Sketch based Distributed Garbage Collection, Theory and Empirical Elements

Joannès Vermorel

Sketch-based Distributed Garbage Collection, Theory and Empirical Elements

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

***Abstract*— Object-Oriented Programming (OOP) is a pillar of actual, and most probably future, software engineering. Within the advances for OOP, Garbage Collection (GC) is one of the major improvements that brought both large productivity and reliability benefits. Yet, to our knowledge, no widely used distributed systems benefit from a complete Distributed Garbage Collector (DGC) at this time. We believe that this situation is due, a least partly, to actual DGC performance issues.**

In this paper, we introduce the idea of sketch-based DGC where object-graph fragments are sketched rather than explicitly represented. We prove, under reasonable assumptions, that sketched messages are smaller up to one order of magnitude than their explicit counterparts. Those results apply to most of the state-of-art DGC methods. In the case of the Veiga and Ferreira DGC algorithm, the improvement is more than a factor 4 under very limited assumptions.

I. Introduction

Object-Oriented Programming (OOP) is a pillar of actual, and most probably future, software engineering. The foundation of OOP consists of considering that computer programs are collections of individual units called *objects* that communicate between each other rather than lists of instructions executed by the computer. Since OOP is omnipresent in modern software, this notion has been very naturally extended as Object-Oriented Distributed Programming (OODP) in the context of distributed systems. Widely used systems such as CORBA [1], Java RMI [2] or Microsoft .Net Framework [3] reflect the interest and the need for OODP. A key concept in OODP is the notion of *distributed objects* (the term “network object” is also found in the literature, see [4] for an extensive review of the topic).

Within the OOP approach, the Garbage Collection (GC) has been a major advance in terms of programmer’s productivity and program’s reliability [5]. A GC identifies *live* (or reachable) and *dead* (or unreachable) objects in the application object graph. The GC removes the burden of manual memory management from the developer to leave it to an automated and more reliable process. Since Distributed Garbage Collection (DGC) is the natural OODP equivalent of the Local Garbage Collection (LGC) in the context of distributed systems, it is natural to expect similar productivity and reliability benefits from the DGC when it comes to the design of distributed systems. Yet, DGCs are, at this time, far from being widely used, even available, components in OODP frameworks (see discussion below). We believe that this limited diffusion is mostly due to the large amount of challenging issues in the design of a DGC. Indeed if a large literature (see [6], [7], [8], [9] for topic review) is devoted to such algorithms, there are no, at this time, to our knowledge, strong DGC implementations comparable, in terms of performance, to the GC implementations commonly available.

The analysis of the performance of a DGC algorithm is a wide problem that includes many dimensions. Among “architectural” properties, asynchronicity and completeness are two highly desirable, yet very challenging properties for a DGC algorithm. The **asynchronicity** states that 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[6], [10], many algorithms are not and require a centralized architecture[11], [12], [13] or group synchronization[14], [15], [16] or consensus[17] 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 [18], and recent works seem to indicate that object-oriented designs tend to generate scale-free graphs [19] (scale-free graphs are cyclic with a very high probability). In this respect, it should be noted that early DGC methods were not complete [20], [21]. 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).

But the performance analysis of a DGC also includes resource consumption properties such as **memory, CPU and bandwidth overheads**. Additionally more subtle criterions such as the *non-disruptiveness* (no pause in the application) and the *promptness* (garbage should be promptly collected) should also be considered.

In respect to this list of desirable properties for DGC algorithms, we believe that the wide spectrum of Distributed Cycle Detection Algorithms (DCDAs) ([6], [10] for recent examples) constitutes the most scalable approach to tackle the DGC problem. To our knowledge, all cycle detection methods are relying on messages carrying subsets of object-graphs in one form or another. 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 totally decentralized fashion, we believe that 200bits is a reasonable lower bound estimation of the global identifier size for present and future

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	> 400
Globus	Java	URLs	> 500

large scale distributed systems[1]. Moreover, the ever-increasing adoption of web-services [23] for distributed systems (Globus [24] being one of the most well-known examples) involves even longer identifiers (500bits is a reasonable lower bound estimation for URL based identifiers) and accentuates even further this issue.

