Sketch based Distributed Garbage Collection, Theory and Empirical Evaluation

Joannès Vermorel

Sketch based Distributed Garbage Collection, Theory and Empirical Evaluation

Joannès Vermorel
Computational Biology Group, École des Mines de Paris
joannes.vermorel@ensmp.fr

November 2, 2005

Abstract

Distributed Garbage Collection (DGC) algorithms are fundamental components of modern object-oriented distributed systems. The design of a DGC raises numerous issues in term of performance and scalability. DGC object-migration approaches are known to be inefficient compared to graph cycle detection approaches because of the tremendous costs associated with object migration compared to graph-fragment migration where only the object identifiers are carried over the network. Yet, even object identifiers have a large memory footprint in practice.

In this paper, we introduce the idea of sketched Cycle Detection Message (CDM). We show that explicit graph-fragment representations are very costly compared to the sketch-based ones. We prove, under reasonable assumptions that sketch-based messages are smaller (up to one order of magnitude) than their explicit counterparts. Those results apply to most of state-of-art DGC methods. The improvement brought by this approach is discussed in details in the particular case of the Veiga and Ferreira DGC algorithm. In this case, the amount of improvement is more than a factor 3 under very limited assumptions.

1 Introduction

Garbage collection has been a major element in terms of programmer productivity and program reliability [19]. Garbage collection identifies *live* (or reachable) and *dead* (or unreachable) objects in the application object graph. From the GC viewpoint, any application can be regarded as a *mutator* with a single relevant operation: reference-assignment. DGC is the natural extension of the Local Garbage Collection (LGC) in the context of distributed systems. The design of a DGC algorithm raises numerous issues and a large literature [18, 10, 1, 17] is devoted to such algorithms.

We believe that asynchronicity and completeness are two highly desirable, yet most challenging properties, for a DGC algorithm. The **asynchronicity**

Figure 1: Global identifier memory footprints

Distributed system	Platform	Global identifiers	Size in bits
NGrid	.Net	`System.Guid`	128*
JoCaml	SSPC	`Location`	128†‡
ProActive	Java	`VMID` + `UID`	288†
Mozart	Mozart	Ticket	272
Globus	Java	URLs	> 500

(*) See UUID discussion in Note 1.
(†) Assume a 32bits IPv4 address.
(‡) Assume a 32bits machine address space.

states no synchronization should be required between more than two machines at the same time. This property is critical for the potential scalability of a DGC algorithm. If recent methods are fully asynchronous[18, 5], many algorithms are not and require a centralized architecture[2, 9, 11] or group synchronization[16, 15, 7] or consensus[6] between the processes. The **completeness** states that all unreachable but only unreachable graphs, in particular *cyclic* graphs, must be collected. Indeed distributed cycles are frequent [14], and recent works seem to indicate that object-oriented design tends to generates scale-free graphs [13] (scale-free graphs are cyclic with a very high probability). Other architectural properties are also desirable such as *fault-tolerance* (robustness against message delay or loss), *isolation*, (DGC implementation must not require modifications of LGC or Remote Procedure Call (RPC) subsystems), *non-disruptive* (no pause in the application), *promptness* (garbage should be promptly collected).

In respect to this list of desirable properties for the DGC algorithms, we believe the wide spectrum of Distributed Cycle Detection Algorithm (DCDA) constitute the most scalable approaches for the DGC problem. To our knowledge, all cycle detections methods ([18, 5] for recent examples) are relying on messages carrying object-graph subsets in one form or an other. We would like to emphasize that global object-identifiers used in distributed systems are expensive. The Figure 1 illustrates the memory footprints of global identifiers in various distributed systems. Since those global identifiers need to be generated in a totaly decentralized fashion, we believe that 200bits is a reasonable lower bound estimation of the global identifier size for present and future large scale distributed systems[1].

