AN ALGORITHM OF DISTRIBUTED GARBAGE COLLECTION ON A MULTICOMPUTER AND ITS PERFORMANCE EVALUATION

並列計算機上の分散ガーベージ・コレクションのアルゴリズムとその性能評価

by
Tomio Kamada
鎌田 十三郎

A Senior Thesis
卒業論文

Submitted to

the Department of Information Science
the Faculty of Science
the University of Tokyo
on February 23, 1993
in Partial Fulfillment of the Requirements
for the Degree for Bachelor of Science

Thesis Supervisor: Akinori Yonezawa 米澤 明憲
Title: Professor of Information Science

ABSTRACT

Automatic memory management is an important issue of massively parallel computing. In a distributed environment where processors are connected by asynchronous networks, the presence of pending messages makes it difficult to traverse references for a GC. In order to confirm the arrival of all pending messages, some existing GC schemes use ACK (a message that informs of the arrival of the sending message) and others use network clearance operations that send sweeping messages through all the network paths. But few studies compare performance with actual implementations. This study presents a distributed garbage collection algorithm that use weight for termination detection to reduce message overheads, and does not suspend ongoing computations, and allows each node to initiate local GC independently. The two methods (one is the variant of method using ACK and the other uses network clear operations) are compared through actual implementations on a multicomputer with 64-512 nodes, AP1000.

論 文 要 旨

高並列計算環境においては自動メモリ管理が重要である。非同期ネットワークでプロセッサが結合された分散環境では、未到着メッセージの存在のため GC 開始時のオブジェクトの参照関係を決定するのは難しい。従来の GC の手法では、全ての GC 開始以前のメッセージの到着を確認するため ACK(送ったメッセージの到着を知らせるメッセージ) を用いるもの、ネットワーク・クリア・オペレーションという全てのネットワークパスにスイープ・メッセージを流すものなどがある。しかし実際に実装し性能を評価した研究は少ない。本研究では、メッセージのオーバーヘッドを減らすため終了判定に重みを用い、実行中計算を止めることもなく、各ノードでの局所的 GC を独立して行うこともできる分散 GC のアルゴリズムを提案する。また 64 〜 512 ノードのプロセッサを持つ並列計算機である AP1000 上で実装し評価する。この際、二つの手法 (ACK を使うものの変種とネットワーク・クリアを行なうもの) を実装を通して比較する。

Acknowledgements

I would like to thank Professor Akinori Yonezawa for various kinds of advices and directions. I also thank Satoshi Matsuoka who recommend me to study distributed garbage collection and advising me, Masahiro Yasugi for theoretical and technical advices, Kenjiro Taura for much encouragement and many technical advices.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

On a multicomputer with distributed memory where processors are connected by asynchronous network, it is difficult to implement distributed garbage collection in concurrent object-oriented programing environments.

Some of difficulties are as follows:

- It is difficult to collect all garbages including cyclic ones ranging over nodes with light network traffic.
- Reference relations change over time by message passing, and in an asynchronous network, pending messages make it difficult to traverse references for GC.

Two Major Garbage collection schemes are **(1)** reference count type and its variant, **(2)** mark and sweep type that marks over nodes.

In this study, we select **(1)** to implement our distributed garbage collection (referring to as **DGC**) scheme to collect garbages including distributed cyclic ones by marking over nodes (referring as **global mark**).

Our goal is to implement DGC as follows:

1. It works concurrently with mutators and does not block ongoing computations.

2. It has local garbage collection system that can collect only local garbages and can run at any time independently of other nodes.

3. It has mark-and-sweep type distributed garbage collection system that can collect garbages including distributed cyclic ones.

To do with light network traffic, our scheme does not send acknowledgment (ACK) messages to confirm each message's arrival. or to confirm terminations of marking from objects. To confirm the arrival of all messages, we use a message count scheme (a variant of ACK messages) and bulldozing scheme (a sort of network clear operations), and compare them through actual implementation. To confirm termination of marking over nodes, we present our termination detecting algorithm using weight given to each marking activity as local collector and remote mark message as a unit.

The remainder of this thesis is structured as follows. Chapter 2 describes the related work on distributed garbage collection. Chapter 3 presents an approach to our DGC scheme. Chapter 4 forms the algorithm of our DGC, message count version and bulldozing version as well as their proof. Chapter 5 presents an implementation on AP1000 and in the performance of our DGC. We conclude in Chapter 6.

Chapter 2

Related Work

This chapter present previous work in distributed garbage collection. Section 2.1 describes problems to implement distributed garbage collections. Section 2.2, 2.3 presents two major types of DGC, reference count and mark and sweep type.

2.1 Problems

It is difficult to implement DGC that collect all garbages ranging over nodes with light network traffics. One major kind of DGC is using reference counter that has difficulties in collecting distributed cyclic garbages. Another major kind of distributed garbage collection does marking over nodes and has to send many messages for marking, confirming of arrival, termination detection of marking. Some algorithms of both are described below.

