Implementing a Family of Distributed Garbage Collectors

Stuart Norcross, Ron Morrison,

School of Computer Science
University of St Andrews
North Haugh, St Andrews, Fife, Scotland.
stuart, ron@dcs.st-and.ac.uk

Dave Munro, Henry Detmold, Katrina Falkner

School of Computer Science
University of Adelaide
Adelaide, South Australia.
dave, henry, katrina@cs.adelaide.edu.au

This paper discusses implementations of distributed garbage collectors derived using a previously developed methodology which involves mappings of distributed termination detection algorithms (DTAs) to local garbage collection schemes. Implementations produced by such mappings preserve the safety and completeness properties of the original local collectors. Through our collector implementations we have come to understand that the derivation technique extends to distributed collection schemes with heterogeneous local collector behaviour. Our contribution, reported here, is the construction of an experimental platform, implementations of the Task Balancing DTA, an extension to the derivation methodology that minimises constraints on local collectors, together with three new mappings and their implementations.

Keywords: Garbage Collection, Distributed Termination.

ACM Classification: C.2.4 (Computer-Communication Networks – Distributed Systems), D.4.2 (Operating Systems – Storage Management)

1. INTRODUCTION

We demonstrate the practical application of the distributed garbage collector derivation methodology from Blackburn, Hudson *et al* (2001) and expand on the benefits gained from its use. As stated,

> *"Our system model is fairly simple. It consists of a number of sites, where computation proceeds asynchronously at each site. Sites communicate (only) by sending messages to one another. We assume that sites do not fail, i.e., they do not crash, stop, or act improperly, and that each message sent arrives at some later time, exactly once and uncorrupted."*

As originally formulated, the methodology starts by making a local collector concurrent and then mapping a DTA onto the resultant collector to provide a distributed garbage collection scheme. This paper extends the methodology to minimise the constraints on the local collector. This is achieved by mapping a DTA onto any (non-distributed) garbage collection scheme, whilst

Manuscript received: 30 June 2003
Communicating Editor: Amos Omondi

preserving the important properties (in particular safety and completeness) of the collector. Each such mapping is used in defining a set of club rules that must be obeyed by each participant (site) in the distributed collection scheme. The participating collectors are free to perform any local actions as long as they preserve the club rules. The benefit of such a structured approach to distributed collector implementation is the clear distinction between providing safety via termination and the mechanics of space reclamation.

We present three new mappings that constitute club rules within which six distinct distributed collection schemes are described. These mappings use implementations of the Task Balancing DTA on the ProcessBase distributed cache from Norcross, Falkner *et al* (2001) which is itself an instantiation of the system model. Our final contribution is to show that once the club rules are defined it is possible to vary the collection schemes at participant sites thereby introducing a degree of heterogeneity. We postulate that future work extends to deriving club rules operating over existing local collectors.

2. TERMINATION AND TASK BALANCING

The distributed termination detection problem can be stated as follows. A distributed computation, known as a job, executes across a number of sites. A job consists of a number of dynamically spawned tasks. A task executes (or runs) at a single site and only sites with a running task can spawn new tasks; these can run locally or may be sent to a remote site. Within the context of a job j:

- a site with a running task of j is said to be active for j whereas a site with no running tasks of j is said to be passive for j;
- a site can transform from the active state to the passive state for j spontaneously, through the completion of all tasks of j running at the site but can only transform from passive to active for j through receipt of a task of j from a remote site;
- a job j is said to be terminated when each site is passive for j and there are no tasks of j in-flight between sites.

The *terminated* condition is a globally stable state of the distributed computation. That is, once the *terminated* condition is satisfied it remains so. It is the goal of the distributed termination algorithm to detect termination.

The Task Balancing (TB) DTA from Blackburn, Hudson *et al* (2001) operates by balancing counts of the tasks sent (*sent*) between sites and (separately) the number of tasks received and subsequently completed (*received/completed*) at each site. To achieve this each site must maintain two values, the value $sent_S$(j,T) which represents the number of tasks of job j *sent* from site S to site T, and the value $received_S$ (j) which records the number of tasks of job j received at site S. A site must also be able to calculate the number of *received/completed* tasks at a given time.

The algorithm requires that a single site be identified as the home site (usually the initiating site) for a given job. The home site is responsible for detecting termination by balancing the *sent* and *received/completed* counts for sites that hold (or held) a task of the job. Progress towards termination detection is made by a remote site S sending, at an appropriate time, to the home site H of job j, an *update* message containing the current *received/completed* and *sent* counts for j at S.

An *update* message for site S and job j is defined as:

- update(j,C,RC_S(j)) where
  - $C = \{<T, sent_S(j, T)> \mid sent_S(j, T) \neq 0\}$ and
  - RC_S(j) is the number of tasks of j *received/completed* at S.

An update message can be sent at any time, at site S and for job j, when either of the following conditions are satisfied.

- All locally spawned tasks of job j at site S have completed.
- There exists, at site S, uncompleted locally spawned tasks of job j and at least one uncompleted received task of j.

On sending an update message a site sets its *sent* counts to zero and decrements its $received_S$ (j) value by the *received/completed* count in the update.

It is the job of the home site H to detect termination. It does this by maintaining the value *count*(j,S) for each site S. On receipt of an *update* message from S, the value RC_S(j) is deducted from *count*(j,S) and for each site T, the value $sent_S$ (j, T) (in the *update* message) is added to *count*(j,T). Testing for termination can occur at any time and the termination condition is satisfied when
$\forall_T.count$(j,T) = 0. Note that we assume that the update messages sent from the home site H to itself are instantaneous.

The *home* site is required to process update messages in the order they are received. This can be achieved through ordered delivery in the channel, between a site and a job's home site, in which *update* messages are sent or by ordering information in the *update* messages themselves. A consequence of ordered delivery is that counts in *update* messages for tasks of a job j from a site S sent to the *home* site from S can be ignored (by the *home* site) since H will receive these tasks before the update message arrives. A fuller description of the algorithm is given in Blackburn, Hudson *et al* (2001).

3. THE EXPERIMENTAL PLATFORM

The platform for our distributed garbage collection experiments is the distributed ProcessBase cache. ProcessBase (Morrison, Balasubramaniam *et al,* 1999) is one of a family of languages designed to support process modelling. The ProcessBase language provides a number of key features such as:

- strong typing with an emphasis on static checking;
- type completeness;
- first class procedures;
- an infinite union type with dynamic projection;
- threads;
- distribution.

The ProcessBase system consists of the language and its object-based runtime environment. That is, in running ProcessBase code (in the form of threads) the execution engine manipulates objects. A ProcessBase program consists of a single computation composed of one or more threads of execution operating in a shared namespace. In the distributed ProcessBase system the shared namespace is mapped across a number of sites. Many different mappings are possible. Here we describe one such mapping and its implementation.

3.1 Distributed ProcessBase

The model of computation for the distributed ProcessBase system is shown in Figure 1 below. A distributed ProcessBase computation is composed of multiple thread closures (T), which execute within a single shared name space on a number of distributed sites (*Site*). The collection of sites is considered a distributed virtual machine (VM) on which a distributed ProcessBase program executes. A site of the distributed VM may hold a number of threads and a thread is resident on only one site at a given time. Each site implements a local object cache (*Cache*) that holds objects containing both executable code and data of the running ProcessBase program.