The memory footprint of object identifiers calls for a more compact representation. Borrowing the sketching idea at the heart of the data streams domain

[1] Considering the UUID standard [8], 128bits seems a natural estimation for a global identifier size. But it should be noticed the UUID standard assume that all actors (see Section 6 of [8]) are fully trusted (no adversarial identifier generation). The price for a certain level of security, that we believe unavoidable in large scale distributed systems, is a larger identifier. Additionally The UUID standard also assumes a 48bits spatial resolution based on 802 MAC compliant addresses; and consequently admit no address resolution. Yet resolution can be a highly desirable property for global identifiers (Globus, JoCaml, ProActive identifiers provides address resolution for example). In such case, 128bits IPv6 addresses will probably be much more adequate for spatial resolution, requiring again a larger identifier size.

(see [12] for an excellent introduction), it can be noted that keeping an explicit representation of the data is, sometimes, neither required nor efficient, when the underlying objective are only associated to some its properties. A *sketch* refers to a compact representation of the data fitting certain pre-determined purposes. Many sketches have been introduced in the literature. It can be noticed that the HashMatrix sketch(see Section 2.2) shares a few similarities with the `Count-Min` sketch [4] used to detect frequent items in data streams (the underlying constrains are very different though).

1.1 Our results

We introduce, to our knowledge, the idea of performing cycle detection through sketched messages rather than explicit ones. This approach states that object identifiers (and sometimes their associated properties) can be sketched rather than explicitly carried over the network. We provide several sketches with various properties. Under reasonable assumptions, those sketches are up to one order of magnitude smaller than their explicit counterparts.

Those sketches apply to most of the DCDA. In particular, we discuss extensively how the Veiga and Ferreira DCDA [18] can be improved through through a sketch-based approach. For this particular example, the overall gain in term of overall network transmission is roughly between a factor 2 and 3 under weak assumptions.

2 Sketching sets of identifiers

Those factual evidence lead to the simple conclusion that the performance of DGC algorithms relying on messages carrying sets of object identifiers are going to be heavily hindered, no matter the quality of the underlying their strategies, because of the identifier weights.

Nevertheless, as we will see in Section 3, the explicit representation of identifiers is not a requirement. Often, requirements (on identifier sets) are restricted to certain operations such as inclusion testing, item insertion, set equality testing, ... In this section, we propose, through sketches, much more compact set representations than explicit extensive listing. Those sketches are only accuracy on a certain (explicitly quantified) probability. The consequences of relying on a possibly inexact identifier representation is very algorithm-dependent and therefore left to the Section 3.

2.1 HashList sketch

Since items are very large, a very naive, yet efficient, approach to set sketching is simply to substitute a small hash to each object identifier. Formally let E be the identifier space (simply called item space in the following). Let $S = \{s_1, \ldots, s_n\} \subset E$ be a set. Let $h : E \to \{0,1\}^b$ be a random hash function that associates a random bit-vector of length b to each item $x \in E$. We define

the HashList sketch sketch of S with $\mathbb{H} = \{h(s_1), \ldots, h(s_n)\}$. Based on this definition, several operations are very natural to define such as

- inclusion test $h(x) \in \mathbb{H}$ as equivalent of $s \in S$,
- union $\mathbb{H}_1 \cup \mathbb{H}_2$ as equivalent of $S_1 \cup S_2$,
- intersection $\mathbb{H}_1 \cap \mathbb{H}_2$ as equivalent of $S_1 \cap S_2$.

Since a hash is non-injective function, false positive inclusion can occur with the HashList sketch.

Lemma 1 (HashList sketch false positive rate) *Let $\mathbb{H}$ be the* HashList *sketch equivalent of $S \subset E$ with b bits allocated per item. Let assume that $|S| = n$. Let $x \in E$. If $x \in S$ then $h(x) \in \mathbb{H}$. If $x \notin S$ then $h(x) \notin \mathbb{H}$ with a false positive probability of $1 - \left(1 - 2^{-b}\right)^n$ (note: $2^{-b}n$ can be used as a more practical upper bound on this probability).*

Proof: Immediate. □

Numerical illustration: By allocating 24bits per item for a HashList sketch containing 1000 items, the false positive rate is lower than 5.10^{-5}.

The result provided by the Lemma 1 is already good enough for practical applications (as suggestion the numerical illustration here above), but is weak somehow because for an allocation of b bits per item, the false positive probability increase (almost) linearly with n the number of items. The following definition formalizes this idea.