Another difficulty is concurrency with mutators (ordinary user programs). We have to care of mutators' activities on unmarked space that may change global snapshot for GC. In environments where processors are connected with an asynchronous network, we also care of pending messages that may change reference relations of objects.

Two major kinds of distributed garbage collection scheme, reference count schemes and distributed marking schemes, are described below.

2.2 Reference Count scheme and its Variants

Basic idea of reference count scheme is as follows:

- Remotely referenced objects and referencing objects have reference count.
- Local GC is done by adding remotely referenced objects to root set of marking.
- If referencing objects are reclaimed, reference counts of referenced object are decremented.

An algorithm of this kind is presented in [2]. This algorithm is simple and where each cell does GC on its local space independently. However it can not collect cyclic garbages ranging over nodes.

In [5], an algorithm where each cell does GC on its local space independently is presented which is fault-tolerant, largely independent of how a processor garbage collects its own space, allows for multiple concurrent active GCs, dose not need centralized control nor global stop-the-world synchronization, does not require to migrate objects from processor to processor and eventually reclaim all inaccessible objects including distributed cycles. Group GC is sometime done to remove dead cycles over nodes. However dead cycles are removed only when cycles are included in nodes doing group GC.

[8] presents the algorithm using timestamp that can collects distributed cyclic garbages. We do not understand it enough.

2.3 Distributed Mark-and-Sweep Schemes

Basic idea of distributed mark-and-sweep schemes is simple, to do mark-and-sweep GC over nodes.

In [1] , a mark and sweep type algorithm is presented where each processor has their collector process that accesses its own memory only. This collection scheme is for POOL system. Marking for remote reference is done by message passing between local collectors on each cell. This algorithm can collect all garbages including distributed cyclic ones. However all local collectors in network have to synchronize to wait for arrival of all old pending messages or for initialization of global mark on all nodes. Its termination detection is done by using three marking colors, black, gray and white. Gray objects are live objects markings from them are not completed. If no gray object exists on all nodes, marking phase is terminate.

In [4, 6], algorithms for Emerald System are presented. The algorithm in [4] has two garbage collectors: a node local collector that can run at any time independently of other nodes and a distributed garbage collector that collects distributed garbage by cooperation with other nodes to mark globally. Their termination detection is done by using three marking colors, same as in [1]. Mutators can run only in marked space. In [6], robust garbage collector is presented that is tolerant to partial fault.

[10, 11, 12] present algorithm of garbage collecting actors in distributed environments connected with an asynchronous network. This algorithm uses bulldozing method to get the consistent global snapshot for GC. Bulldozing method is a kind of network clearance operation where communications between nodes are FIFO order. By sending bulldozing messages through network, arrivals of sent messages before bulldozing are confirmed. However no actual implementation is not done.

[7] presents an algorithm of DGC in an unreliable asynchronous FIFO network. By using vector timestamp, each node knows global time and does not have to synchronize to get global states. To confirm arrivals of old pending messages, nodes only have to send acknowledgment messages to last messages for each node-to-node communications.

Chapter 3

Our Approach

This chapter presents our approach to distributed garbage collection. Section 2.1 presents our goal of DGC. Section 2.2 describes how to do local GC. Section 2.3 presents how to do global GC and our approaches to reduce global GC overhead.

3.1 Goals of Our Distributed Garbage Collection

We assume our DGC to work on multicomputers where processors are connected with an asynchronous network, where each channel has reliable and FIFO communication. Actually implementation is done on AP1000, a multicomputer with 64 - 512 nodes.

Our DGC is assumed to work in concurrent object-oriented programing environment, and to work concurrently with mutators and not to block mutators.

The definition of garbages in our DGC scheme is as described below.

- Active objects (that are currently processing messages or have messages that are not processed) are alive.
- Root objects (that are always useful objects such as I/O devices and global variables in programs) are alive.
- Objects are alive if other live objects (or their messages) know their object address.

- Garbage objects are objects that are not alive according to the above definitions.

Goals of our DGC are as follows:

1. It works concurrently with mutators and does not block ongoing computations.
2. It has local garbage collection system that can collect only local garbages and can run at any time independently of other nodes.
3. It has mark-and-sweep type distributed garbage collection system that can collect garbages including distributed cyclic ones with marking ranging over nodes (referring to as **global mark**).
4. It does not send acknowledgment messages for each sending message.
5. It does not send acknowledgment messages for each marking to detect terminations of global marking.

We select mark-and-sweep type garbage collection to collect garbages comprehensively. To reduce problem of mark-and-sweep type collector, that is overhead especially in network traffic, **4** and **5** are done.