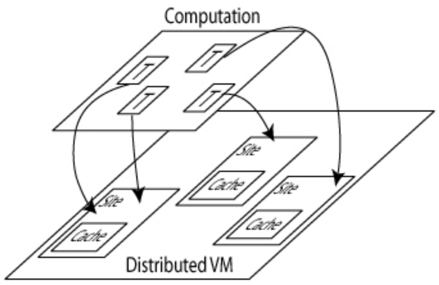


Figure 1: ProcessBase Computational Model

The distributed computation begins with a single thread executing on one site. As this thread executes it can spawn thread closures that are run locally or are exported to a remote site, thus distributing the computation. The placement of newly formed thread closures can be controlled by an application-level scheduling policy. Although the specific location of threads is not of importance in our model, it is relevant that the application is able to signify where relationships between threads and data exist that may affect placement decisions. Sites communicate only through message passing and a communications channel, providing guaranteed ordered delivery, is maintained between each site.

The distributed ProcessBase system provides a platform for multiple areas of experimentation. We are conducting experiments not only into suitable garbage collection algorithms for distributed systems, but also in the areas of distributed shared memory coherency protocol design and flexible distribution architectures.

3.1.1 Addressing Mechanisms

The shared namespace of a distributed ProcessBase program is mapped onto a distributed graph of objects. Inter-site references between these objects are represented by two part distributed addresses of the form

<*site, local id*>.

The local id part is symbolic and provides one level of indirection for object addresses (representing references between objects within the distributed graph) that allows for independent relocation of objects at a site. We will see later that this is helpful in the implementation for heterogeneous local collectors. Addresses that are entirely local to a site (i.e. that represent references between objects on the same site) are optimised to omit the site part of the address. In such addresses, the local id is no longer symbolic but is instead the local cache address (CA) for the object at that site.

A thread running at a site may create objects in its local cache during execution. A thread closure exported from site *S* to site *T* will result in *T* obtaining references to objects local at *S*. On export, cache addresses are translated to distributed addresses. Such references are known as remote references at T.

To implement the distributed address (DA) mechanism each site maintains a distributed address to cache address translation table. This table maps the symbolic (*local id*) part of an object's DA to its cache address, or CA, at the local site, and is called the $DA_{sym} \rightarrow CA$ translation table. When a reference from site S is exported to site T, a DA is constructed and added to the $DA_{sym} \rightarrow CA$ translation table at S. If the object is moved at S (i.e. given a different CA) the $DA_{sym} \rightarrow CA$ table entry is updated.

3.1.2 Object Duplication

The system as described here is neutral towards cache coherency policy and program synchronisation mechanism since the garbage collectors operate over a graph of objects irrespective of these other sub-systems. It is possible that the combination of the garbage collector, coherency policy and synchronisation mechanisms will be more efficient (Munro, Falkner *et al*, 2001) and we see the possibility of extending the club rules to accommodate this as future work.

To accommodate different coherency policies the system allows for object duplication. A site that holds a remote reference to an object can request a copy of the object from its creator site. Such a copied object is known as *remote resident*. In support of this each site maintains a second address table, called the DA → CA address translation table, which maps from the DA of a *remote resident* to its CA in the local cache. On discovery of a DA a site can inspect the DA → CA table to see if it already holds a copy of the object and thereby locate it in the local cache. Table entries are added on object import and (as with the $DA_{sym} \rightarrow CA$ translation table) updated as copied objects are moved within the local cache. Note that this table does not constitute a root set for local collection. Table entries are removed when copied objects are reclaimed by a site's local cache collector.

In order to validate the neutrality of our system towards cache coherency protocol, we have developed and implemented several commonly used protocols. This exploration serves to support our claim, but also provides a suitable platform for investigation into potential combinations of coherency protocol and collection algorithm. Our experiments include the development of multiple coherency schemes, such as a simplified non-duplicating scheme, an invalidate-based sequential consistency model and several schemes based on release consistency. In a release consistency model, writes to mutable objects require explicit object locking, and hence explicit messages are required to maintain object locking and object consistency which increases the message complexity of the system. Our implementations include eager and lazy release consistency models, with update (rather than an invalidate) models, and with the assumption of a single writer. The site component of the distributed address structure can be used to support home-based coherency protocols.

3.1.3 Implementation

The distributed ProcessBase system is implemented on a Linux based Beowulf cluster (Becker, Sterling *et al*, 1995.). A site of the distributed VM corresponds to a node of the network and communications channels between sites are provided by full-duplex TCP streams.

4. FORMING THE CLUB RULES

The methodology for implementing distributed garbage collectors consists of mapping a distributed termination algorithm onto a non-distributed garbage collector. This is done by identifying the parts of the non-distributed collector that require distributed termination detection, mapping this to a DTA, in our case Task Balancing, and separating it from the actions necessary to implement collection. The club rules at each site are thus the combination of the implementation of the DTA and the collection actions. Using new mappings we will show how the club rules are constructed for six examples: distributed mark-sweep, distributed reference counting and distributed generational collectors each with homogeneous and heterogeneous local collectors. The three distributed collectors implement the same DTA mapping but differ in the local actions required thereby yielding a different set of club rules.

4.1 Distributed Mark-Sweep Collection

A typical mark-sweep scheme, whether it be stop-the-world or concurrent, is composed of two phases; a mark phase followed by a sweep. In the mark phase the graph of objects is traced from a root set marking reachable objects. A sweep of the whole space is then required to identify unmarked (unreachable) objects. Often collectors take the opportunity during the sweep phase to unmark reachable objects and relocate objects to compact the free space.

To implement distributed mark sweep we identify the mark phase as the part that requires distributed termination. The club rules are thus an implementation of the DTA at each site plus the actions necessary to perform the sweep phase.

In our implementations each site has a distinguished root object from which all locally reachable objects may be found. The root set for a distributed collection is determined by the union of all these local roots. Our distributed garbage collector operates as follows. Garbage collection starts by sending a *mark* message to all sites. To avoid global synchronisation we assume, for the moment, that a single predetermined site is charged with the responsibility for starting collection. Each site then traces the local object graph from its root, marking all reachable local objects. If a DA is encountered during tracing a message is sent to the DA's site to mark all objects reachable from the DA. In this way objects that are referred to remotely are also marked during this phase. The initiating site detects when marking is complete (DTA termination) at which point it sends a sweep message to all sites. On receipt of the sweep message the local site in the homogeneous scheme identifies and collects unmarked (unreachable objects) immediately. In the heterogeneous case, the sweep action may delegate the collection of objects to a local collector. In either case, the local sites inform the co-ordinator when the sweep is complete to allow subsequent distributed collections. This mechanism is similar to Plainfosse and Shapiro's (1995) description of a generic distributed mark-sweep.