Definition 1 (Sketch ϵ-capacity) *Let $\mathbb{S}$ be an empty sketch where b bits are allocated per item on average. The ϵ-capacity of $\mathbb{S}$ is defined as the number of items that can be added to $\mathbb{S}$ while maintaining a collision[2] probability lower than ϵ for elements not previously added.*

The following theorem provides some insights on the ϵ-capacity of the HashList sketch.

Theorem 1 (HashList sketch ϵ-capacity) *Let $\mathbb{H}$ be a* HashList *sketch with b bits allocated per item. The ϵ-capacity of $\mathbb{H}$ is greater than $2^{b/2}\epsilon$.*

Proof: Immediate using Lemma 1. □

[2]We says that a *collision* happen when then insertion of an item into the sketch does not modify the sketch (although it should).

2.2 HashMatrix sketch

The Theorem 1 indicates that the HashList sketch capacity for a given amount b of bits allocated per item is *bounded*. This result can be considerably improved. Below, we introduce a new sketch structure named the "HashMatrix sketch" that provides a much better capacity that the HashList sketch for any given ϵ false positive error rate.

Let's start with a empty set S, and consider an arbitrary sequence of additions $x_0, x_1, \ldots, x_n$ applied to S where x_k is an arbitrary element taken from a possibly infinite set. The additions are nothing but set union, in this respect, it should be noted than the union is associative $(a \cup b) \cup c = a \cup (b \cup c)$, symmetric $a \cup b = b \cup a$ and idempotent $a \cup a = a$. Therefore if $\mathbb{S}$ is refers to the sketch used in place of S and $\oplus$ to the union-equivalent operation, the three properties $(a \oplus b) \oplus c = a \oplus (b \oplus c)$, $a \oplus b = b \oplus a$ and $a \oplus a = a$ are required as well. Among the common data structures and operations, it can be noticed that bit vectors associated to bitwise-OR (or bitwise-AND) verify those three properties.

We define a HashMatrix sketch $\mathbb{M} = (v_1, \ldots, v_p)$ as a set of p bit-vectors of length q. In the following, such sketch $\mathbb{M}$ is refered as a (p, q) sketch. We assume that independent random hash functions $h_i : E \mapsto \mathbb{N}$ are available *a priori*. The empty set is associated, by definition, with $\mathbb{M}_\emptyset = (0, \ldots, 0)$. Let 1_i be the bit-vector where the i^{th} bit is the sole bit set to 1. The singleton sketch associated to the object $x \in E$, is defined with $\mathbb{M}_x = (1_{i_1}, 1_{i_2}, \ldots, 1_{i_p})$ where i_k corresponds to $h_i(x)$ **mod** q. The union-equivalent $\oplus$ operation is defined with

$$(v_1, \ldots, v_p) \oplus (w_1, \ldots, w_p) = (\mathrm{OR}(v_1, w_1), \ldots, \mathrm{OR}(v_p, w_p))$$

Based on the previous discussion, it's clear that $(\mathbb{M}, \oplus)$ verifies the union properties. The inclusion-equivalent test, based on $\mathbb{M}$ is defined with

$$x \lhd \mathbb{M} \Leftrightarrow (AND(s_1, 1_{i_1}), \ldots, AND(s_p, 1_{i_p})) \neq \mathbb{M}_\emptyset$$

The inclusion-equivalent test is not strictly equivalent because there is the possibility of *false positive*, i.e. items not included in S but declared as included in $\mathbb{M}$. The following lemma characterizes the false positive rate.

Lemma 2 (False positive rate of the HashMatrix sketch) *Let $(\mathbb{M}, \oplus, \lhd)$ be the a (p, q) sketch equivalent to $(S, \cup, \in)$ a set with n elements. Then $x \in S$ implies $x \lhd \mathbb{S}$ and*

$$x \notin E \Rightarrow x \not\lhd \mathbb{M} \text{ with a probability } P(p, q, n) = \left(1 - \left(1 - \frac{1}{q}\right)^n\right)^p$$

Proof: Immediate. □

Numerical illustration: Let $P(k, n) = \min_{pq=k} P(p, q, n)$ we have $P(5 * 16, 5) < 0.001$. This result can be interpreted as: with only 16bits per identifier, the sketch of 5 objects can accurately assert object inclusion with a probability lower than $\frac{1}{1000}$. The memory footprint of the explicit is 1.000bits assuming a

reasonable 200bits per object whereas the memory footprint of the HashMatrix sketch is only 80bits. The sketch is more than 10 times smaller than the initial set.