Large identifiers, if explicitly carried over the network, are an unavoidable hindering factor for large scale DGC working over bandwidth constrained network. It should also be noted that more bandwidth does necessarily solve the DGC bandwidth requirements. Indeed, more bandwidth allows: first, the design of finer-grained distributed objects (that simplifies the developer's life at the cost of a larger distributed object graph); second, more distributed object migrations (achieving a more effective load balancing at the cost of a more complex distributed object graph). Those two elements adversely impact the amount of work required from the DGC.

Therefore, the memory footprint of object identifiers calls for more compact representations. This point will be discussed more extensively in Section III but it should be noted that identifier compression is a weak solution to solve this problem. A **sketch** refers to an approximate compact representation of the data that fits certain pre-determined purposes. The sketching idea, at the heart of the data streams domain (see [25] for an introduction), states that keeping an explicit representation of the data is, sometimes, neither required nor efficient, when the underlying objective are only associated to some of the data properties. Many sketches have been introduced in the data stream literature. In this paper, we are focusing on *set approximations* whose most famous representatives are the *bloom filters*, originally introduced in [26]. Bloom filters have been used to solve a large variety of network related problems (although not for DGC to our knowledge), see [27] for a survey. Recent works on bloom filters include the study of compressed bloom filters [28], optimal replacements of bloom filters [29], enhanced bloom filter with lookup capabilities [30] or frequency estimations [31].

A. Our results

We introduce the idea of performing garbage 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 known DCDAs. In particular, we discuss extensively how the Veiga and Ferreira DCDA [6] can be improved through a sketch-based approach. For this particular example, the overall gain in term of overall network transmission is roughly a factor 4 under weak assumptions.

The theoretical elements to support our sketch-based approach are introduced in Section II. The application of those elements to the DGC tasks is discussed in Section III.

II. SKETCHING SETS OF IDENTIFIERS

A memory footprint of 200bits per global identifier is an hindering factor for any DGC algorithm relying on identifier transmissions. Yet identifier transmissions are at the core of most of the DGC algorithms (with the notable exception of object-migration approaches). Nevertheless, the explicit representation of identifiers is not a requirement (this point will be extensively discussed in Section III). Often, requirements on identifier sets are restricted to certain operations such as inclusion testing, item insertion, set equality testing, $\ldots$ In this section, we propose, through sketches, much more compact representations than the explicit extensive listing. Those sketches are *approximate* but the accuracy probability can be precisely quantified. The consequences of relying on possibly inexact identifier representations are very algorithm-dependent and therefore left to the Section III.

Since the discovery of hashing, decades ago, it has been known that sets can be represented in a more compact manner than explicit listing if a certain degree of approximation is tolerated. As we will see, even the very naive hashing sketch can be very efficient and appropriate in the DGC context. The hashing sketch is discussed in Section II-A. Although, the hashing sketch has been dramatically improved with the introduction of bloom filters (see [26]) that are known to be within a factor 1.44 of the optimal set representation size. The bloom filters are described in Section II-B. Nevertheless, the bloom filters present a few shortcomings for the purpose of DGC. Those shortcomings are addressed in Section II-C with the introduction of incremental bloom filters.

A. The hashing sketch

When items are large, a naive, yet efficient, approach to set sketching is simply to substitute a small hash to each item. The hashing sketch is not new, it has been used for decades (see [32]). This section introduces this structure with a perspective

[1] Considering the UUID standard [22], 128bits may seem a natural estimation for a global identifier size. But it should be noticed that the UUID standard assume that all actors (see Section 6 of [22]) 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 larger identifiers. 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, Mozart identifiers provides address resolution for example). In such case, 128bits IPv6 addresses will probably be much more adequate for spatial resolution, requiring again larger identifiers.

[2] See UUID discussion.

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

[4] Assume 32bits IPv4 addresses.

that will be useful to evaluate it's potential in the context of DGC.