In terms of mapping the TB DTA there is one job that corresponds to the distributed marking phase, called the distributed marking job (DMJ). The site that initiates the job is called the DMJ *home* site. A job consists of two types of tasks, the first is called a Root Marking Task (RMT) and the second is called a Distributed Address Marking Task (DAMT). When the DMJ *home* site starts a job, it sends a RMT to every site (including itself).

Both task types trace the local graph of objects from a given start point. Each object at a site has a distributed mark bit (DMB) associated with it. As an object is traced by a task its DMB is set (to indicate that it is marked) and then it is scanned for references. Both types of task complete when they have fully traced the local graph from their specified start point.

Unlike Hughes' collector (Hughes, 1985), we do not assume instantaneous message passing and so our system must maintain safety even in the presence of DAs that are in-flight between sites when marking begins. The following example demonstrates how such in-flight DAs can cause problems. A site S holds the only reference to an object *O* on site R. S sends the DA of *O* to a site T and immediately deletes its own copy. Distributed collection may begin at S and T before the message containing the DA arrives at T and the distributed marking mechanism will incorrectly determine that *O* was unreferenced.

This problem is solved by having sites record any DA sent to a remote site in a table called the *in-flight* table. Each site is required to send an acknowledgement to the sender site on receipt of any message containing a DA. On receiving the acknowledgement a site can then remove the table entries for the DAs in the original message. All entries in the *in-flight* table are treated as roots for the distributed collection.

4.1.1 Club Rules for Distributed Mark-Sweep

The following describes the club rules that are generic to both distributed mark-sweep collectors. For site S that is not the DMJ *home* site:

- S maintains data structures for recording TB *sent* counts. These counts are maintained as follows. Each site S maintains a *sent* count array with an element for each site T recording $sent_S(T)$. When a task is sent to site T, $sent_S(T)$ is incremented. An update is sent on the completion of each received task, thus RC_S is always 1 and therefore no received count is required. When the update is sent, for each site T, $sent_S(T)$ is set to zero. Note that RMT and DAMT task counts may be combined here.
- S maintains an *in-flight table* where all DAs sent in messages to remote sites are recorded.
- On receipt of a message, from a site T, containing DAs, S must send an acknowledgment message back to T. The acknowledgment contains the DAs that were sent to S from T.
- When S receives an acknowledgement it removes the *in-flight* table entries for each DA in the message.
- Implementation of an RMT task at S is as follows. Mutator activity is paused at S during the execution of an RMT. An RMT traces the object graph from the distinguished local root at S marking reachable objects using their DMB. For each DA found during the trace a DAMT is generated and sent to the site as determined by the site address component of the DA. An obvious optimisation is to ensure that only one DAMT task is sent for each distinct remote DA at a site. A DAMT is also sent for each DA in the *in-flight table*. On completion of an RMT a TB update is generated.
- Implementation of a DAMT is as follows. Mutator activity is paused at S during the execution of a DAMT. A DAMT message contains the DA of an object that is remotely referenced. The local object graph is traced from this object marking reachable objects using their DMB. For each DA found a DAMT is sent to the remote site (as determined by the site address component of the DA). On completion of a DAMT a TB update is generated and sent to the *home* site.
- When a DAMT is sent from S to a remote site T the *sent* count for T at S, $sent_S(T)$, is incremented.
- On receipt of a *sweep* message the behaviour of a local site is implementation dependent. The actions taken for the homogeneous and heterogeneous collectors are described in the following sections. However, all implementations share some common characteristics; mutator activity at S is paused while the sweep executes, during the sweep phase all local objects marked by tasks are unmarked and when the local sweep has completed a *sweep acknowledgment* is sent to the *home* site.
- To allow the interleaving of mark task execution and mutator activity at S during distributed collection new objects have their DMB set on creation. This guarantees safety, since new objects are guaranteed to survive at least the distributed collection cycle in which they were created. To avoid the need for global synchronisation on distributed collection start-up all sites must assume that collection is continually in progress.
- Messages containing tasks, updates and termination notification are sent via the inter-site communications channels and as such are subject to site-to-site ordered delivery. Tasks are executed and *update* messages processed in strict order of delivery. As a consequence, the home site H can ignore the $sent_S(H)$ value in an update from a site S, since the tasks to which this count relates have already completed and been balanced.

For the DMJ home site H the club rules are all of the above and:

- A distributed collection is started by sending an RMT to each site including this site.
- The home site H maintains a task count array with an element for each site T holding the value *count*(T). On receipt of an *update* message from a site S, RC_S is deducted from *count*(S) and for

each site T, $sent_S(T)$ is added to $count(T)$. When $\forall_T.count(T) = 0$ then the termination condition holds and distributed marking has completed.

- The sweep phase is synchronised across all sites by sending a sweep message to each site, on DMJ termination detection, and waiting for all site to reply. Having received a *sweep acknowledgment* message from all sites, the home site is free to start the next distributed collection.

4.1.2 Club Rules for a Homogeneous Mark-Sweep Collector

Our homogeneous mark-sweep collector implements the generic club rules as described above and implements the sweep phase as follows.

On receipt of a *sweep* message, a site pauses mutator activity and scans the whole of its local cache. Unmarked objects are reclaimed at this point. During the sweep phase the address translation table entries of unmarked objects are removed. On completion of its local sweep a site sends a *sweep acknowledgement* message to H.

We have abstracted over the behaviour of a site during the local sweep phase. Whether a site compacts the free space or constructs a free list is orthogonal to the distributed collector. However during its sweep all objects in the local cache are unmarked in preparation for the next distributed collection cycle. This may be achieved by flipping the meaning of the DMB for a site and thus unmarking all objects simultaneously.

4.1.3 Separating Local and Distributed Collection

Our homogeneous distributed mark-sweep scheme restricts the reclamation of objects at a site to the sweep phase following termination of distributed marking. This is clearly unacceptable as many local collections may be required in between distributed collections. Separating local and distributed collection allows both the timing of the collections to be independent and also the nature of each of the local collectors and the distributed collector to vary. To this end we must provide a site with a set of local roots that will allow for safe independent local collection. This local root set contains the distinguished local root plus all local objects referenced by a remote site. The latter part of this root set is called the *distributed root set*. A conservative, but safe, view of this root set is already implemented at each site in the $DA_{sym} \rightarrow CA$ tables. On DA export an entry is added to the $DA_{sym} \rightarrow CA$ table at a site and safe local collection is possible if each entry is treated as a local root.

The club rules maintain the *distributed root* set at a site by identifying those entries in the $DA_{sym} \rightarrow CA$ table that represent objects still referenced by a remote site. The local collections may take place autonomously, from distributed collections by using its local root set, thereby separating the implementation of safety from the reclamation of space at a site.

The reader should note that we adopt no particular stand on the suitability, desirability or efficiency of independent (and possibly heterogeneous) local collection as compared to homogeneous schemes. Our aim is only to show how within the confines of a single DTA to GC mapping a number of collectors may be implemented.