It should be noticed that the bitwise comparison between two set sketches provide a global equality equivalent. The following corollary of the Lemma 2 quantifies the quality of this equivalence.

Corollary 1 (False positive rate on HashMatrix sketch equality) *Let $(\mathbb{M}_1, \oplus, \lhd)$ (resp. $\mathbb{M}_2$) be the a (p, q) sketch equivalent to $(S_1, \cup, \in)$ (resp. S_2) a set with n (resp. m) elements. Then $S_1 \cap S_2 = k$ implies $\mathbb{M}_1 = \mathbb{M}_2$ with probability*

$$P(p, q, n)^{m-k} P(p, q, m)^{n-k}$$

Therefore, if two sets differ by more than one element, the false positive rate of HashMatrix sketch equality is lower than the inclusion false positive rate.

The design of a HashMatrix sketch depends the two parameters p and q. Nevertheless, as suggest the Lemma 2, for a fixed bit allocation $p \times q$, it exists an *optimal* repartition. The following theorem characterizes this optimality.

Theorem 2 (Asymptotic optimal set sketch design) *Let us consider a* HashMatrix *sketch with b bits allocated per item such that $p.q = b.n$. Let γ be such that $q = \gamma n$ then*

$$\lim_{n \to \infty} \min_{\gamma} P(p, q, n) = 2^{-b \ln(2)} \text{ with } \gamma = \frac{1}{\ln(2)} = 1.442695041..$$

Proof: Tips: $\lim_{n \to \infty} \left(1 - \frac{a}{n}\right)^n = e^{-a}$. □

The result of the Theorem 2 can be interpreted with for n items and b bits per items, we must choose $p = b \ln(2)$ and $q = \frac{n}{\ln(2)}$. Note that this choice is not exactly optimal, but a more detailed numerical analysis indicates that the optimal ratio is already very close to $1/\ln(2)$ for n not larger than 10.

Corollary 2 (HashMatrix sketch ϵ-capacity) *Let $\mathbb{M}$ be a* HashMatrix *sketch with b bits allocated per item. The ϵ-capacity of $\mathbb{M}$ is greater than $2^{b \ln(2)} \epsilon$.*

Proof: Consider $\epsilon = 2^{-b \ln(2)} N$ where N is a lower bound on the ϵ-capacity (based on the Theorem 2). □

The capacity bound of the HashMatrix sketch is much better than HashList sketch'one because $\ln(2) \simeq 0.69 > 1/2$. Additionally, it should be noticed that this bound is not *tight*, indeed we have used the following upper bound on the collision probability

$$1 - \prod_{i=1}^{n} (1 - P(b \ln(2), n \ln(2), i)) > 1 - n * P(b \ln(2), n \ln(2), n)$$

2.3 Incremental HashMatrix sketch

In practice, the number n of items to be added in the sketch is not known initially. Additionally, even if this number can be known or estimated, the sketch may possibly be carried over the network after each addition, therefore a large initialization defeats the sketch purpose. A simple solution to this issue consists of *incrementally increasing the sketch size*. This solution is very simple and natural for the HashList sketch but more delicate for the the HashMatrix sketch. In this section, we will discuss how the HashMatrix sketch can be modified to perform the insertions *incrementally*.