Formally let E be the identifier space (simply called item space in the following). Let $S = \{s_1, \dots, s_n\} \subset E$ be a set. Let $h : E \rightarrow \{0,1\}^b$ 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 of S with $\mathbb{H} = \{h(s_1), \dots, h(s_n)\}$. 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 function is non-injective, 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 Section III) that the initial choice of b, the number of bits allocated per item determines the number of items that can ultimately be added to the sketch while 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 the insertion of the next item.*

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 equal to $2^b\epsilon$.*

Proof: Based on the Lemma 1, if N is a lower bound of the hashing sketch capacity, the capacity definition implies that $2^{-b}N < \epsilon$. This expression can be rewritten $N < 2^b\epsilon$. □

As we will see in the following section, the Lemma 1 is largely improved by the bloom filter sketch. Nevertheless, as we will see in Section III, the simplicity of the hashing sketch has many advantages for DGC purposes.

B. The bloom filter sketch

In this section, we analyze 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 [28]. Because of their complexities, those structures are beyond the scope of the present work.

We define[7] a bloom filter sketch $\mathbb{M} = (v_1, \dots, v_p)$ as a matrix comprising p bit-vectors of length q. Each bit-vectors is associated to a random hash function $h_i : E \mapsto \{1, \dots, q\}$ 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, to $\mathbb{M}_\emptyset = (0, \dots, 0)$. 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 element $x \in E$, is defined with $\mathbb{M}_x = (1_{i_1}, 1_{i_2}, \dots, 1_{i_p})$ where $i_k = h_k(x)$. The union-equivalent $\oplus$ operation is defined with

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

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

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

There is a possibility of false positive inclusion, the following lemma characterizes the FPR of the bloom filter sketch.

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 [27]. □

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 link this: with only 16bits per identifier, the sketch of 5 elements can accurately assert elements inclusion with a probability lower than $\frac{1}{1000}$. The memory footprint of the explicit listing is 1.000bits, assuming a reasonable 200bits per element, whereas the memory footprint of the bloom filter

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

[6] We do not exactly introduce the bloom filters as originally presented in [26]. Instead, we are presenting a variant (see [27]), that slightly improves the bloom filter performance but provides the same asymptotic bounds.

[7] In comparison, the original bloom filter is a single bit-vector of length m associated to k hash functions (taking the usual notations of the literature). The bit-matrix can be viewed as a single vector of length $m = p \times q$. In both cases, there are two parameters: the size of the structure and 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 [29] for recent work on that matter). Nevertheless, in the case of DGC, the total number of hash functions required in practice is very limited. Moreover hash functions can be pooled and reused. In practice, hash functions' memory costs are negligible in the case of DGC.

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

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

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

The design of a bloom filter sketch depends on the two parameters p and q. Nevertheless, as the Lemma 2 suggests, 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 [27] for the proof. □

The result of the Theorem 2 can be interpreted like this: for n items and b bits per item, 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. For a carefully initially designed bloom filter sketch, if $b > \ln(\frac{1}{\epsilon})/\ln(2)^2$ the ϵ-capacity of $\mathbb{M}$ is unbounded.*

Proof: Based on the result provided by the Theorem 2, the Definition 1 implies $2^{-b \ln(2)} < \epsilon$. Note that this expression is valid for any number N of items. This expression can be rewritten $b > \ln(\frac{1}{\epsilon})/\ln(2)^2$. □

The capacity bound of the bloom filter sketch is much better than hashing sketch's one because for a fixed amount of bits allocated per item, there is no limit to the number of items that can be inserted into the bloom filter sketch.

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 bloom filter sketch requires a full initialization before the insertion of the first item. 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 insertion, therefore a large initialization defeats the sketch compacity purpose. Therefore, the sketch size must *incrementally* increase when items are inserted. Such behavior is very natural for the hashing sketch but more troublesome for the bloom filter sketch. In this section, we introduce a general method to incrementally adapt the sketch size[10].

Formally, let us consider the sketch list

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

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 item 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

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

The choice of b, α and the base sketch type clearly defines the incremental sketch *capacity*. The following theorem quantifies its FPR.