4.1.4 Club Rules for a Heterogeneous Mark-Sweep Collector

Our heterogeneous collection scheme implements the generic mark-sweep club rules as described above. Here we describe the implementation of the local sweep and additional rules that are necessary for *distributed root set* maintenance and heterogeneous local collection support. For all sites S:

- Each $DA_{sym} \rightarrow CA$ table entry at S has two flags associated with it. The first is the *distributed root* flag (DRF). Table entries for which this flag is set represent roots of reachability at S. On

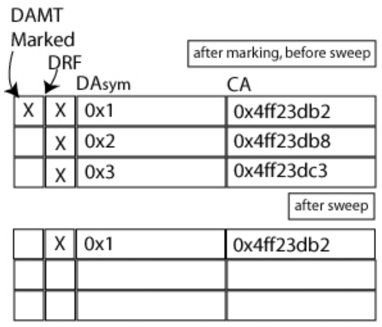


Figure 2: Marking Table Entries

DA export a table entry is created for this DA with this flag set. The local root set for local collection at S contains the distinguished root object and each object with a $DA_{sym} \rightarrow CA$ table entry with its *distributed root* flag set. The second flag is the *DAMT marked* flag. The purpose of this flag is explained in the next rule. This flag is set in all new table entries.

- On receipt of a DAMT for a DA the *DAMT marked* flag is set in the corresponding $DA_{sym} \rightarrow CA$ table entry at S. After setting the *DAMT marked* flag the DAMT executes as specified in the generic rules.
- On termination of a distributed marking phase the $DA_{sym} \rightarrow CA$ table entries that have their *DAMT marked* flag set represent the remotely referenced objects at S. On receipt of a sweep message S first pauses mutator activity and then scans its $DA_{sym} \rightarrow CA$ table to reconstruct the *distributed root set* by using the *DAMT marked* flags. This involves setting the *distributed root* flag in each entry that has its *DAMT marked* flag set, and clearing the *distributed root* flag for all entries that do not have their *DAMT marked* flag set. During the scan all *DAMT marked* flags are cleared.
- After the table scan the meaning of the DMB is flipped effectively unmarking all local objects and ensuring that they are unmarked before the start of the next distributed mark phase. Here we see the benefit of such an approach to unmarking since there is no need to scan the whole local cache. After its sweep of the distributed root set a site sends a *sweep acknowledgment* to the distributed marking *home* site.

Following the local sweep, each entry in a site's $DA_{sym} \rightarrow CA$ table with its distributed root flag set constitutes a root of reachability for local collection. To allow for the interleaving of local mutator activity and the execution of marking tasks all new table entries must have their *distributed root* flags set. Figure 2 shows the state of a $DA_{sym} \rightarrow CA$ immediately following a distributed marking phase and the same table after the sweep. Only those entries with their *DAMT marked* flag set are maintained.

4.1.5 Local Collection in the Heterogeneous System

Here we describe two local collection mechanisms for sites in the heterogeneous system. In both cases the club rules have a minimum impact on the behaviour of the local collectors. The local collectors are charged with updating a local site's address translation table entries if objects are moved and with removing entries for reclaimed objects. The local collectors treat *remote resident* objects as local objects thus ensuring that there is no interference between local collection and the object duplication policy. These objects are identified by having entries in the $DA \rightarrow CA$ table.

4.1.6 A Local Semi-Space Copying Collector

As a first example of independent local collection we describe a non-incremental, semi-space, copying collector implementation based on Cheney's list compaction algorithm from Cheney (1970).

The local cache is split into two equally sized areas (semi-spaces). While mutator activity is ongoing all objects reside in one area and any new objects are created in this area; during this phase the other area is unused. The idea of semi-space collection is to trace the object graph copying reachable objects from one space to the other. When an object is copied, the new address of the copied object (a forwarding pointer) is written into the original object. As tracing proceeds each copied object is scanned for references. For references to copied objects the forwarding pointer is used to update the reference, otherwise the object is copied and the reference updated. A consequence of copying objects to a vacant space is that the free space is compacted.

The local collector must update the address translation table entries for all copied objects. After all reachable objects have been copied the address translation tables are scanned. Each entry, that references an object with a forwarding pointer (i.e. a copied object), is updated and those entries for objects with no forwarding address, are removed. Finally, mutator activity is resumed, now using the space to which objects were copied.

When collection begins each object in the local root set is copied. Copied objects are then sequentially scanned for references, resulting in a breadth first traversal of the local object graph.

4.1.7 A Local Mark Sweep Collector

As a second example of independent local collection we describe a stop-the-world, mark-compact local collector, based on the Lockwood Morris (1978) algorithm.

A local garbage collection can be performed at any time. Local collection proceeds as follows. Mutator activity at the local site is first stopped and then the object graph is traced from the local root set. Each object has an associated local mark bit (LMB). The LMB is set during the marking phase for each object that is traced.

When marking is complete the heap is scanned and compacted, clearing LMBs and updating local references and address translation table entries (in both the $DA_{sym} \rightarrow CA$ and $DA \rightarrow CA$ tables). During the scan/compact phase the address translation table entries of unmarked objects are removed.

4.1.8 Discussion

Our new TB to mark-sweep mapping minimises the number of tasks by only spawning tasks for inter-site references. This contrasts with the DM-S mapping in Blackburn, Hudson *et al* (2001) where a task is mapped to the marking of an individual object and a task is spawned for each reference. Our mapping also benefits from avoiding the need to balance locally spawned tasks since there are none.

Separation of local and distributed collection work enables flexibility in local collector behaviour. Sites are free to implement any local collection scheme, and are constrained only by having to implement the club rules. An important consequence is that a site can do as many local collections as is necessary, independently of global collection.

5. DISTRIBUTED REFERENCE COUNTING

A traditional reference counting garbage collector, for instance Collins (1960), associates a reference count variable (initialised to zero) with each created object. On reference copy (stack push, reference export and pointer field update) the reference count is incremented and on reference deletion (stack pop and pointer field update) the count is decremented. An object can be reclaimed as soon as its reference count reaches zero. Such a collector is incremental by its very nature since it allows for the immediate collection of garbage objects, however it is not complete. The reference counts for objects in isolated (i.e., garbage) cycles will stabilise with non-zero values.

In deriving a distributed reference counting mechanism that allows for heterogeneous local collection behaviour we find it necessary to distinguish between local and remote (inter-site) references. That is, a site logically maintains two reference counts for each object. The first count is for local references and is maintained exactly as described above. The second count is a count of remote references to the object which is, by definition, distributed state. An object is reclaimed when both the local and remote reference counts are zero.

The remote reference count for an object is captured through a DTA mapping. Sites must be able to detect a remote reference count of zero. Therefore our DTA mapping is as follows:

- A job corresponds to a non-zero remote reference count for an object, called a distributed reference count job (DRCJ).
- A task of a DRCJ for an object x, written $DRCJ_x$, corresponds to an inter-site reference to x.

An object's creator site is designated as the DRCJ *home* site for that object. DRCJx is created when the first remote reference to x is exported from the home site. When DRCJx terminates, the remote reference count for the object x is zero. The club rules are thus an implementation of the DTA for each object at each site that has exported a reference plus the actions necessary to reclaim objects.

