by the read operation might have smaller cardinality than it should have. Fortunately, it is possible to choose the quorum size k such that the probability of such an undesired event becomes neglibile.

Let us first investigate the consequences of a particular quorum size selection before we make recommendations on the choice of k. Clearly, a crucial figure is the expected size of the set returned by a read operation.

Proposition 2. *Suppose that the read and write quorums use the quorum system $\mathcal{Q}_k = \binom{[n]}{k}$ with uniform access probability. If the replica sets represent a set M of cardinality $m = |M|$, then the expected value of the size X of the set returned by* `read(M)` *is at least*

$$\mathrm{E}[X] \geq m(1-\varepsilon) \quad \textit{with} \quad \varepsilon = \binom{n-k}{k} \Big/ \binom{n}{k}.$$

Equality holds if and only if each element of M is represented by exactly k replicas.

Proof. Denote by M_r the replica at server r, where $1 \leq r \leq n$. It is clear from the definition of a write operation that an element x of M is contained in at least k replicas. It follows that the set $W_x = \{ r \,|\, 1 \leq r \leq n, x \in M_r \}$ has cardinality $s_x = |W_x| \geq k$. Let R denote a read quorum of size k and denote by Y_x the indicator random variable for the event $W_x \cap R \neq \emptyset$. Then the probability $\Pr[Y_k = 1] = \Pr[W_x \cap R \neq \emptyset]$ is given by

$$\Pr[Y_x = 1] = 1 - \binom{n-s_x}{k} \Big/ \binom{n}{k} \geq 1 - \varepsilon. \quad (1)$$

We have $X = \sum_{x \in M} Y_x$. By linearity of expectation, we obtain

$$\mathrm{E}[X] = \mathrm{E}\Big[\sum_{x \in M} Y_x\Big] = \sum_{x \in M} \Pr[Y_x = 1] \geq m(1-\varepsilon),$$

which proves our claim. $\square$

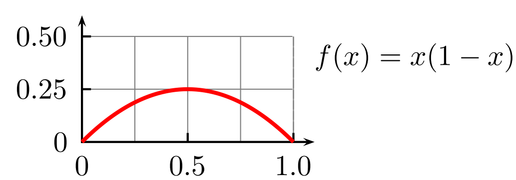
We would like to choose the quorum size k such that $\mathrm{E}[X]$ is close to the cardinality $|M|$ and such that it is unlikely that a particular read operation returns a set of size X that deviates much from the expected value $\mathrm{E}[X]$. Let us first derive a bound on $\mathrm{var}[X]$.

Lemma 3. *We keep the notation of the previous proposition. If the quorum size k is chosen*[1] *such that $\varepsilon \leq 1/2$, then $\mathrm{var}[X] \leq m\varepsilon(1-\varepsilon)$.*

Proof. Recall that X is the sum of independent random variables Y_k; therefore, the variance sum theorem yields

$$\mathrm{var}[X] = \mathrm{var}[Y_1] + \mathrm{var}[Y_2] + \cdots + \mathrm{var}[Y_m]. \quad (2)$$

The random variable Y_k is a Bernoulli random variable with probability of success given in (1). Recall that the function $x(1-x)$ is monotonically descreasing on the interval $[1/2, 1]$, as is illustrated by the function plot:



Let $p_k := \Pr[Y_k = 1]$. Then $p_k \geq 1 - \varepsilon \geq 1/2$, we obtain that the variance $\mathrm{var}[Y_k] = p_k(1-p_k)$ is bounded from above by

$$\mathrm{var}[Y_k] = f(p_k) \leq f(1-\varepsilon) = \varepsilon(1-\varepsilon).$$

Therefore, we can conclude from equation (2) that $\mathrm{var}[X] \leq m\varepsilon(1-\varepsilon)$, as claimed. $\square$

[1] Actually, we will of course choose k such that $\varepsilon \ll 1/2$.

Tail Estimates. We have shown that the size X of a set returned by the read operation `read(M)` has expectation value $\mathrm{E}[X] \geq m(1-\varepsilon)$. We can select the quorum size k such that ε becomes as small as we please, as we will show in the next paragraph. We will demonstrate now that it is unlikely that the value X deviates much from $\mathrm{E}[X]$.

