The Memory Behavior of the WWW, or The WWW Considered as a Persistent Store

Nicolas Richer and Marc Shapiro

INRIA - SOR group, BP. 105,
78153 Le Chesnay Cedex, France
{Nicolas.Richer, Marc.Shapiro}@inria.fr
http://www-sor.inria.fr/

Abstract. This paper presents the performance evaluation of five memory allocation strategies for the PerDiS Persistent Distributed Object store in the context of allocating two different web sites in the store. The evaluation was conducted using (i) a web gathering tool, to log the web objects graph, and (ii) a PerDiS memory simulator that implements the different allocation strategies. Our results show that for all the strategies and parameters we have evaluated, reference and cycle locality are quiet poor. The best policy seems to be first sequential fits. Results are linear with the size of the garbage collection Unit (a bunch). There is no clear optimum, but 64K to 128K appear to be good choices.

1 Introduction

PerDiS [FSB+98] is a new technology for sharing information over the Internet: Its abstraction is a Persistent Distributed Object Store. PerDiS provide a simple, efficient and safe abstraction to the application programmer. To attain this goal, it is necessary to understand the way target applications use the persistent store and how the different mechanisms and heuristics perform. This study is the target of task T.C 3 of the PerDiS project. This paper presents the results we have obtained.

The PerDiS persistent store is logically divided into clusters. A cluster is the naming and security unit visible by the programmer. A cluster is attached to a "home" site[1] but can be accessed from any site. From the garbage collection point of view, cluster is further subdivided into bunches, i.e. garbage collection Units. The Garbage Collector always runs on whole bunch and this unit will be completely replicated at one site in order for the GC to run on[2]. This means also that bunches will be stored in a contiguous area in memory. A reference inside a bunch is cheaper than between two bunches (an inter-bunch reference). The latter needs using stubs and scions. The fewer references will be inter-bunch, the better the PerDiS system will perform.

[1] The home site ensures data security and reliability.

[2] For latency reasons, bunches could be divided in pages but the page forming a bunch should be fully located at a site for the Garbage Collection to operate on.

G.N.C. Kirby, A. Dearle, and D.I.K. Sjøberg (Eds.): POS-9, LNCS 2135, pp. 161–176, 2001.

1.1 What Are Garbage Collection Components

A Garbage Collection components is a set of objects reclaimed by the garbage collector all at the same time. This could be a single object or several objects link together in a way that they become unreachable all at the same time. Typically, this can be a set of objects with circular references between them.

In this context, Strongly Connected component is an approximation for Garbage Collection component. Rigorously, Garbage Collection components are Strongly Connected component plus all the objects reachable only from this components, but we don't make the difference in this study, because, unfortunately, we are not able to extract real Garbage Collection components from an object graph yet. This approximation may grow up the overall proportion of objects and bytes included in Garbage Collection components. Studying Strongly Connected component provide only minimum proportions, and we don't currently know the maximums.

Note also that Garbage Collection components are often called cycles in this paper because it's shorter.

1.2 The PerDiS Garbage Collector

The PerDiS Garbage Collector use a reference counting algorithm to handle inter-bunches references, this to avoid costly distributed Garbage Collection Algorithms. In consequences, some garbage could not be reclaimed in some situation. This is the case of Garbage Collection components composed by several objects spread between different bunches. This kind of garbage could be reclaimed on the of all the bunch is containing the objects were run located at the same site. This is, unfortunately, the drawback of using reference counting.

The assumption we have made during the design of the PerDiS Garbage Collector is that such kind of unreclaimable components are sufficiently rare to be forgotten and we don't put any kind of global tracing collector for them. Of course, this assumption should be verified in practice and this is precisely why we are interested by Garbage Collection components in this study. Depending of our experimental results, the need for a global tracing collector mechanism will be reconsidered. The important point here is the fewer cycles are inter-bunches, the less garbage will remain unreclaimable. This suggest that object placement strategies (i.e. allocation algorithms) should minimize inter-bunch Garbage Collection components and inter-bunch references. A secondary goal will be to minimize also the number of scions and store fragmentation.

In the current PerDiS implementation, the initial allocator clustering is permanent, because PerDiS support the uncooperative C++ language that does not provide natively all the informations required to perform object relocation. In the future, we plan to implement object relocation in PerDiS using type information extracted by `typedesc` [Sal99], a tool we have specifically implemented for that.

A complete description of the PerDiS Garbage Collection Algorithm is available in [FS94,FS96]. [BFS98] describe more deeply the implementation.