Watson and Watson (1987) and Bevan (1987) both describe a distributed reference counting algorithm called *binary weighted reference counting*. In this scheme a weight value is associated with each reference and when a reference is copied its weight is split between the original reference and the copy. The sum of all the weights for a particular object is analogue for the reference count for the object. An arbiter, that records the total weight, is associated with each object. When a reference is deleted the weight of that reference is subtracted from the total weight held by the arbiter. When the total weight is zero, the object may be reclaimed. The Credit Recovery DTA from Mattern (1989) is derived from the binary weighted reference counting collection scheme.

Whilst our approach ultimately yields a similar algorithm to weighted reference counting the difference is that our algorithm distinguishes between distributed and local collection work. Distributed termination detection is applied only to the distributed collection work thus allowing for heterogeneous local collection behaviour.

5.1.1 Club Rules for Distributed Reference Counting

The DTA mapping for reference counting provides a collector framework that uses a TB implementation to identify objects with a zero remote reference count. We make the following assertions:

- A site S sends a TB update for $DRCJ_j$, only when S holds no locally spawned tasks of j. At such a time, we say that s is *idle* for $DRCJ_j$.
- The method by which the idle state is detected is specific to a particular collector. For the description of the generic club rules it is sufficient to assume that a site can detect idle jobs.

The following describes the club rules that are generic to both distributed reference counting collectors. For a $DRCJ_j$ at a site S that is not the *home* site for j:

- S maintains data structures for recording TB *receive* counts as follows: the value *$received_S$*(j) records the number of tasks of j, received at S.

- S maintains the value $sent_S$(j,T) which records the number of task of j sent from *S* to *T*. When a task is sent to site T, $sent_S$(d,T) is incremented.
- When the site S detects that a j is idle, a TB update is sent to the home site of j. On sending an update for j, $sent_S$(j,T) is set to zero for all sites T and $received_S$(j) is set to zero.

  For the *home* site H of $DRCJ_j$ the club rules are all of the above and:
- The home site H maintains a task count array the job j. The count array has an element for each site T holding the value $count_H$(T). On receipt of an *update* message from a site S, RC_S is deducted from $count_H$(S) and for each site T, $sent_S$(T) is added to $count_H$(T). When $\forall_T.count_H(T) = 0$ then the termination condition is satisfied.

5.1.2 Club Rules for a Homogeneous Reference Counting Collector

In the homogeneous scenario, two separate reference counts are maintained for local objects, one for local references and the other (maintained by the TB implementation) for references from remote sites. An object x must have a zero local reference count *and* a zero remote reference count before it is collected.

A site detects the idle state for a DRCJj, using a local task counting mechanism. That is, a site S records the number of tasks it holds for each DRCJ job. When the count is zero, for a DRCJ j, S holds no tasks of j and an update message is sent for j. In other words, whenever a site S creates or deletes an inter-site reference to an object (which can be detected through the use of a write barrier for instance), it modifies the equivalent of a local reference count. When this count is zero, S holds no references to the object. Note that this is not the only means by which a site can detect the idle state for a DRCJ but we have chosen to describe this method since it most closely resembles a traditional reference counting scheme.

Our homogeneous reference counting collector implements the generic club rules as described above, and the following additional rules:

- S maintains a *local task count* value for each $DRCJ_j$ that it currently holds a task, written *LTC*(j). When a new task of j is created locally, *LTC*(j) is incremented. When a task of j is deleted (overwritten), *LTC*(j) is decremented.
- If *LTC*(j)=0 then DRCJj is idle at S.
- H maintains a *remote reference count* value for each object x, written *RRC*(x), for which it holds a corresponding DRCJ. On creation of the DRCJ for x, *RRC*(x) is initialised to one.
- A site maintains *local reference count* value for each local object x, written *LRC*(x).
- When a local reference to x is created, *LRC*(x) is incremented, and when a local reference to x is deleted *LRC*(x) is decremented. If *LRC*(x) becomes equal to zero, x is reclaimed at this point if and only if *RRC*(x) is also zero.
- On termination of the DRCJ for an object x, *RRC*(x) is set to 0. The object x is reclaimed at this point if and only if *LRC*(x) also equals zero.
- When an object x is reclaimed the local site carries out the appropriate actions for the deletion of each reference in x.

The homogenous collector is not complete since local and inter-site cycles of garbage are not reclaimed.

5.1.3 Separating Local and Distributed Collection

Since the distributed reference counting collection is only concerned with the counts of remote references, the separation of local and distributed collection is almost trivial. The only requirement of the local collector is to identify idle jobs.

To enable safe independent local collection, we adopt a similar approach to that of the distributed mark-sweep scheme described earlier. When a reference (DA) to a local object is first exported from a site, the object is added to a *distributed root set* for that site. Here the distributed reference counting collector will remove an object for the *distributed root set* when it determines that no other site holds a reference to the object, i.e., on termination of the DRCJ associated with that object. As before, local collection can proceed at any time based on reachability from the *local root set* (which contains the distinguished local root and the *distributed root set*).

5.1.4 Club Rules for a Homogeneous Reference Counting Collector

Here we describe two local collection mechanisms for sites in the heterogeneous distributed reference counting system. As in the distributed mark-sweep collector, the club rules have a minimum impact on the behaviour of the local collectors. The local collectors are charged with updating a local site's address translation table entries if objects are moved and with removing entries for reclaimed objects. The local collectors treat *remote resident* objects as local objects thus ensuring that there is no interference between local collection and the object duplication policy. These are identified by having entries in the DA → CA table.

Our heterogeneous collection scheme implements the generic distributed reference counting club rules as described above and the following rules:

- Each DA_{sym} → CA table entry has an associated *distributed root* flag (as in the distributed mark sweep collector), or DRF. The flag is set to identify an object that is currently in the *distributed root set*.
- On termination of the DRCJ corresponding to an object x, the DRF flag is cleared in the DA_{sym} → CA table entry for x.

5.1.5 A Local Mark Sweep Collector

As a first example of independent local collection we again describe a stop-the-world, mark-compact local collector. Local collection proceeds as follows. Mutator activity at the local site is first stopped and then the object graph is traced from the local root set. Each object has an associated local mark bit (LMB). The LMB is set during the marking phase for each object that is traced.

The TB data structure that records the sent and received counts for each DRCJ is extended to also include a LMB. During marking the LMB for a DRCJ is set on discovery of a remote reference to the object corresponding to that job. This can be thought of as marking the DRCJ jobs for which the site currently holds tasks.

When marking is complete the heap is scanned and compacted, clearing LMBs and updating local references and address translation table entries (in both the DA_{sym} → CA and DA → CA tables). The TB data structures are then scanned. Any unmarked DRCJ is idle at this site.

5.1.6 A Local Semi-Space Copying Collector

As a second example of independent local collection we illustrate a semi-space collector as described in section 4.1.6. To determine idle DRCJs, the local collector, at the end of a copy phase, records the set of remote references discovered during the copy phase. Before mutator activity is restarted, this set is compared with the set from the previous copy phase. Any remote reference missing from the latest set corresponds to a DRCJ that is idle at this site.

