Distributed Garbage Collection for Wide Area Replicated Memory

from Alfonso Sánchez, Luís Veiga, Paulo Ferreira

presented by Andrea Rezzonico
Overview

1. Introduction
2. Memory Model
3. Algorithm
4. Example
Introduction
Introduction (2)

\( y \) is not reachable from the process's local root
Introduction (3)

other processes may have a reference to $y$
Current Solutions

- Distributed reference counting
  - Problem with cycles
- Reference listing
  - Mark & sweep

What happen if we allow object replication?
Replication

Mark & sweep consider \( z \) garbage
but after a propagation $z$ is reachable
Motivation

- Shortcomings of Current Solutions
  - Replication is not considered
  - Constraints on scalability

- Solution
  - Deals with replication
  - Does not impose constraints on scalability
  - It is independent of the used coherence protocol
  - Does not deal with failures (i.e. communication failures)
Overview

1. Introduction
2. Memory Model
3. Algorithm
4. Example
WARM Model

- Several communicating processes
- Memory
- Mutator: modify the reference graph through assignment ($x := y_i$)
- Coherence engine: propagate the object replicas to keep them coherent ($\text{propagate}(y)_{i \rightarrow j}$)
What is Garbage?

process i

```
local root
```

```
\( \text{X} \)
```

```
\( \text{Z} \)
```

process j

```
local root
```

```
\( \text{X} \)
```

Concurrency Seminar: Distributed Garbage Collection for Wide Area Replicated Memory
What is Garbage? (2)

- Unreachable objects are considered garbage

Union Rule
A target object \( z \) is considered unreachable only if the union of all the replicas of the source objects do not refer to it.
Overview

1. Introduction
2. Memory Model
3. Algorithm
4. Example
Data Structures

- **stub** \((\text{OutRef, SourceObj, Scion, Chain})\)
- **scion** \((\text{InRef, Stub, Chain})\)
Data Structures (2)

- `inPropList / outPropList`  
  `(propObj, propProc, sentUMessage/recUMessage)`
Data Structures (3)

- inPropList / outPropList
  
  \((propObj, propProc, sentUmess/recUmess)\)
Garbage Collector Root

- Local root (stacks and static variables)
- Set of scions
- inPropList
- outPropList
Messages

• unreachable
  • object replica is reachable only from the inPropList
Messages (2)

- unreachable
  - object replica is reachable only from the \texttt{inPropList}
Messages (3)

- reclaim

  - all object replicas are reachable only from the inPropList
Messages (4)

- reclaim
  - all object replicas are reachable only from the inPropList
Messages (5)

- `newSetStubs`
  - when a new set of stubs is available
• **newSetStubs**
  - when a new set of stubs is available
Messages (7)

- `newSetStubs`
  - when a new set of stubs is available
DGC Algorithm

- Local Collector
  - Starts the *trace* from the process's local root, the set of scions, the inPropList and the outPropList
  - For every outgoing inter-process reference a stub is created and added to the *new set of stub*
  - For objects reachable only from the inPropList an *unreachable* message is sent and the sentUmess bit is set
  - For objects reachable only from the outPropList for which an *unreachable* message from all process was received a *reclaim* message is sent
DGC Algorithm (2)

• Distributed Collector
  • Does not require waiting for replicas to be coherent
  • From time to time a newSetStubs message is sent to the processes holding the scions corresponding to the stubs in the previous stub set
  • unreachable message received
    → set recUmess bit
  • reclaim message received
    → delete the corresponding entry in the inPropList
Overview

1. Introduction
2. Memory Model
3. Algorithm
4. Example
Example

\[ \text{propagate}(y) \rightarrow_i \]
\[ <x := y>_i \]
\[ \text{propagate}(x) \rightarrow_j \]
\[ <y := 0>_j \]
\[ \text{propagate}(y) \rightarrow_i \]
\[ <x := 0>_i \]
\[ x_j := 0 \]
\[ x_i := 0 \]
Example (2)

\[
\begin{align*}
  \text{propagate}(y)_{j \to i} & \quad <x := y>_{i} \\
  \text{propagate}(x)_{i \to j} & \quad <y := 0>_{j} \\
  \text{propagate}(y)_{j \to i} & \quad <x := 0>_{i} \\
  x_{j} & := 0 \\
  x_{i} & := 0
\end{align*}
\]
Example (3)