Lemma 4. *If X denotes the size of a set returned by `read(M)` and m is the cardinality of M, then*

$$\Pr\big[\,|X-\mathrm{E}[X]| \geq \varepsilon^{1/2}\,\mathrm{E}[X]\,\big] \leq \frac{1}{(1-\varepsilon)m}.$$

Proof. Recall that Chebychev's inequality states that the probability $\Pr[\,|X - \mathrm{E}[X]| \geq \lambda]$ is at most $\mathrm{var}[X]/\lambda^2$. Thus, if we set $\lambda = \varepsilon^{1/2}\,\mathrm{E}[X]$, then we obtain the claim with the help of Proposition 2 and Lemma 3. □

Lemma 5. *If X denotes the size of a set returned by `read(M)` and the cardinality of M is m, then*

$$\Pr\big[\,|X-\mathrm{E}[X]| > d\big] < 2e^{-2d^2/m}.$$

In particular, if we set $d = \varepsilon^{1/2}\,\mathrm{E}[X]$, then

$$\Pr\big[\,|X-\mathrm{E}[X]| > \varepsilon^{1/2}\,\mathrm{E}[X]\big] < 2e^{-2\varepsilon(1-\varepsilon)^2 m}.$$

Proof. Recall that the random variable X is given by the sum of indicator random variables $X = \sum_{x\in M} Y_x$. Define new random variables $Z_x := Y_x - p_x$ with $p_x = \Pr[Y_x = 1]$ for all $x \in M$. We have $\Pr[Z_x = 1 - p_x] = p_x$ and $\Pr[Z_x = -p_x] = 1 - p_x$. The main point of this definition is that the sum of the random variables Z_x is given by $X - \mathrm{E}[X] = \sum_{x\in M} Z_x$, and we can estimate the tails of the right hand side by

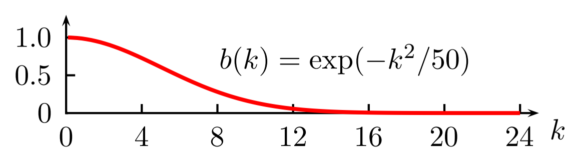
$$\Pr[|X-\mathrm{E}[X]| > d] = \Pr[|\sum_{x\in M} Z_x| > d] \leq 2e^{-2d^2/m},$$

where the last inequality is a bound of Chernoff, see [2] or [1, Lemma A.4]. □

Quorum Size. Suppose that we have n servers and select a quorum of size $k < n/2$, then we can estimate

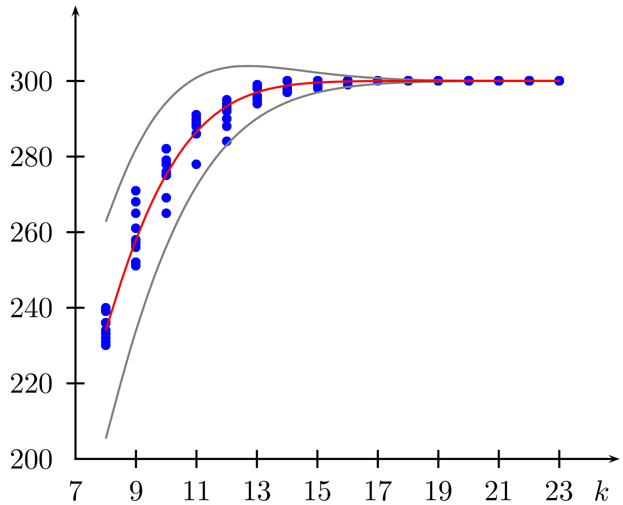
$$\varepsilon = \binom{n-k}{k}\binom{n}{k}^{-1} \leq e^{-k^2/n},$$

see, for instance, Jukna [3, p. 21]. Therefore, the probability $\varepsilon = \varepsilon(k)$ that two quorums of size k fail to intersect *decreases rapidly*, especially for quorum sizes larger than $n^{1/2}$. The next figure illustrates the bound for $n = 50$ servers and a range of quorum sizes:



Recall that the size X of a set returned by a read operation `read(`M`)` is the sum of m independent random variables $X = Y_1 + \cdots + Y_m$ provided that the set M has cardinality m. If m is large, then the central limit theorem tells us that we can approximate X by the normal distribution $N(\mathrm{E}[X], \mathrm{var}[X])$, see [6]. Thus, as a rule of thumb, the value X will lie within the range $m(1-\varepsilon) \pm 4\sqrt{m\varepsilon(1-\varepsilon)}$ about 99.99% of the time.