6. DISTRIBUTED GENERATIONAL COLLECTION

A generational collection scheme partitions the address space into two or more parts (generations) and places objects in generations based on their age. All objects are created in the youngest (zeroth) generation. On some threshold of collections (age) an object, if it is not garbage, is promoted from its current generation to the next older generation. The effect of age based promotion is that the zeroth generation acts as a nursery and older objects are found in older generations. Generations are collected in age order, starting at the youngest generation, allowing the collector to reclaim unused space from one generation without having to trace the entire space.

Here we describe a generic stop-the-world generational collector, based on Leibermann and Hewitt (1983). To collect a generation, a data structure known as a remembered set (remset) is used to record all references into the generation. To maintain these remsets two mechanisms are used; a write barrier to catch references from older to younger generations and age ordering of generation collection to discover references from younger to older generations.

In our distributed generational collector the address space is partitioned by generations that span sites. A segment is defined as a portion of a generation held on a particular site. To simplify the collector, we specify that the number of generations is fixed and each site holds a segment of every generation. A segment is a fixed size and represents a contiguous area of storage at a site. Each site maintains a portion of the remset for each generation. Distributed collection is concerned with the identification of garbage within a single generation across all of its sites.

A goal of our scheme is that local collection should be independent of distributed collection. We wish to allow local collection to proceed between distributed collections and interleaved with mutator activity. In our collector, promotion of an object takes place from one generation to another within a single site thereby avoiding forced migration of objects.

The club rules for distributed generational collection provide the mechanisms to identify garbage objects across all the sites of a generation and maintain the remsets. Reachable objects can be identified through copying or marking. Since we have already described a distributed marking scheme we will use a modified version of this scheme in the collection of a generation. Distributed collection always starts with generation G_0 and then collects as many older generations as dictated by policy. Collection starts by first pausing mutator activity on all sites. The collection of generation G_i begins with a distributed marking phase to identify all objects in G_i reachable from its remset. Following distributed marking objects are promoted.

In terms of mapping the TB DTA there are two jobs for the collection of each generation. The first job corresponds to the distributed marking phase, called the distributed marking job (DMJ) and the second is the distributed promotion job (DPJ). The site that initiates the jobs is called the DMJ/DPJ *home* site.

A DMJ consists of two types of tasks, the first is called a Root Marking Task (RMT) and the second is called a Distributed Address Marking Task (DAMT). When the DMJ home site starts a job, it sends a RMT to every site (including itself). Both task types trace the local graph of objects from a given start point, completely within the local segment of the generation being collected. Each object at a site has a distributed mark bit (DMB) associated with it. As an object is traced by a task its DMB is set and it is then scanned for references. Both types of task complete when they have fully traced the local graph within the segment, from their start point.

To maintain remset entries for remote references, sites need to record the generations of the remote objects they reference. On export of a DA to a site T, the generation of the referenced object is also sent to T. Each site maintains a DA generation lookup table that enables the site to determine the generation of each DA it holds. When an object is promoted, sites that reference this object are informed, allowing these sites to update their lookup tables. The set of referencing sites consists of each site that sent a DAMT for the promoted object during the distributed collection.

Each site implements the write barrier as described above. As each intra-site older to younger generation reference is created an entry of the form,

<*source object, target object*>

is added to the remset of the target (younger) generation.

On the creation of an inter-site older to younger generation reference an entry of the form,

<*source generation, DA*>

for the referenced object is added to the remset of the younger generation at the local site.

Thus, a site's remset for generation G_i may contain entries for both local and remote objects. For each remote object entry a DAMT is sent on execution of the RMT for G_i.

An RMT at site S for generation G_i begins by removing remset entries at S for all references from G_i to all other generations. As G_i is collected, remset entries are added for all local references found thereby reconstructing the accurate remsets.

A DPJ is started after DMJ termination for a generation and consists of two types of task; a *promotion* task and a *generation update* task. The DPJ starts with the DPJ *home* site sending a *promotion* task to each site T. The *promotion* task executes at a site S by sending a *generation update* task to each site that references a promoted object at S, and then completes. A *generation update* task sent from S to U updates the DA generation lookup table at U for DAs of promoted objects at S, and then completes.

On DPJ termination, collection of the current generation is complete. The *home* site may then begin collection of the next older generation, or restart mutator activity.

6.1 Club Rules for Distributed Generational Collection

The following describes the club rules for distributed generational collection. For a site S ≠ DMJ home site:

- S maintains data structures for recording TB sent counts for both RMTs and DAMTs. Task counts are maintained as follows. Each site S maintains a sent count array with an element for each site T recording sentS(T). When a task is sent to site T, sentS(T) is incremented. An update is sent on the completion of each task, thus RCS is always 1 and therefore no received count is required. When the update is sent, for each site T, sentS(T) is set to zero.
- When a DAMT is sent from S to a remote site T the *sent* count for T at S, $sent_S$(T), is incremented.
- New objects are created in the youngest generation.
- S implements a write barrier that acts on the creation of all (inter- and intra-site) older to younger generation references. When such a reference is written into an object, an entry is added to the remset for the local segment of the referenced object's generation.
- S maintains an *in-flight* table where all DAs sent in messages to remote sites are recorded. On receipt of a message containing DAs, from a site T, S must send an acknowledgment message back to T. The acknowledgment contains the DAs that were sent to S from T. When S receives an acknowledgement it removes the *in-flight* table entries for each DA in the message.
- Implementation of an RMT for generation G_i at S is as follows. Any remset entries at S for references from G_i are removed. The RMT then traces the object graph from each object in G_i's remset, marking reachable objects using their DMB. Only objects in G_i are traced. For each reference from G_i to $G_{j≠i}$ (local or DA) found during tracing, an entry is added to G_j's remset at S. For each DA found for a remote object in G_i (as determined by the DA generation lookup

table) a DAMT is sent to the remote site. On RMT completion, a TB update is generated.

- Implementation of a DAMT for generation Gi is as follows. A DAMT message contains the DA of an object O that is remotely referenced. The local object graph is traced from O, marking reachable objects using their DMB. Only objects in Gi are traced. For each reference from Gi to Gj≠i (local or DA) found during tracing, an entry is added to Gj's remset at S. For each DA found for a remote object in Gi (as determined by the DA generation lookup table) a DAMT is sent to the remote site. On completion of a DAMT a TB update is generated and sent to the home site.
- Messages containing tasks and updates are subject to ordered delivery. Tasks are executed and *update* messages processed in strict order of delivery. As a consequence, the home site H can ignore the $sent_S$(H) value in an update from a site S, since the tasks to which this count relates have already completed and been balanced.
- S must promote an object *O* in generation G_i when it first encounters *O* during the collection of G_i. Each referencing site must be informed of the promotion so that its DA generation lookup table may be updated. S must record each site from which a DAMT, for a promoted object *O*, is received. This defines the set of remote sites that reference *O*.
- On receipt of a *promotion* task for generation G_i, S sends a *generation update* task to each site that references a promoted object. A TB *update* message is then sent to the DPJ *home* site. The TB *sent* count data structures and actions used for the DPJ are identical to those used for the DMJ.
- To complete the garbage collection of G_i at S the local collector must move all objects that it has previously decided to promote from G_i to G_{i+1} and unmark all DMB marked objects in G_i When an object *O* in G_i is promoted, remset entries for *O* in G_i are transferred to G_{i+1} and appropriate remset entries added and updated for any references in *O*.

