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

***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.

I. Introduction

Garbage Collection (GC) has been a major advance in terms of programmer productivity and program reliability [1]. 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 [2], [3], [4], [5] 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** 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[2], [6], many algorithms are not and require a centralized architecture[7], [8], [9] or group synchronization[10], [11], [12] or consensus[13] 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 [15] (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).

TABLE I
Global identifier memory footprints

System	Platform	Global identifiers	Size in bits
NGrid	.Net	`System.Guid`	128[2]
JoCaml	SSPC	`Location`	128[3]
ProActive	Java	`VMID` + `UID`	288[4]
Mozart	Mozart	Ticket	272
Globus	Java	URLs	> 500

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 ([2], [6] 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 Table I 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 more compact representations. A *sketch* refers to an approximate compact representation of the data fitting certain predetermined purposes. The sketching idea, at the heart of the data streams domain (see [17] for an excellent introduction), states that keeping an explicit representation of the data is, sometimes, neither required nor efficient, when the underlying

[1] Considering the UUID standard [16], 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 [16]) 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.

[2] See UUID discussion.

[3] Assume 32bits IPv4 addresses and a local 32bits machine.

[4] Assume 32bits IPv4 addresses.

objective are only associated to some the data properties. Many sketches have been introduced in the data stream literature. In this paper, we are focusing on *set approximation* that are commonly known as *bloom filters*, originally introduced in [18]. Bloom filters have been used to solve a large variety of network related problems (although not for DGC to our knowledge), see [19] for a survey. Recent works on bloom filters includes the study of compressed bloom filters [20], optimal replacements of bloom filters [21], enhanced bloom filter with lookup capabilities [22] or frequency estimations [23].

A. 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 do not extend the bloom filter theory (that has been extensively studied), but we analyze and adapt previously known results for the purpose of DGC. Under reasonable assumptions, those sketches are up to one order of magnitude smaller than their explicit counterparts in the case of global identifier summarization.

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

II. Sketching sets of identifiers

A memory footprint of 200bits per global identifier is hindering factor for any DGC algorithm relying on identifier transmission. Yet identifier transmission is at the core of most of the DGC algorithm (with the notable exception of object-migration approaches).

Since the discovery of hashing, it has been known that sets can be represented in more compact manner than explicit listing if a certain degree of approximation is tolerated. Although very naive the hashing sketch can be very efficient and appropriate in the DGC context. The hashing sketch is discussed in Section II-A and it's DGC application is detailed in Section III. Since the introduction of bloom filters it is known that the hashing sketch is far from being an optimal solution. In Section II-B, we introduce a variant of bloom filters (previously known in the literature) and analyze its shortcoming for the purpose of DGC.

Nevertheless, as we will see in Section III, 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 III.

A. The hashing 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. The hashing sketch is not new (it has been used for decades, see [24]). This section introduce this structure with a perspective that will be useful in the context of DGC applications.

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 hashing 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 hashing sketch. The following quantifies the False Positive Rate (FPR) of the hashing sketch.

Lemma 1 (FPR of the hashing sketch) *Let $\mathbb{H}$ be the hashing 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 hashing sketch containing 1000 items, the false positive rate is lower than 5.10^{-5}.

For the purpose of sketch based DCDA, it should be noticed (this point will be more extensively discussed in SectionIII) that the initial choice of b, the number of bits per item determines the number of items that can ultimately be added to the sketch while a maintaining a bounded collision probability[5]. The following definition formalizes this concept.

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 probability lower than ϵ for elements not previously added.*

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

Theorem 1 (hashing sketch ϵ-capacity) *Let $\mathbb{H}$ be a hashing 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. □

It should be noticed that this result is weaker than what can be achieved with a simple bloom filter [18] where the

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

ϵ-capacity is $2^{\frac{b}{2^{\ln(2)}}}\epsilon$ ($\frac{1}{2^{\ln(2)}} \approx 0.62 > 0.5$). Nevertheless, as we will see the simplicity of the hashing sketch has many advantages for the DGC purposes (see Section III).

B. The bloom filter sketch

In this section, we introduce the bloom filter sketch[6] that is known to provide better performance than the hashing sketch. More efficient, yet more complicated structures exist such as the compressed bloom filters [20]. Because of their complexity and additional computation requirements, those structures are beyond the scope of this paper.

We define[7] a bloom filter sketch $\mathbb{M} = (v_1, \ldots, v_p)$ as a matrix comprising p bit-vectors of length q. Each bit-vectors is associated to a random hash function $h_i : E \mapsto \{1, \ldots, q\}$ available *a priori*[8]. In the following, such sketch $\mathbb{M}$ is refered as a (p, q) sketch. 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 $i_k = h_k(x)$. The union-equivalent $\oplus$ operation is defined with

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

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 (FPR of the bloom filter 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 $x \notin E$ implies $x \not\lhd \mathbb{M}$ with a probability*[9]

$$P(p, q, n) = \left(1 - \left(1 - \frac{1}{q}\right)^n\right)^p$$

Proof: Immediate, see also [19]. □

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 bloom filter sketch is only 80bits. The sketch is more than 10 times smaller than the initial set.