Theorem 3 (FPR of incremental sketch) *Let $2^{-b\gamma}$ be an upper bound of the FPR of the non-incremental base sketch for b bits allocated per item. With b bits allocated per item, the incremental sketch provides a FPR lower than $2^{-b\gamma/\alpha} \log_\alpha(n)$ where n is the total number of items in the incremental sketch.*

Proof: In order to distinguish the number of bits allocated per item for the base sketch from its incremental counterpart, let b be the number of bits allocated per item for the *base* sketch and β the number of bits allocated per item for the *incremental* sketch. In terms of bit allocations for the incremental sketch, the worst case corresponds to the step where a k^{th} base sketch has just been added (this k^{th} base sketch is still empty). The number of bits allocated per item in the incremental sketch at this step can be expressed with

$$\begin{aligned} \beta &= \frac{\alpha^k m b + \sum_{i=0}^{k-1} \alpha^i m b}{\sum_{i=0}^{k-1} \alpha^i m} \\ &= \frac{\alpha^{k+1} - 1}{\alpha^k - 1} b \text{ using } \sum_{i=0}^{k} \alpha^i = \frac{\alpha^{k+1} - 1}{\alpha - 1} \\ &= \left(\alpha + \theta\left(\frac{1}{\alpha^k}\right)\right) b \end{aligned}$$

Therefore, for an initial allocation of b bits per item, the incremental sketch requires $\beta = (\alpha + o(1))b$ in the worst case. In the following, for the sake of simplicity, we will consider $\beta \approx \alpha b$ since it is asymptotically equivalent. Let f_i be the FPR of the i^{th} base sketch. The FPR of the incremental sketch is defined by $1 - \prod_{i=0}^{k-1} (1 - f_i)$. Since we assume that the FPR of the base sketch is upper bounded by $2^{-b\gamma}$, the FPR f of

[10] We are not aware of earlier introduction of the incremental bloom filter sketch. Yet, bloom filters have been extensively studied for 3 decades and this structure may have been already used (possibly left unpublished).

the incremental sketch can be bounded with

$$\begin{aligned} f &= 1 - \prod_{i=0}^{k} (1 - f_i) \\ &\leq \sum_{i=0}^{k-1} f_i = 2^{-b\gamma} k \\ &\leq 2^{-b\gamma} \log_\alpha(n) \approx 2^{-\beta\gamma/\alpha} \log_\alpha(n) \end{aligned}$$

The third line is justified by the fact that the number of items added to the incremental sketch can be expressed $n = \sum_{i=0}^{k-1} \alpha^i m = \frac{\alpha^k - 1}{\alpha - 1} m$. Thus $n \geq \alpha^k + o(1)$ for any $m > 1$, and finally $\log_\alpha(n) \geq k$. □

Numerical illustration: considering an incremental bloom filter sketch with $\alpha = 1.1$ and allocating no more than 20bits per item, we can incrementally add 1000 items with a FPR staying below 0.012 at all time.

Corollary 2 (ϵ-capacity of the incremental sketch) *Let $\mathbb{S}^*$ be an incremental sketch with b bits allocated per item. Let N be a lower bound on the ϵ-capacity of $\mathbb{S}^*$, we have*

$$\log_\alpha(N) < 2^{b\gamma/\alpha} \epsilon$$

Numerical illustration: considering an incremental bloom filter sketch with $\alpha = 1.1$, allocating 32bits per item and with a FPR lower than $1/1000$, the capacity of the sketch is greater than 10^{48}.

Asymptotically, the ϵ-capacity of 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 the hashing sketch capacity for any $b > 20$.

III. Sketch-based DGC

In this section, we will see how the sketches previously introduced can be used to improve DGC performance.

Before, digging into sketch-based DGC, let us discuss why compression is a weak solution for identifier transport. A simple experiment illustrates the problem: the `GZip` compression of a stream built from the concatenation of 1000 UUID identifiers (generated on the same machine and therefore including an intrinsic redundancy due to identical spatial resolution) produces a compressed stream of length roughly equal to 24Kbyte, whereas the base (uncompressed) stream is exactly 16Kbyte long[11]. In this example, an usual compression scheme does not improve the situation but, on the contrary, makes it worse. This situation is essentially due to the intrinsic nature of object identifiers that are precisely designed to avoid collision. Indeed the pseudo-random mechanisms used to generate object identifiers make them "resistant" to compression schemes that exploit information redundancy.