For the DMJ home site H the club rules are all of the above and:

- Before collection begins, mutator activity on all sites must be paused.
- Collection of a generation is started by sending an RMT to each site including this site.
- The home site H maintains a task count array with an element for each site T holding the value *count*(T). On receipt of an *update* message from a site S, RC_S is deducted from *count*(S) and for each site T, $sent_S$(T) is added to *count*(T). When $\forall_T$.*count*(T) = 0 the *terminated* condition holds and the current job has completed.
- On DMJ termination detection the DPJ is started by sending a *promotion* task to each site. On DPJ termination, collection of the current generation is complete.

6.2 Club Rules for Homogeneous Distributed Generational Collection

The homogeneous distributed generational collector restricts the reclamation of objects at a site to the promotion phase following distributed marking. Each site implements the generic club rules as defined above and takes the following actions on receipt of the *promotion* task for generation G_i.

Any objects in G_i that are to be promoted are moved to G_{i+1} and all local references and address translation table entries updated accordingly. The local segment of G_i is then compacted using a Lockwood-Morris implementation operating only on intra-segment references. Each object on the local site in a generation G_x (x≠i) that holds a reference to an object in G_i is identified in the remset for G_i. All inter-generation (intra-site) pointers to objects in G_i and any address translation table entries are updated as the segment is compacted (and objects are moved).

The address translation tables for unmarked objects are removed. During the compaction of the local segment of G_i all DMB marked objects are unmarked.

6.3 Separating Local and Distributed Collection

To allow for independent local collection we adopt an approach similar to that of the distributed mark-sweep scheme described earlier. When a reference (DA) to a local object is first exported from a site, the object is added to a *distributed root set* for that site. The distributed collector removes an object for the *distributed root set* when it determines that no other site holds a reference to the object. After the collection of the generation in which an object in the *distributed root set* is held, if a site has received no DAMT for that object and there is no remset entry for that object then the object can be removed from the *distributed root set*.

A local collector can work over the whole space at a site using the local root and the *distributed root set* as its roots of reachability. This allows multiple local collections to execute between distributed collections.

6.4 Club Rules for Heterogeneous Distributed Generational Collection

Our heterogeneous collection scheme implements the generic distributed generational club rules as described above. Here we describe the actions taken by a site on receipt of the *promotion* task for generation G_i necessary for *distributed root set* maintenance and heterogeneous local collection support. The rules for maintenance of the distributed root set are almost identical to those for distributed mark-sweep and so only a summary is given here. For all sites S:

- Each $DA_{sym} \rightarrow CA$ table entry at S has a *distributed root* flag and a *DAMT marked* flag associated with it. The *distributed root* flag is set for all newly exported DAs.
- On receipt of a DAMT for a DA the *DAMT marked* flag is set in the corresponding $DA_{sym} \rightarrow CA$ table entry at S. After setting the *DAMT marked* flag the DAMT executes as specified in the generic rules above.
- On receipt of the *promotion* task for generation G_i the $DA_{sym} \rightarrow CA$ table is scanned and the portion of the distributed root set for G_i at S is reconstructed. The *distributed root* flag is set for each entry, for an object in G_i, that has its *DAMT marked* flag set and for each entry for an object that the remset for G_i at S holds an entry. The *distributed root* flag is cleared for all other $DA_{sym} \rightarrow CA$ table entries for objects in G_i. At this point the *DAMT marked* flag is cleared for all entries for objects in G_i. Having reconstructed the portion of the *distributed root set* for G_i all objects in the local segment must be (DMB) unmarked. To avoid the necessity to sweep the whole segment the meaning of the DMB mark bit for G_i at S is flipped as before. This requires that a site maintains the DMB marked value for the local segment of each generation.

Following the completion of a distributed collection cycle each entry in a site's $DA_{sym} \rightarrow CA$ table with its *distributed root* flag set constitutes a root of reachability for local collection.

6.5 Local Collection in the Heterogeneous System

Here we describe a local collection mechanism for sites of the heterogeneous distributed generational system. As in the heterogeneous mark-sweep system the address tables for moved objects are updated and *remote residents* are treated as local objects. The local collector is aware of the generations at a site and must update the local remsets when objects are moved.

6.6 A Local Semi-Space Copying Collector

Our local collection mechanism is a copying collector that divides each generation segment at a site into semi-spaces. On collection, all reachable objects in each segment are copied from their current location to the free semi-space of that segment, leaving a forwarding pointer to indicate their new location.

Local collection begins by pausing mutator activity and then copying the distinguished root object and each of the distributed root objects to the free-semi space of their respective generation segments. Each copied object is scanned for references to other objects which are in-turn copied.

When copying has completed the address tables for the local site are scanned and updated. Table entries for non-copied (garbage) objects are removed at this point.

6.7 Discussion

The stop-the-world mechanism used in the collection of a distributed generation is not well suited to a scalable distributed system. The distributed generational collection scheme as presented here represents more of a proof of concept of the DTA to GC mapping derivation technique than a suitable distributed collector. We have however attempted to keep our description as close as possible to that of Leibermann and Hewitt (1983).

Clearly a distributed generational collector where mutator activity can be interleaved with the collection of a generation is preferable to the approach we describe. We see the development of such a scheme as further work. We consider the problem of remset maintenance in the face of interleaved generation collection and mutator activity to be similar to that faced by an implementation of the DMOS collector (Hudson, Morrison *et al*, 1997) in maintaining car and train remsets.

Our use of synchronous communication for the transmission of inter-site references (DAs) also reduces the scalability of the collector. Addressing this problem is also seen as further work.

7. RELATED WORK

Our approach to implementing distributed garbage collection algorithms by combining a DTA and a non-distributed collector is unusual but has some derivation in previous work. In particular, Mattern and Tel (1993) have shown that at least one distributed termination detection algorithm may be derived from any distributed garbage collector. This led to our earlier work, Blackburn, Hudson *et al* (2001), where a methodology for combining a DTA and a non-distributed garbage collection scheme yields a distributed collector that is both safe and maintains the properties of the original GC scheme. We have developed this work significantly by enabling independent and heterogeneous local collection.

Prior to this, Hughes (1985) described a distributed collector in terms of the combination of a known centralised collection mechanism, mark–sweep, and a specific distributed termination detection algorithm, Rana (1983). The collector uses timestamps issued from a central site; each timestamp signifying a new wave of collection. Termination of each wave is determined by Rana's DTA. Hughes' collector described one collector. Here we are concerned with a family of collectors using any DTA and collection scheme.

Interestingly Tel, Tan and van Leeuwen (1986) described how distributed graph marking algorithms may be derived from termination detection algorithms. The relationship to garbage collection is obvious.

We have stated that our collectors are neutral towards coherency policy although we observe that there may be efficiency benefits where the garbage collection and coherency mechanisms co-operate. The issues surrounding the combination of collection and coherency mechanisms are addressed by Ferreira and Shapiro (1994) and by Munro, Falkner *et al* (2001).