The design of a bloom filter 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 (Optimal allocation for the bloom filter sketch) *Let us consider a bloom filter sketch with b bits allocated per item such that $p.q = b.n$. Let γ be such that $q = \gamma n$ then*

$$\lim_{n\to\infty} \min_{\gamma} P(p, q, n) = 2^{-b\ln(2)}$$

and $\gamma = \frac{1}{\ln(2)} = 1.442695041..$ is the minimum.

Proof: Indication: $\lim_{n\to\infty} \left(1 - \frac{a}{n}\right)^n = e^{-a}$. See [19] for the proof. □

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

Proof: Immediate. □

The capacity bound of the bloom filter sketch is much better than hashing sketch'one because $\ln(2) \simeq 0.69 > 1/2$. Although, the bloom filter sketch raises a new issue compared to the hashing sketch: the ϵ-capacity is determined by the sketch size as a whole in a *non-incremental* manner. Contrary to the hashing sketch, the size of the bloom filter sketch is independent of the number of sketched items. Intuitively, this implies that a large capacity bloom filter sketch requires a large amount of bits, even if the sketch contains initially a very limited number of items. If used *as such*, this drawback defeats the bloom filter sketch purpose of being a better replacement for the hashing sketch.

C. Incremental sketches

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 compacity purpose. Therefore, the sketch size must *incrementally* increase when items are added. Such behavior is very natural for the hashing sketch but more delicate for the the bloom filter sketch. In this section, we introduce a general method to incrementally grow a sketch.

Formally, let us consider the sketch list

$$\mathbb{S}_* = \{\mathbb{S}_0, \mathbb{S}_1, \ldots, \mathbb{S}_k\} \quad (1)$$
$$= \left\{(b, m); (b, \alpha^1 m); (b, \alpha^2 m); \ldots; (b, \alpha^k m)\right\} \quad (2)$$

[6] We do not exactly introduce the bloom filters as originally presented in [18]. Instead, we are presenting a variant (see [19]), that slightly improves the bloom filter performance (but provides the same asymptotic bounds)

[7] In comparison, a bloom filter is a single bit-vector of length m associated to k hash functions (taking usual notations of the literature). The bit-matrix of the bloom filter sketch can be viewed a single vector of length $m = p \times q$. In both case, there are two parameters: the size of the structure and the the number of hash functions.

[8] It has been emphasized in the literature that hash functions are not *free* and require memory as well (see [21] for recent work on that matter). Nevertheless, in the case of DGC applications, the total number of hash functions required in practice is very limited. Moreover hash functions can be pooled (and reused). In practice, hash function costs are negligible in the case of DGC applications.

[9] Notice the similarity with the FPR of the original bloom filter that is equal to

$$P(k, m, n) = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k$$

where b is the number of bits allocated *per item*, and $\alpha^i m$ the number of items of the i^{th} sketch. The addition of a new element is always performed on the last sketch $\mathbb{S}_k$. The sketch $\mathbb{S}_k$ is considered as *full* when it's capacity is reached, i.e. after $\alpha^k m$ additions. At this point, a new empty sketch $\mathbb{S}_{k+1}$ is added (following the exponential allocation pattern). The inclusion test $x \lhd \mathbb{S}_*$ is defined with

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

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 (FPR of incremental sketch) *With b bits allocated per item on average, the incremental sketch provides a false positive rate lower than $2^{-b\gamma\frac{1+\alpha}{2\alpha}}\log_\alpha(n)$ where γ depends of the initial FPR (assuming a FPR equal to $2^{-b\gamma}$).*

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

Numerical illustration: starting with a single bloom filter 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 2 (ϵ-capacity of the incremental sketch) *Let $\mathbb{S}^*$ be an incremental sketch with b bits allocated per item. The ϵ-capacity of $\mathbb{S}^*$ is greater than*

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

Where W is the Lambert W function (i.e, the inverse of $x \mapsto xe^x$, see [25]).

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

III. Sketch based DCDA

Based on the approximate compact representations of identifier sets (introduced in the previous section), we will now discuss how the sketching approach can be used to improve the DCDA performance.