Formally, let us consider the sketch list

$$\mathbb{M}_* = \{\mathbb{M}_0, \mathbb{M}_1, \ldots, \mathbb{M}_k\} = \left\{(p,q); (p,\alpha^1 q); (p,\alpha^2 q); \ldots; (p,\alpha^k q)\right\}$$

The addition of a new element is always performed on the last HashMatrix sketch $\mathbb{M}_k$. The sketch $\mathbb{M}_k$ is considered as *full* when it's capacity is reached, i.e. after $\alpha^k/\ln(2)$ additions. At this point, a new empty sketch $\mathbb{M}_{k+1}$ is added (following the exponential allocation pattern). The inclusion test $x \lhd \mathbb{M}_*$ is defined with

$$x \lhd \mathbb{M}_1 | x \lhd \mathbb{M}_2 | \ldots | x \lhd \mathbb{M}_k \text{ (where } | \text{ is the logical OR)}$$

The choice of b and α for an incremental sketch clearly defines it's *capacity*. The following theorem quantifies the false positive error rate.

Theorem 3 (False positive rate of incremental HashMatrix sketch) *With b bits allocated per item on average, the incremental* HashMatrix *sketch provides a false positive rate lower than $2^{-b\ln(2)\frac{1+\alpha}{2\alpha}}\log_\alpha(n)$.*

Proof: Tips: $\sum_{i=0}^{k}\alpha^i = \frac{\alpha^{k+1}-1}{\alpha-1}$. Consequence of Theorem 2 □

Numerical illustration: starting with a sketch smaller than 200bits (single explicit identifier representation) and taking $\alpha = 2$, we can incrementally add 1000 items, allocating 40bits per item on average, with a false positive rate staying below 5.10^{-6} at all time.

Corollary 3 (Incremental HashMatrix sketch ϵ-capacity) *Let $\mathbb{M}^*$ be an incremental* HashMatrix *sketch with b bits allocated per item. The ϵ-capacity of $\mathbb{M}^*$ is greater than*

$$\frac{2^{b\ln(2)\frac{1+\alpha}{2\alpha}}}{W\left(2^{b\ln(2)\frac{1+\alpha}{2\alpha}}\right)}$$

Where W is the Lambert W function (i.e, the inverse of $x \mapsto xe^x$, see [3]).

Asymptotically, this capacity bound on the incremental HashMatrix sketch is better than the HashList sketch one. In practice, with $\alpha = 1.1$, the incremental HashMatrix sketch capacity gets higher than HashList sketch capacity for any $b > 15$.

Incremental set sketch global equality. If a simple bitwise sketch comparison is sufficient to check for HashMatrix sketch equality (see details in previous section); the incremental case is more complicated. Indeed, since we have not made any assumption on the *order* of the item additions, we cannot rely on a bitwise comparison of the two sketch sequences. Nevertheless, the global equality can be obtained through a simple and very lightweight hash. The following lemma defines and quantifies the error rate of such set equality hash.

Lemma 3 (Hash based set equality) *Let $S_1 \subset E$ and $S_2 \subset E$ be two sets. Let $h : E \to \{0,1\}^b$ be a random hash function. The set hash of S_1 is defined with $h(S_1) = \underset{s \in S_1}{XOR}\, h(s)$ (idem for S_2). If $S_1 = S_2$ then $h(S_1) = h(S_2)$ (no false negative) and if $S_1 \neq S_2$ then $h(S_1) \neq h(S_2)$ with a false positive probability of 2^{-b}.*

The global equality operation can be made available for a incremental HashMatrix sketch through the adjunction of a set hash, as defined in Lemma 3. Since that false positive error rate of Lemma 3 does not depend on the size of considered sets, the practical cost (in bits) of the additional hash is negligible[3].

3 Sketching the Veiga & Ferreira DCDA

Veiga and Ferreira present in [18] an asynchronous, complete DGC. Intuitively the DCDA proceeds by an initial candidate selection that is followed by a sequence of CDMs send between the processes. CDMs are equivalent to the `GraphSummary` data structure here below (note that the graph summary is already been optimized to avoid identifier duplicates). In this section, we propose to leverage the compactness of the sketch-based CDMs to quickly detect cycles (with a risk of false positive detection) and to verify the detection correctness afterwards through explicit CDMs. This approach has no virtually impact on the DCDA, only the CDM encoding is truly affected.

```
type graph_summary = {
    list of (
        unique_id,
        is_source_flag, is_target_flag,
        timestamp);
}
```