We will first introduce a very brief description of the Veiga & Ferreira DCDA in Section III-A. This description is followed by the introduction of an equivalent sketched Cycle Detection Message (CDM) based on the hashing sketch in Section III-B. The performance of this first approach is detailed in Section III-C. We refine this sketched CDM in Section III-D with the incremental bloom filter sketch. Finally the practical implementation of such a DGC is detailed in Section III-E.

A. Veiga & Ferreira DCDA

Veiga and Ferreira present in [6] an asynchronous, complete DGC. Intuitively the DCDA proceeds by an initial candidate selection (i.e. selecting an object estimated as probable garbage) that is followed by a sequence of CDMs send between the processes. CDMs are equivalent to the `graph_summary` here below (note that this data structure is already optimized to avoid the duplication of identifiers). In this section, we propose to leverage the compactness of sketched CDMs to quickly detect garbage cycles (with a risk of false positive detection) and to verify the detection correctness afterwards through explicit CDMs. This approach has virtually no impact on the DCDA, only the CDM encoding is really 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[12] 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 of the summary timestamp and another 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 equal to 200bits (see discussion in Section I). The two flags require naturally 2bits. In [6], 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 of 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` using the hashing sketch previously introduced. The CDM becomes

```
type graph_sketch = {
```

[11] The experiment has been performed under Microsoft .Net 2.0 using the class `System.Guid` as a base implementation for the 128bits UUID standard and the class `System.IO.Compression.GZipStream` as `GZip` implementation.

[12] 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 target.

```
    list of (
        hash,
        is_source_flag, is_target_flag);
}
```

The proposed graph_sketch 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) No change.

Since the hashing sketch has no false negative, it can easily be proved that the detection errors caused by the graph_sketch are restricted to false positive cycle detections. Therefore we propose the 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[13]. The proof is that this mixed strategy is *correct* (i.e. all garbage but only garbage is collected) is simple, but the detail is out the scope of this paper.

C. Sketched DCDA performance analysis

The per-item transportation cost of the explicit CDMs can be straightforwardly estimated as $c_e = 266$bits per item (see discussion here above). In order, to estimate the relative interest of the sketch-based approach, we need an estimation of this cost when sketched and explicit CDMs 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 systems 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 involves a majority of failures of DCDA initiatives. This belief is motivated by the results of [18] concerning the object age frequency distribution (no simple behavior seems to govern the object-lifetime). Additionally, the availability of much cheaper CDMs is, itself, a strong bias in the estimation of P_g. Indeed, the cheaper the DCDA is, the more detection initiatives can be started for a given amount of network bandwidth dedicated to the DGC. Therefore cheaper CDMs enable the improvement of both 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. In the Veiga & Ferreira DCDA, timestamps are used to prevent race conditions with local mutators 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[14] to have any noticeable impact on performance in practice. Indeed, a race condition involves a complicated root displacement that must occur 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 that is one or two orders of magnitude lower than that). Therefore, in the following, those events will simply be ignored.

Let P_n be the *a priori* probability for a DCDA initiative that the graph summary reaches n items at a point of its execution. As empirical measurements of scale-free graphs suggest (we are considering the measurements of the size frequencies of strongly connected components in [18]), small graphs are more frequent than large ones. In the present analysis, the values of P_n are important because they will be used in practice to determine the initial capacity (see Definition 1) of the sketch.

Let ϵ be an upper bound of the false positive cycle detections due to the FPR of the hashing sketch. A false positive cycle detection can occur because one or more hash collision in the graph_sketch (recall: a cycle is detected if all identifiers are flagged both as "source" and "target"). For the sketched CDM, the worst case corresponds to the situation where the cycle detection should fail because of a single unflagged identifier. In such a case, we have ϵ lower or equal to the FPR of the hashing sketch. Although, the worst case here is an *adverse* assumption compared to the average case where the cycle detection fails because of multiple unflagged identifiers. we will consider $\epsilon = 2^{-b}n$ (see Lemma 1) for the sake of simplicity in the following.