A. Veiga & Ferreira DCDA

Veiga and Ferreira present in [2] 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);
}
```

Upon CDM delivery, only a limited list of operations are performed[10] 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 I). The two flags require naturally 2bits. In [2], 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.

B. Veiga & Ferreira CDM sketching

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 proposed `graph_sketch` is based on a hashing sketch and shares some similarities with the original graph summary. 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

[10]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.

`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. 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[11]. It's easy to prove that this mixed strategy is *correct* (i.e. all garbage but only garbage is collected).

C. Sketched 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 of 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). Additionally, the availability of much cheaper CDM is, itself, a strong bias in the estimation of P_g. Indeed, the cheaper the DCDA, the more detection initiatives can be started for a given amount of network bandwidth dedicated to the DGC. Therefore cheaper CDM enable the improvement of both the DGC promptness and DGC bandwidth allocation by increasing the rate of cycle detection initiatives (consequently lowering the value of P_g). 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[12] 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.

TABLE II
PER-ITEM COSTS OF HASHING 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 hashing 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.

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 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 Table II 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 II-A). Notice that the higher b is initially chosen, the higher the graph sketch capacity.

D. Improved sketched DCDA

In Section II-B, we have seen that the bloom filter sketch is more efficient than the hashing sketch. Yet the bloom filter sketch cannot efficiently handle the identifier flags like the hashing sketch. Therefore we propose to improve the graph sketchf[13] by using a mix of hashing sketch and bloom filter sketch. Intuitively, we propose to store all single-flagged identifiers in the hashing sketch as we do here above. But when an item becomes fully flagged, the item is removed from the hashing sketch and moved into the bloom filter sketch. Since the bloom filter sketch is more efficient than the hashing sketch, smaller CDM footprint can be expected. The more fully flagged items we have, the closer we are from the bloom filter 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[14] 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 $P_2 = \frac{2}{3}$,

[11] 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.

[12] 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.

[13] Since the graph summary can be seen as a lookup table associating two bits to each identifier one can be tempted to rely on the bloomier filter introduced in [22]. Unfortunately, bloomier filters, in the present situation, are roughly 2 times larger (mostly due to constant factors) than the naive hashing sketch approach.

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

TABLE III
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 hashing sketch(resp. bloom filter 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.

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 Table III. The improvement is roughly 10% over the results of the pure hashing sketch approach presented in Table II.

IV. 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 [2]), 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 this approach can be improved by better sketches.

REFERENCES

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

[2] L. Veiga and P. Ferreira, "Asynchronous Complete Distributed Garbage Collection." in *19th International Parallel and Distributed Processing Symposium (IPDPS 2005)*. IEEE Computer Society, 2005.

[3] M. C. Lowry, "A new approach to the train algorithm for distributed garbage collection." Ph.D. dissertation, Adelaide University, 2004.

[4] S. E. Abdullahi and G. A. Ringwood, "Garbage collecting the internet: a survey of distributed garbage collection," *ACM Computing Surveys*, vol. 30, no. 3, pp. 330–373, 1998.

[5] M. Shapiro, F. L. Fessant, and P. Ferreira, "Recent Advances in Distributed Garbage Collection," in *Advances in Distributed Systems, Advanced Distributed Computing: From Algorithms to Systems*. London, UK: Springer-Verlag, 1999, pp. 104–126.

[6] F. L. 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*. New York, NY, USA: ACM Press, 2001, pp. 200–209.

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

[8] B. Liskov and R. 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*. New York, NY, USA: ACM Press, 1986, pp. 29–39.

[9] U. Maheshwari and B. Liskov, "Collecting cyclic distributed garbage by controlled migration," in *PODC'95: Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing*. New York, NY, USA: ACM Press, 1995, pp. 57–63.

[10] H. Rodrigues and R. Jones, "Cyclic Distributed Garbage Collection with Group Merger," in *ECCOP'98: Proceedings of the 12th European Conference on Object-Oriented Programming*. London, UK: Springer-Verlag, 1998, pp. 260–284.

[11] ——, "A Cyclic Distributed Garbage Collector for Network Objects," in *WDAG '96: Proceedings of the 10th International Workshop on Distributed Algorithms*. London, UK: Springer-Verlag, 1996, pp. 123–140.

[12] B. Lang, C. Queinnec, and J. Piquer, "Garbage collecting the world," in *POPL'92: Proceedings of the 19th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. New York, NY, USA: ACM Press, 1992, pp. 39–50.

[13] J. Hughes, "A distributed garbage collection algorithm," in *Proc. of a conference on Functional programming languages and computer architecture*. New York, NY, USA: Springer-Verlag New York, Inc., 1985, pp. 256–272.

[14] N. Richer and M. 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*. London, UK: Springer-Verlag, 2001, pp. 161–176.

[15] A. Potanin, J. Noble, M. Frean, and R. Biddle, "Scale-Free Geometry in OO Programs," *Communications of the ACM, Volume 48, Number 5*, May 2005.

[16] 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.

[17] S. Muthukrisnan, "Data streams: Algorithms ans applications," http://www.cs.rutgers.edu/~muthu/.

[18] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[19] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," in *Internet Mathematics*, vol. 1, 2003.

[20] M. Mitzenmacher, "Compressed bloom filters," *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, pp. 604–612, 2002.

[21] A. Pagh, R. Pagh, and S. S. Rao, "An optimal Bloom filter replacement," in *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2005, pp. 823–829.

[22] B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal, "The Bloomier filter: an efficient data structure for static support lookup tables," in *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2004, pp. 30–39.

[23] S. Cohen and Y. Matias, "Spectral bloom filters," in *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM Press, 2003, pp. 241–252.

[24] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[25] R. 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*, vol. 5, 1996, pp. 329–359.