\[
\begin{align*}
\text{propagate}(y)_{j \rightarrow i} \\
<x := y>_{i} \\
\text{propagate}(x)_{i \rightarrow j} \\
<y := 0>_{j} \\
\text{propagate}(y)_{j \rightarrow i} \\
<x := 0>_{i} \\
x_{j} := 0 \\
x_{i} := 0
\end{align*}
\]
Example (4)

\[
\begin{align*}
\text{propagate}(y)_{j \rightarrow i} & \quad <x := y>_{i} \\
\text{propagate}(x)_{i \rightarrow j} & \quad <y := 0>_{j} \\
\text{propagate}(y)_{j \rightarrow i} & \quad <x := 0>_{i} \\
\end{align*}
\]
Example (5)

\( \text{propagate}(y)_{j \rightarrow i} \)

\( <x := y>_{i} \)

\( \text{propagate}(x)_{i \rightarrow j} \)

\( <y := 0>_{j} \)

\( \text{propagate}(y)_{j \rightarrow i} \)

\( <x := 0>_{i} \)

\( x_{i} := 0 \)

\( x_{j} := 0 \)
**Example (6)**

\[ \text{propagate}(y) \] from \( j \) to \( i \)
\[ \langle x := y \rangle \] in \( i \)
\[ \text{propagate}(x) \] from \( i \) to \( j \)
\[ \langle y := 0 \rangle \] in \( j \)
\[ \text{propagate}(y) \] from \( j \) to \( i \)
\[ \langle x := 0 \rangle \] in \( i \)
\[ x := 0 \] in \( j \)
\[ x := 0 \] in \( i \)
Example (7)

\[
\text{propagate}(y)_{j \rightarrow i} \\
<x := y>_{i} \\
\text{propagate}(x)_{i \rightarrow j} \\
<y := 0>_{j} \\
\text{propagate}(y)_{j \rightarrow i} \\
<x := 0>_{i} \\
x_{j} := 0 \\
x_{i} := 0
\]
Example (8)

\[
\begin{align*}
\text{propagate}(y) & \\
\langle x := y \rangle & \\
\text{propagate}(x) & \\
\langle y := 0 \rangle & \\
\text{propagate}(y) & \\
\langle x := 0 \rangle & \\
\end{align*}
\]
Example (9)

\text{propagate}(y)_{j->i}
\langle x := y \rangle_i
\text{propagate}(x)_{i->j}
\langle y := 0 \rangle_j
\text{propagate}(y)_{j->i}
\langle x := 0 \rangle_i
x_i := 0
x_j := 0
Remarks

- $x_i$ is reachable only from the inPropList
- $x_j$ is reachable only from the outPropList
- $z_k$ is reachable only from the scion S1
- no object can be collected yet
Collecting Garbage

• LGC in process i

• $x_i$ is reachable only from the inPropList

• an unreachable message is sent from i to j

• the sentUmess bit is set
Collecting Garbage (2)

- LGC in process j
- $x_j$ is reachable only from the outPropList
- delete the entry from the outPropList in j
- send a `reclaim` message
- delete the entry from the inPropList in i
Collecting Garbage (3)

- LGC in process j
- $x_j$ is collected
- S7 is deleted
- A newSetStubs message is sent to process i
- S6 is deleted
Collecting Garbage (4)

- LGC in process i
- x is reclaimed in process i
- S5 and S4 are deleted
- a newSetStubs message is sent to process j
- the distributed collector deletes S3 in process j
Collecting Garbage (5)

- LGC in process j
- S2 is deleted
- a `newSetStubs` message is sent from j to k
- S1 is deleted
Collecting Garbage (6)

- LGC in process k
- z can now be collected
Conclusion

● Advantages
  ● Deals with replication
  ● Scalability
  ● Orthogonal to the coherence engine

● Disadvantages
  ● Not fault-tolerant
Questions?
Appendix 1: Safety Rules

Safety rule 1 (Clean Before Send Propagate)
Before sending a propagate message for an object $y$ from a process $j$, $y$ must be cleaned (i.e. it must be scanned for references) and the corresponding scions created in $j$.

Safety rule 2 (Clean Before Deliver Propagate)
Before delivering a propagate message for an object $y$ in a process $i$, $y$ must be cleaned (i.e. it must be scanned for outgoing inter-process references) and the corresponding stubs created in $i$, if they do not exist yet.