8. CONCLUSIONS

We have demonstrated the practical application of the distributed garbage collector derivation methodology presented in our previous work, Blackburn, Hudson *et al* (2001). Our contribution is the construction of an experimental platform, implementations of the Task Balancing DTA, an extension to the methodology that minimises constraints on local collectors, together with three new mappings and their implementations.

Our first contribution is the implementation of the ProcessBase cache and its associated addressing mechanism as the experimental platform for our system model. Secondly we extended the methodology to minimise the constraints on the local collector. This is achieved by mapping a DTA onto any (non-distributed) garbage collection scheme, whilst preserving the interesting properties (in particular safety and completeness) of the collector. Each such mapping defines a set of club rules that must be obeyed by each participant in the distributed collection scheme. The participating collectors are free to perform any local actions as long as they preserve these club rules.

We presented three new mappings that constitute club rules for six distinct distributed collection schemes. These mappings use implementations of the Task Balancing DTA on the ProcessBase distributed cache. Our final contribution is to show that once the club rules are defined it is possible to vary the collection schemes at participant sites thereby introducing a degree of heterogeneity in the local collectors. We postulate that future work may extend to deriving a set of club rules that operate over existing local collectors.

9. ACKNOWLEDGEMENTS

The work is supported in part by EPSRC GR/R84881/01 – Visiting Fellowships for Distributed Garbage Collection, EPSRC GR/R85907/01 – DIMNET: A Network of Excellence in Distributed Information management, ARC Large Grant 2000: A10033014 (Australia) – Deriving a Family of Distributed Garbage Collection Algorithms and EPSRC GR/M74931 – Collecting Distributed Garbage using the DMOS Family of Algorithms.

REFERENCES

BECKER, D.J., STERLING, T., SAVARESE, D., DORBAND, J.E., RANAWAK, U.A., PACKER, C.V. (1995): Beowulf: A parallel workstation for scientific computation. *International Conference on Parallel Processing*.

BEVAN, D.I. (1987): Distributed garbage collection using reference counting. *Lecture Notes in Computer Science*, 258. Springer-Verlag.

BLACKBURN, S.M., HUDSON, R.L., MORRISON, R., MOSS, J.E.B., MUNRO, D.S., ZIGMAN, J.N. (2001): Starting with termination: A methodology for building distributed garbage collection algorithms. *24th Australasian Computer Science Conference*: 20-28.

CHENEY, C.J. (1970): A non-recursive list-compacting algorithm. *Comm. ACM* 13(11): 677–678.

COLINS, G.E. (1960): A method for overlapping and erasure of lists. *Comm. ACM*, 2(12): 655–657

FERREIRA, P., SHAPIRO, M. (1994): Garbage collection and DSM consistency. *Proceedings of the First Symposium on Operating System Design and Implementation*, 14–17.

LOCKWOOD MORRIS, F. (1978): A time and space efficient garbage collection algorithm. *Comm. ACM*, 21(8): 662–665

HUDSON, R.L., MORRISON, R., MOSS, J.E.B., MUNRO, D.S. (1997): Training distributed garbage: The DMOS collector. *Technical Report, University of St Andrews, School of Computer Science*.

HUGHES, J.R.M. (1985): A distributed garbage collection algorithm. *Lecture Notes in Computer Science*, 201: 256–272. Springer-Verlag.

LIEBERMAN, H., HEWITT, C. (1983): A real-time garbage collector based on the lifetimes of objects. *Comm ACM*, 26(6): 419–429.

MATTERN, F. (1989): Global quiescence detection based on credit distribution and recovery. *Information Processing Letters*, 30(4): 195–200.

MORRISON, R., BALASUBRAMANIAM, D., GREENWOOD, M., KIRBY, G.N.C., MAYES, K., MUNRO, D.S., WARBOYS, B.C. (1999): ProcessBase Reference Manual v1.0.6. *Universities of St Andrews and Manchester Report*. http://www.dcs.st-and.ac.uk/rsch/publications/MBG+99a.shtml

MUNRO, D.S., FALKNER, K.E., LOWRY, M.C., VAUGHAN, F.A. (2001): Mosaic: A non-intrusive complete garbage collector for DSM systems. Proceedings of the First IEEE/ACM *International Symposium on Cluster Computing and the Grid, Third International Workshop on Software Distributed Shared Memory*: 539–546.

NORCROSS, S.J., FALKNER, K., MUNRO, D.S., BALASUBRAMANIAM, D., KIRBY, G.N.C., MORRISON R. (2001): An object cache for the distributed processBase interpreter. *Technical Report TR2001-02*, Department of Computer Science, University of Adelaide.

PLAINFOSSÉ, D., SHAPIRO, M. (1995): A survey of distributed garbage collection techniques. *Lecture Notes in Computer Science*, 986: 211–249. Springer-Verlag.

RANA, S.P. (1983): A distributed solution to the distributed termination problem. *Information Processing Letters*, 17: 43–46.

TEL, G., MATTERN, F. (1993): The derivation of distributed termination detection algorithms from garbage collection schemes. *ACM Transactions on Programming Languages and Systems*, 15(1): 1–35.

TEL, G., TAN, R.B., VAN LEEUWEN, J. (1986): The derivation of graph marking algorithms from distributed termination detection protocols. *Technical Report RUU-CS-86-11*, Department of Computer Science, University of Utrecht.

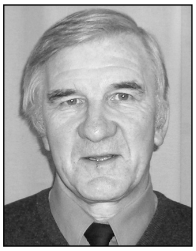
WATSON, P., WATSON, I. (1987): An efficient garbage collection scheme for parallel computer architecture. *Lecture Notes in Computer Science*, 258. Springer-Verlag.

BIOGRAPHICAL NOTES

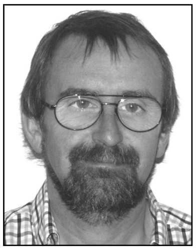
Dr Stuart Norcross has just completed his PhD entitled "Deriving Distributed Garbage Collectors for Distributed Termination Algorithms". His special interests are in distributed termination, distributed garbage collection and distributed object stores.

Stuart Norcross

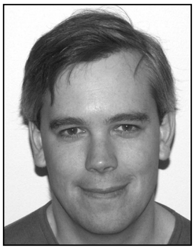
Professor Ron Morrison holds the Chair of Software Engineering at the University of St Andrews where he leads the Software Architecture Group. He is an adjunct professor at the University of Adelaide. His special research interests are programming language design, persistent object systems, reflective computing, and evaluation of complex systems, distributed garbage collection, and compliant systems architectures. He currently leads the UK Network of Excellence on Distributed Information Management.


Ron Morrison

Dr David S. Munro leads the Jacaranda Group at the University of Adelaide. His primary interests are in distributed persistent object systems, distributed garbage collection and compliant systems architectures.


David Munro

Dr Henry Detmold and Dr Katrina Falkner are members of the Jacaranda Group at Adelaide. Henry's principal interests are in wide area distributed systems, distributed systems with decentralised control and open persistent systems. Katrina's primary interests are in distributed memory management, evolution in compliant systems and naming in distributed and mobile object systems.


Henry Detmold

Katrina Falkner