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 cost c_m for the whole mixed strategy can be estimated with $c_m = b + 2 + (1 - (1 - P_g)(1 - \epsilon)) * c_e$ (sum of the sketch 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 is. The results are limited to a capacity of 5.10^6 items. This value may seem a bit low, but two factors must be taken into account in the interpretation. First, it should be noted that distributed objects are typically "heavy" data-structures handling at least hundreds, usually

[13] Furthermore, it is possible to exploit the information of the sketched CDM in order to speed up the explicit DCDA execution. Indeed, the graph exploration performed by the Veiga & Ferreira DCDA can be pruned considering that items that are not contained in the hashing sketch cannot be part of the cycle of garbage. Since the hashing sketch has no false negative, the correctness of the DCDA would not be affected. Nevertheless, the approach requires substantial modifications of the DCDA which go beyond the scope of this paper.

[14] 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. In the very preliminary experiments performed with NGRID (see Section III-E), we have never observed so far any race condition leading to a timestamp mismatch.

TABLE II

Per-item costs of hashing sketch-based CDM.

b_L	ϵ	capacity	c_m
10	0.009	10	41
14	0.006	100	44
17	0.008	1.000	48
20	0.010	10.000	51
24	0.006	100.000	54
27	0.007	1.000.000	58
32	0.011	5.000.000	64

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 per-item CDM footprint in bits.

thousands, of regular objects. Second, the graph summary is already, as the name suggests, a “summary” of the underlying object graph. In [6], the authors estimate the graph summary to be one or two orders of magnitude smaller than the original graph.

D. Improved sketched DCDA with bloom filters

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_sketch` [15] by using a mix of the hashing sketch and the bloom filter sketch. Intuitively, we propose to store all single-flagged identifiers into 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 that in case of random insertions of items either flagged as “source” or “target”, we have $P_2 = \frac{2}{3}$.

Lemma 3 (Fully flagged item ratio for a cyclic graph) *Let G be an arbitrary cyclic graph with n vertices. Let assume that we have an algorithm $\mathcal{A}$ that explores this graph, and that visits each vertex exactly twice. Let assume that at each step of the algorithm $\mathcal{A}$, a vertex v is visited for the first time or for the second time. Let $V_{1,k}$ (resp. $V_{2,k}$) be the set of vertices visited once (resp. twice). Then, considering a random exploration, we have*

$$\lim_{n\to\infty} \frac{E\left[|V_1 \cap V_2|\right]}{E\left[|V_1 \cup V_2|\right]} = \frac{2}{3}$$

TABLE III

Per-item costs of hashing sketch and bloom filter sketch mixed CDM.

b_L	b_M	ϵ	capacity	c_m
14	18	0.007	10	46
16	20	0.007	100	48
18	22	0.005	1.000	50
22	22	0.007	10.000	51
24	22	0.009	100.000	53
26	23	0.014	1.000.000	54
29	24	0.007	5.000.000	55

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 per-item CDM footprint in bits.

Proof: Let P_k be the probability at step k that a vertex, visited at least once, has been visited twice. This probability can be expressed with $P_k = \frac{k-1}{2n-1}$ for any $k > 0$. Therefore, the “visited twice” ratio can be expressed with

$$\frac{\sum_{k=0}^{2n} kP_k}{\sum_{k=0}^{2n} k}$$

Note that we are weighting the probability sum with the cardinal of $V_1 \cup V_2$ (we have $|V_1 \cup V_2| = k$). This choice reflects that the “weight” of the step is proportional to the overall size of V_1 plus V_2. Considering $\sum_{i=0}^{n} i = 1/2(n+1)^2 - 1/2n - 1/2$ and $\sum_{i=0}^{n} i^2 = 1/3(n+1)^3 - 1/2(n+1)^2 + 1/6n + 1/6$, this ratio can be rewritten

$$\frac{1/3(2n+1)^3 - (2n+1)^2 + 4/3n + 2/3}{(2n-1)(1/2(2n+1)^2 - n - 1/2)}$$

Considering the terms in n^3, the limit of this expression when n tends to infinity is clearly $\frac{2}{3}$. □