[3] Additionally, it can be noticed that if we are considering an equality between *two* incremental HashMatrix sketches, then only *one* hash is necessary. When a new item is added to one of two sketches, the hash is updated with a XOR between the previous hash value and the item hash. The equality between the two sketches is then simply defined by having the hash equal to $[1, 1, \ldots, 1]$ (because x XOR $x = 1$ for any bit-vector x)

Upon CDM delivery, only a limited list of operations are performed[4] on the graph summary. Those operations are

(a) insertion in the summary of a new timestamped identifier either flagged as source or (exclusive) as target,

(b) test of equality between one the summary timestamp and an other timestamp for a specified identifier,

(c) setting the second flag of an identifier to *true*,

(d) testing global equality between the set of identifiers flagged as source and those flagged as target.

In terms of CDM memory footprint, we will consider the identifier size equals to 200bits (see discussion in Section 1). The two flags require naturally 2bits. In [18], it is suggested to code the timestamps as 32bits integers. Timestamps are incremented on each object operation, since 2^{30} is already the order of magnitude the number of elementary operations performed per second by an common desktop processor, we believe that 32bits is too low to guarantee collision-free timestamps in future large scale distributed systems. Based on empirical hardware consideration, we believe that 64bits is a more realistic timestamps size.

We propose to sketch the `graph_summary` based on the set sketch previously introduced. The CDM becomes

```
type graph_sketch = {
    list of (hash, is_source_flag, is_target_flag);
}
```

The `graph_sketch` handles separately sources, targets and timestamps. The `sources` and `targets` sketches are respectively associated to the identifiers flagged as source and target. As detailed below, the timestamps are simply ignored. The sketch-equivalent operations are

(a) Insertion of the identifier hash flagged correspondingly. Ignore timestamp.

(b) Always return *true*.

(c) Setting both flags of the identifier hash to *true*.

(d) Identical.

Since, the set sketch has no false negative, in can easily be proved that the detection errors caused by the `graph_sketch` are restricted to false positive cycle detections. Therefore we propose following *mixed* strategy: when a candidate is chosen for cycle detection, start a DCDA initiative based on sketched CDMs.

[4]When an identifier is added to the CDM list, the local invocation counter timestamp must match, if present, its CDM counterparts; if not, a race condition has been encountered and the CDM is terminated. The cycle detection condition is defined as an equality between the items flagged as source and the items flagged as targets.

In case of cycle detection, start a new DCDA initiative based on explicit CDMs, taking the suspected object (from the cycle detection point) as initiative candidate[5]. It's easy to prove that this mixed strategy is *correct* (i.e. all garbage but only garbage is collected).

Mixed DCDA performance analysis. The per-item cost for the explicit CDMs cost is estimated as $c_e = 266$bits (see discussion here above). In order, to estimate the relative interest of the sketch-based approach, we need an estimation of this cost when sketch-based and explicit CDM s are mixed following the strategy described here above.

The performance analysis of the mixed DCDA requires several additional hypotheses. Let P_g be the *a priori* probability of detecting a cycle for a DCDA initiative. In practice, the value of P_g is highly dependent of the candidate generation heuristics. The lack of widely used distributed object system providing complete DGC is an obstacle to provide a rigorous estimation of P_g at this time. Nevertheless, we believe that a good tradeoff between DGC promptness and DGC resource consumptions involve a majority of failures for DCDA initiatives. This belief is motivated by the results of [14] concerning the object age frequency distribution (no simple behavior seems to govern the object-lifetime). For the purpose of the analysis, we will rely on $P_g = 10\%$ in the following. We believe this estimate to be quite high, empirical evaluations may provide a lower success rate.

Let P_T be the `a priori` probability of DCDA interruption based on the timestamps matching. Actually, timestamps in the Veiga & Ferreira DCDA are used to prevent race conditions with local mutator that would break the DCDA validity. If timestamps are critical in term of *correctness*, we believe the DCDA interruptions based on timestamp matching is too low[6] to have any noticeable impact on performance in practice. Indeed, race condition involve a complicated root displacement that must occurs in a timely fashion with the DCDA execution. Those elements lead us to strongly believe that $P_T < 1/1000$ (empirical evaluations may provide a bound one or two order of lower than that). Therefore, in the following of the analysis, those events will simply be ignored.