3.2 Local Garbage Collection

We have to mark from objects referenced from remote live objects in local GC. To do local GC at any time independently of other nodes, conservative and simple method is selected in our collector. The set of nodes that are potentially referenced from remote objects is created as **exported set** on each node, and at starting local GC local collectors add members of their exported set to root set for local marking. To implement this exported set, objects are added to exported set when their object pointers are exported to remote objects. Deletions from exported set are done based on reference relationship at global garbage collection. When global garbage collection starts, each node clears their exported set and only objects that receive remote messages to mark (referring to as **remote mark messages**) are put on exported set again.

In our implementation on AP1000, physical pointers of objects are fixed and local collectors is mark-and-sweep type, but our distributed garbage collection algorithm is independent of local collector's type. If you select copying collector as local collector, you only have to extend exported set as export table to have physical pointers to objects and all remote references are to be done through export table.

3.3 Global Garbage Collection

We implement a DGC that works concurrently with mutators and does not block mutators.

To collect all garbage including cyclic ones ranging over nodes, we select a scheme that marks globally over nodes and reclaims memory on each cell. GC Host is assumed in our algorithm that manages the opening of global GC to avoid two or more global marking start simultaneously, and determines the termination of global marking. The algorithm of this global garbage collection is described in following subsection.

At global garbage collection, global GC collects garbages at global snapshots at starting of global GC. New created objects during GC are allocated as marked objects.

To do global GC correctly, we have to mark from root objects and active objects and their messages at the global snapshot. But some unprocessed messages of active objects sent before global GC starts may be pending in the network. We must confirm the arrival of these messages and mark from them. We describes about confirming scheme of the arrival of all pending messages sent before global GC starts in following subsection.

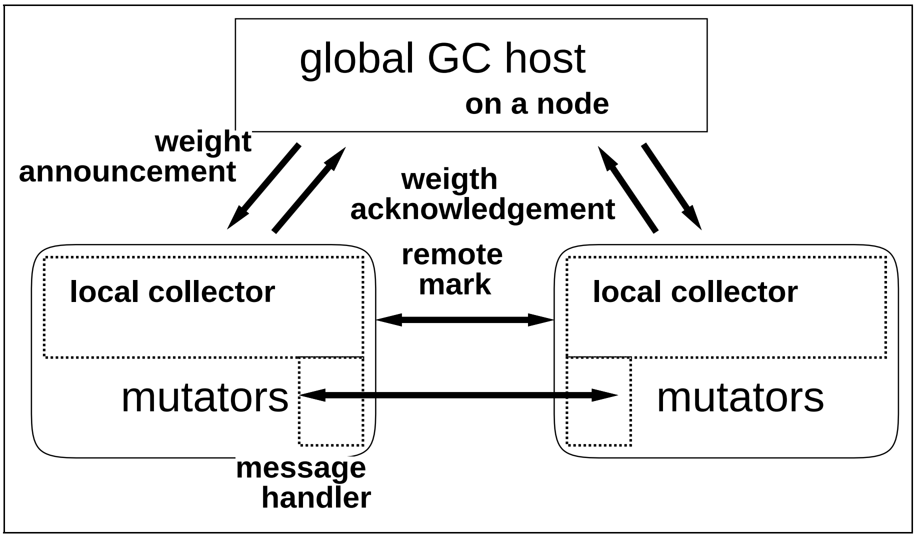

Figure 3.1: Overview of Global GC

Termination detection of marking over nodes is a factor to increase network traffic. We present a termination detection algorithm using weight, that does not send acknowledge messages informing terminations of markings from each object.

3.3.1 Overview of Our Global GC mechanism

Our global garbage collector consists of local collectors on each node and global GC host that manages the opening of global GC to avoid two or more global marking start simultaneously, and determines the termination of global marking. Marking over nodes is implemented as remote message passing between node local collectors.

A local collector has each **marking set** from that collector continue marking. Collectors have two sides state, **quiet** and **active**. Marking sets at starting of global GC are equal to root sets, root objects and active objects and their unprocessed messages. In marking object A, A is deleted from marking set and collector marks A and add referenced objects from A to marking set if A is not marked. A collector is active when marking set is not empty and it can continue marking, It becomes quiet when marking set becomes empty and it does not have object to mark at the situation.

When a local collector wants to start global GC, it sends message to GC host requesting starting of global GC. When GC host accepts the request, it broadcasts messages to all nodes announcing starting of global GC and giving weight (mentioned later). If another global marking is done, GC host rejects the request.

Local collectors on each node know the start of a global GC when announcement from GC host or remote mark messages from other nodes (or new color messages from other nodes in case of using message count scheme as described below) are arrived. They become active and set root sets and clear exported set on each node. (In case of using message count scheme, they send send_message_counts to host too, as described later.)