We bolster this claim by giving some experimental results of a randomized set with $m = 300$ elements which is realized with $n = 50$ replica servers. We plot the quorum size k in the range $8 \leq k \leq 23$ against the number of elements that have been returned by a read operation. The solid red curve shows the expected number of elements $300(1-\varepsilon)$, with $\varepsilon = \binom{50-k}{k}/\binom{50}{k}$, that are returned by a read when the quorum size is k. The gray curves show $300(1-\varepsilon) \pm 4\sqrt{300\varepsilon(1-\varepsilon)}$. The blue dots represent the measured number of elements. For each quorum size k we have repeated the experiment 10 times.



The graph illustrates that for a quorum size of $k \geq 17$ all 300 elements were returned in each run. Moreover, it illustrates that all experimental results are within the region that we predicted with our theoretical considerations.

3 Further Operations

We describe in this section further operations on randomized sets, most of which are based on the primitives that we have introduced and analyzed in the previous section.

• **Cardinality.** We can obtain an estimate for the cardinality of the set M by

$$\mathrm{X} := \texttt{size(read(}M\texttt{))}.$$

Due to the probabilistic nature of the read operation, we cannot expect to get a precise answer and need to be contend with an estimate. It follows from Proposition 2 and Lemma 3 that $X \approx m$ assuming that the quorum size k is chosen such that $\varepsilon \ll 1$.

• **Union.** We can obtain a union of two sets (or multisets) B and C by

$$\texttt{read(B) union read(}C\texttt{)}. \tag{3}$$

We expect that `read(`B`)` and `read(`C`)` will fail to return at most $\varepsilon\,|B|$ and $\varepsilon\,|C|$ elements, respectively. Therefore, we can expect that the operation (3) yields at least $|B{\cup}C|(1{-}\varepsilon)$ elements. Furthermore, the result of (3) is contained in $B \cup C$.

• **Intersection.** The intersection

$$\texttt{read(B) intersection read(}C\texttt{)} \tag{4}$$

of two randomized sets (or multisets) has an expected value of at least $(1{-}\varepsilon)|B{\cap}C|$ elements, because the operation `read(`B`)` is expected to omit at most $\varepsilon|B|$ elements. And the result of (4) is contained in $B \cap C$, so one can boost the probability of success by several repetitions.

• **Difference.** The most subtle behavior has the operation

$$\texttt{read(B) minus read(}C\texttt{)}.$$

We expected number of elements returned by this operations is between $(1{-}\varepsilon)|B \setminus C|$ and $(1{+}\varepsilon)|B \setminus C|$; the potential increase of elements is a result of the fact that `read(`C`)` might return too few elements. We suggest to use a larger read quorum for `read(`C`)` to remedy this effect.

• **Delete.** The operation

$$\texttt{delete(x,M)}$$

removes the element `x` once from each replica. For instance, if M is a multiset with $M = \{\texttt{x}, \texttt{x}, \texttt{y}\}$, then after `delete(x,M)` the multiset is $M = \{\texttt{x}, \texttt{y}\}$.

• **Containment.** The operation

$$\texttt{x in } M \tag{5}$$

is realized by sending the request $\langle$is `x` in M?$\rangle$ to a quorum of servers that is selected uniformly at random from $\binom{[n]}{k}$. If at least one server replies with `true` then the result is `true`, otherwise it is `false`.

The answer has one-sided error. If `x` is not an element of M, then (5) will always correctly return `false`. If `x` is an element of M, then we get the incorrect answer `false` with probability $\leq \varepsilon$.

4 Application

In this section, we illustrate how to use our randomized multisets for location tracking in mobile ad hoc networks. Recall that the location tracking problem in a mobile ad hoc network is to provide location information of mobile nodes as part of a name resolution protocol. Such a location service is important because a simple name resolution protocol does not work in a mobile environment without static infrastructure.

Lee, Welch and Vaidya [5] proposed a scheme based on replicated location tracking servers. Each server has a replica of the location database containing the most recent location information of the mobile nodes. Each mobile node is responsible to periodically update its location.