Let P_n be the *a priori* probability for an DCDA initiative that the graph summary reaches n items at a point of its execution. As would suggest statistical consideration of scale-free graphs, the results provided in [14] indicates (we are considering the measurements of the size frequencies of strongly connected components of the considered graphs by [14]). The values of P_n are important because, they will be used in practice to determine the initial capacity of 1 of the sketch.

Based on the previous considerations, if ϵ is the false positive cycle detection rate when relying on an allocation of b bits per item, the per-item c_m cost for the

[5] It's possible to exploit the information of the sketch CDM in order to speed up the explicit DCDA execution. Such discussion goes beyond the scope of this paper.

[6] Caution: we *do not* say that timestamps checking can be ignored for performance. We say that race condition detections are too rare to impact the overall DCDA performance.

Figure 2: Per-item costs of HashList sketch-based CDM.

b_L	ϵ	capacity	c_m
20	0.009	10	51
26	0.012	100	58
32	0.015	1000	64
40	0.009	10000	71
46	0.011	100000	78
52	0.014	1000000	84

Legend: b_L is the number of bits allocated per item in the HashList sketch. ϵ is an upper bound of the false positive cycle detection rate. *capacity* is a lower bound on the maximal number of items that can be incrementally added. c_m is the average CDM footprint in bits.

whole mixed strategy can estimated with $c_m = b + 2 + (1 - (1 - P_g)(1 - \epsilon)) * c_e$ (sum of the sketch-based per-item cost plus the explicit per-item cost when it occurs). The Figure 2 provides a list of numerical values for c_m depending on the various initial choices for b and α (those values have been computed based on the results of Section 2.1). Notice that the higher b is initially chosen, the higher the graph sketch capacity.

Improving the graph sketch In Section 2.2, we have seen that the HashMatrix sketch is more efficient than the HashList sketch. Yet the HashMatrix sketch cannot efficiently handle the identifier flags like the HashList sketch. Therefore we propose to improve the graph sketch by using a mix of HashList sketch and HashMatrix sketch. Intuitively, we propose to store all single-flagged identifiers in the HashList sketch as we do here above. But when an item becomes fully flagged, the item is removed from the HashList sketch and moved into the HashMatrix sketch. Since the HashMatrix sketch is more efficient than the HashList sketch, smaller CDM footprint can be expected. The more fully flagged items we have, the closer we are from the HashMatrix sketch performance.

In order to quantify this approach, we need to know P_2 the average percentage of items being fully flagged during the DCDA execution. It's possible to prove[7] that in case of random insertions of items either flagged as source or target, we have $P_2 = \frac{2}{3}$. In the following, we assume $P_2 = \frac{2}{3}$, although we believe this estimate to be quite low. Indeed, random insertions correspond to the worst case situation whereas the DCDA execution is highly biased in our favor because the items flagged as target are explored first. Based on all those assumptions, the numerical results are gathered in Figure 3. The improvement is roughly 10% over the results of the pure HashList sketch approach presented in Figure 2.

[7]Such proof goes beyond the scope of this paper.

Figure 3: Per-item costs of HashList and HashMatrix mixed CDM.

b_L	b_M	ϵ	capacity	c_m
22	15	0.009	10	47
30	20	0.007	100	52
36	24	0.010	1000	58
42	29	0.010	10000	63
49	34	0.009	100000	69
57	40	0.014	1000000	74

Legend: b_L (resp. b_M) is the number of bits allocated per item for the HashList sketch(resp. HashMatrix sketch). ϵ is an upper bound of the false positive cycle detection rate. *capacity* is a lower bound on the maximal number of items that can be incrementally added. c_m is the average CDM footprint in bits.

4 Conclusion

The large footprint of distributed object identifiers (estimated as more than 200 bits) calls for more compact representations. In this paper, we have introduce several sketches, that is approximate compact representations, of sets of identifiers. The properties, in particular the level of approximation, have been rigourously quantified.