This result can be straightforwardly interpreted as $P_2 \leq \frac{2}{3}$. In the following, we assume $P_2 = \frac{2}{3}$ for the sake of simplicity, although we believe this estimate to be quite low. Indeed, random insertions correspond to a bad situation (it’s not the worst case though) whereas the DCDA execution is highly biased in our favor (i.e. higher values for P_2) because the items flagged as “target” are explored first by the algorithm. The numerical results, obtained with those assumptions, are gathered in Table III. The improvement is roughly 15% over the results of the pure hashing sketch approach presented in Table II.

E. Practical implementation of sketched DCDAs

The sketch-based variant of the Veiga & Ferreira presented in Section III-B has been implemented in `C#` and is available as a part of the NGrid project[16]. Based on the results gathered in Table II and practical considerations, the number of bits allocated per item has been set to 32, thus the hashing sketch relies on unsigned integers that can be efficiently managed.

[15] 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 [30]. Unfortunately, bloomier filters, in the present situation, are roughly 2 times larger (mostly due to constant factors) than the naive hashing sketch approach.

[16] The NGrid project is a LGPL open source distributed computing framework. See `http://ngrid.sourceforge.net`.

Yet, due to the very preliminary nature of the present work and the very careful and precise methodology required to get any significant empirical DGC evaluations (and therefore more space than we can afford here), empirical results obtained with NGRID have not been included in this paper.

IV. CONCLUSION AND FUTURE DIRECTIONS

OODP, although widely adopted for the design of distributed systems, is still usually lacking the GC benefits that have been available for years for non-distributed applications. We believe that the large footprint of distributed object identifiers, estimated as more than 200 bits per identifier, is an hindering performance factor for DGC implementations.

In this paper, we have introduced the idea of an approximate representation of object identifiers through sketches for the specific purpose of DGC. Considering one of the more recent DGC at this time (see [6]), the improvement brought by the sketched CDM is roughly a factor 4 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 DGCs. Moreover, due to the preliminary nature of this work, it's also probable that our sketch-based approach can be improved by better sketches.

DGC is far from being the sole potential application of the sketch-based approach for the purpose of OODP. In the context of distributed objects, load balancing algorithms also rely on object graph transmissions which make them good candidates for the sketch-based approach.

V. ACKNOWLEDGEMENTS

I would like to thanks Jean-Philippe Vert and Franck Cappello for their advice and their guidance that have made this work possible.

REFERENCES

[1] "Common Object Request Broker Architecture (CORBA), Wikipedia," http://en.wikipedia.org/wiki/CORBA.

[2] "Java Remote Method Invocation (RMI)), Wikipedia," http://en.wikipedia.org/wiki/Java_RMI.

[3] "Microsoft .Net Framework, Wikipedia," http://en.wikipedia.org/wiki/Microsoft_.NET_Framework.

[4] D. Caromel and L. Henrio, *A Theory of Distributed Objects*. Springer-Verlag, 2005.

[5] 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.

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

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] ——, "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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] A. Birell, D. Evers, G. Nelson, S. Owicki, and E. Wobber, "Distributed Garbage collection for Network Objects," digital - Systems Research Center, Palo Alto, California, United States of America, Technical Report 116, Dec. 1993.

[21] D. I. Bevan, "Distributed garbage collection using reference counting," in *Volume II: Parallel Languages on PARLE: Parallel Architectures and Languages Europe*. London, UK: Springer-Verlag, 1987, pp. 176–187.

[22] 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.

[23] "Web Services activity, World Wide Web Consortium," http://www.w3.org/2002/ws/.

[24] "The Globus Alliance," http://www.globus.org.

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

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

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

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

[29] 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.

[30] 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.

[31] 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.

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