Here we sketch a location tracking scheme using our randomized multisets in implementing the location information database M. *The basic idea of the new scheme is that we keep a "history" of movement of mobile nodes rather than the single most recent location information.* We conjecture that this scheme works well because movement is continuous and it is always better to have some outdated information than having no information at all. Consequently, the errors due to our randomized multiset have less impact in a system keeping the location history.

Each element of M is a tuple

$$\langle \textit{location data}, \textit{process id}, \textit{timestamp} \rangle.$$

Each mobile node (client process) is uniquely identified by the *process id*. Furthermore, each client process has a local timestamp which is incremented by one for each `add` operation it performs.

• **Add Location.** Whenever a client process i wants to record its new location information *loc*, it increments its timestamp t by one and invokes `add(x,`M`)`, where the element `x` is of the form $\langle loc, i, t \rangle$. The implementation of the `add` operation is given in Section 2.

Each replica server r keeps a replica M_r of M. When a replica server r receives $\langle$`add x`$\rangle$ message, it adds `x` to M_r. There is a system parameter *expire* that serves as a threshold of how long the old information should be kept. For example, if *expire* $= 5$, then each server will keep at most the five most recent location entries of each client process. Thus, the server will check after each `add` operation whether it should remove old location data of the client.

• **Lookup.** When a mobile node i wants to know the location of mobile node j, it performs `lookup(`j`,`M`)`. As a result of the `lookup` operation, process i will receive a list of location data of process j.

To implement the `lookup` operation, the client process does the following. It chooses a random quorum Q of replicas and sends the message $\langle$`lookup` $j\rangle$ to the replica server r in Q. The servers reply with lists of the location data of j, which the client merges and sorts according to increasing timestamps. Then it returns the sorted list of location data of j to the application.

When a server receives $\langle$`lookup` $j\rangle$ message from client i, it creates a list (that is ordered by timestamp values) of j's location data in M_r and sends the list back to i. If no such data exists in M_r, it sends i the list with a single value $\perp$, indicating that it does not have information on j.

5 Conclusions

We introduced distributed shared data structures for randomized sets and multisets. We described basic operations for the manipulation of these data structures. We will make a generic C++ implementation available that illustrates the fundamental principles of randomized sets. Our program is specified in Knuth's *literate programming* style so that it is readable for humans [4]. The documentation contains a self-contained explanation of all implementation details. The reader can experiment with our program to explore the practical consequences of quorum size choices and other details.

We showed that randomized multisets can be used to implement a location tracking service for mobile ad hoc networks. A little loss in location data is tolerable in this application, especially since the message complexity is significantly reduced so that more frequent location updates are feasible. Other applications in mobile computing, such as group communication, can benefit from such properties as well. As another type of application, we note that it is possible to modify Rabin's Byzantine agreement algorithm [9, 8] to take advantage of randomized multisets.

Acknowledgments. We thank Jennifer Welch for numerous discussions on randomized data structures, and Riccardo Bettati for discussions about implementations of Lamport's time. The research by H.L. was supported by University of Denver PROF grant 88197. The research by A.K. was supported by NSF CAREER award CCF 0347310, NSF grant CCR 0218582, a TEES Select Young Faculty award, and a Texas A&M TITF initiative.

References

[1] B. Chazelle. *The Discrepancy Method – Randomness and Complexity*. Cambridge University Press, 2001.

[2] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on a sum of observations. *Ann. Math. Statistics*, 23(4):493–507, 1952.

[3] S. Jukna. *Extremal Combinatorics with Applications in Computer Science*. Springer, 2001.

[4] D.E. Knuth. Literate programming. *Computer Journal*, 27:97–111, 1984.

[5] H. Lee, J.L. Welch, and N.H. Vaidya. Location tracking using quorums in mobile ad hoc networks. *Ad Hoc Networks*, 1(4):371–381, 2003.

[6] D.S. Lemons. *An Introduction to Stochastic Processes in Physics*. The Johns Hopkins University Press, 2002.

[7] D. Malkhi, M.K. Reiter, A. Wool, and R.N. Wright. Probabilistic quorum systems. *Information and Computation*, 170:184–206, 2001.

[8] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[9] M.O. Rabin. Randomized byzantine generals. In *Proc. of the 24th Annual Symp. on Foundations of Computer Science*, pages 403–409, 1983.