Considering one the more recent DCDA at this date (see [18]), the improvement brought by the sketch-based CDM is roughly a factor 3 under limited (partly adversarial) assumptions. Since our approach is not specific of this algorithm, we believe that similar improvements can be obtained with most of the other DCDA. Additionally, since this work is only preliminary, it's also probable that better sketches can be found.

References

[1] Saleh E. Abdullahi and Graem A. Ringwood. Garbage collecting the internet: a survey of distributed garbage collection. *ACM Computing Surveys*, 30(3):330–373, 1998.

[2] Peter Bishop. Computer systems with a very large address space, and garbage collection. Technical Report MIT/LCS/TR-178, Massachusetts Institute of Technology, May 1977.

[3] Robert M. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and D. E. Knuth. On the Lambert W Function. In *Advances in Computational Mathematics*, volume 5, pages 329–359, 1996.

[4] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[5] Fabrice Le Fessant. Detecting distributed cycles of garbage in large-scale systems. In *PODC'01: Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, pages 200–209, New York, NY, USA, 2001. ACM Press.

[6] John Hughes. A distributed garbage collection algorithm. In *Proc. of a conference on Functional programming languages and computer architecture*, pages 256–272, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[7] Bernard Lang, Christian Queinnec, and José Piquer. Garbage collecting the world. In *POPL'92: Proceedings of the 19th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 39–50, New York, NY, USA, 1992. ACM Press.

[8] P. Leach, M. Mealling, and R. Salz. A Universally Unique IDentifier (UUID) URN Namespace, RFC 4122. `ftp://ftp.rfc-editor.org/in-notes/rfc4122.txt`, July 2005.

[9] Barbara Liskov and Rivka Ladin. Highly available distributed services and fault-tolerant distributed garbage collection. In *PODC'86: Proceedings of the fifth annual ACM symposium on Principles of distributed computing*, pages 29–39, New York, NY, USA, 1986. ACM Press.

[10] Matthew Clifton Lowry. *A new approach to the train algorithm for distributed garbage collection.* PhD thesis, Adelaide University, 2004.

[11] Umesh Maheshwari and Barbara Liskov. Collecting cyclic distributed garbage by controlled migration. In *PODC'95: Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing*, pages 57–63, New York, NY, USA, 1995. ACM Press.

[12] S. Muthukrisnan. Data streams: Algorithms ans applications. `http://www.cs.rutgers.edu/~muthu/`.

[13] Alex Potanin, James Noble, Marcus Frean, and Robert Biddle. Scale-Free Geometry in OO Programs. *Communications of the ACM, Volume 48, Number 5*, May 2005.

[14] Nicolas Richer and Marc Shapiro. The Memory Behavior of the WWW, or The WWW Considered as a Persistent Store. In *POS-9: Revised Papers from the 9th International Workshop on Persistent Object Systems*, pages 161–176, London, UK, 2001. Springer-Verlag.

[15] Helena Rodrigues and Richard Jones. A cyclic distributed garbage collector for network objects. In *WDAG '96: Proceedings of the 10th International Workshop on Distributed Algorithms*, pages 123–140, London, UK, 1996. Springer-Verlag.

[16] Helena Rodrigues and Richard Jones. Cyclic Distributed Garbage Collection with Group Merger. In *ECCOP'98: Proceedings of the 12th European Conference on Object-Oriented Programming*, pages 260–284, London, UK, 1998. Springer-Verlag.

[17] Marc Shapiro, Fabrice Le Fessant, and Paulo Ferreira. Recent Advances in Distributed Garbage Collection. In *Advances in Distributed Systems, Advanced Distributed Computing: From Algorithms to Systems*, pages 104–126, London, UK, 1999. Springer-Verlag.

[18] Luís Veiga and Paulo Ferreira. Asynchronous Complete Distributed Garbage Collection. In *19th International Parallel and Distributed Processing Symposium (IPDPS 2005)*. IEEE Computer Society, 2005.

[19] Paul R. Wilson. Uniprocessor garbage collection techniques. In *International Workshop on Memory Management*, Saint-Malo, France, 1992. Springer-Verlag Lecture Notes in Computer Science no. 637.