Local collectors start marking from each root set. When remote reference is found in marking, the collector sends remote mark message to owner node of referenced object to mark it. Receiving remote mark messages, collectors added object pointer in the messages to their marking sets and continue marking. When marking set become empty, it becomes quiet and some messages are sent to GC host as described below.

Ongoing computation is done concurrently with local collector as usual except the following points:

- Arrival messages are at first captured by the garbage collectors. For messages that may be sent before global GC start, local collectors add to their marking sets object pointer information included in them. (The ways of judgment whether messages are possibly old or not are different between message count scheme and bulldozing scheme as described **3.3.2**.)
- Writing on unmarked objects is checked and erased object pointer data information is added

to marking sets. (as described **3.3.4**.)

When GC host determines termination of global marking, GC host sends announcement messages to all nodes, and local collector starts to reclaim garbages on each node. Algorithm of termination detection is described in **3.3.3**.

3.3.2 Confirming the Arrival of All Pending Messages

Marking from all unprocessed messages of objects have to be done, in an asynchronous network we have to confirm the arrival of all pending messages sent before global GC start and mark from them.

A well known method of confirming the arrival of sending messages is to use acknowledgment message (referring to as ACK) that is sent from receiver nodes to sender nodes to inform sending message arrival. If we implement this naively, message traffic for ordinary computation increased twice and messages must include the information of sender node ID.

Another method is called network clear operations that sweep out old messages from network. Bulldozing method is one of network clear operation that needs no special hardware supports except the FIFOness of each network channel, as described in [10, 11, 12]. By sending bulldozing messages through network, we can confirm the arrival of all old messages when bulldozing finished.

This time, we implement our DGC algorithm through two schemes, a scheme that we call message count scheme (improved scheme of naive ACK method) and bulldozing scheme, and compared through implementation.

Our message count scheme colors all messages of ordinary computation to distinguish whether messages are sent before global GC start or after, and local collector count numbers of all sent messages and arrival messages by their message color, as sent_message_count and arrival_message_count.

When global GC start, GC host compares the sums of sent_messages_count and arrival_messages_counter in distributed system. In actual implementation, a local collector sends sent_message_counts of old color to GC host and sets the counter to zero at beginning of a global GC. When a local collector becomes quiet, it sends arrival_message_counts of old color to GC host and sets the counters to zero. If marking set increased from arrival of remote messages or so, the local collector sends arrival_message_count again at becoming quiet.

Message counting is done all the time and old messages may arrive to a quiet collector in global GC. In the case, the collector becomes active again. Marking to possibly old message is done with old color messages. For arrival of old color messages in global GC, collectors add object pointer information including the messages to their marking set and continue marking. Naturally local collectors send arrival_message_count at becoming quiet again.

GC host confirms the arrival of all pending messages sent before global GC start, if following condition is attained.

1. Sent_message_counts are received from all nodes.

2. Sums of sent_message_counts and arrival_message_counts are equal.

Safety property of above statements is obvious. **1** means sum of sent_message_counts become sum of all of them. As sum of arrival_message_counts increase monotonically, and sum of arrival_message_counts is not more than sum of all arrival_message_counts, which is equal to sum of all sent_message_count, **2** means all message arrived. Termination is also obvious because all sent_message_counts are sent at the beginning of global GC and for all arrival of messages arrival_message_counts are returned to GC host in finite time.

On the other hand, bulldozing schemes are described in detail in [10, 11, 12]. By sending bulldozing messages through network, we can confirm the arrival of all old messages when bulldozing finished. AP1000 has 2-dimension torus network and implementation of bulldozing is done considering with specific hardware architecture. Detail of our bulldozing algorithm is described in **5.3**. Bulldozing is implemented to work concurrently with global marking. In [11] messages are colored to distinguish whether messages are sent before global GC or not. In this studies this time we make two version of bulldozing algorithm, one use coloring information and another do not use colors and messages arrived before bulldozing finished are judged as possibly old messages.

3.3.3 Distributed Termination Detection of Global Marking

Another difficulties of mark and sweep type distributed garbage collection is to implement termination detection algorithm of global marking with light network traffic.

A naive termination detection algorithm uses acknowledgments for marking when marking from marked object as a root is terminates. Termination condition of this type algorithm is achieved when all root sets on each nodes have been acknowledged. However acknowledgment messages are sent for each remote mark message and latency of acknowledgment is long when marking is ranging several nodes. Memory space for remembering not acknowledged objects is necessary.

A well known termination detection algorithm is setting three colors for marking, in [1]. Acknowledgment for marking when all references of marked object are traversed. Black objects are alive and acknowledged objects. Gray objects are alive but not acknowledged objects. White objects are not marked objects. Termination condition of this type algorithm is achieved when all nodes have no gray objects. However acknowledgment messages are sent for each remote mark message. And messages are sent many times to gain termination condition.

Our termination detection algorithm uses weight and gives it to marking activities, remote

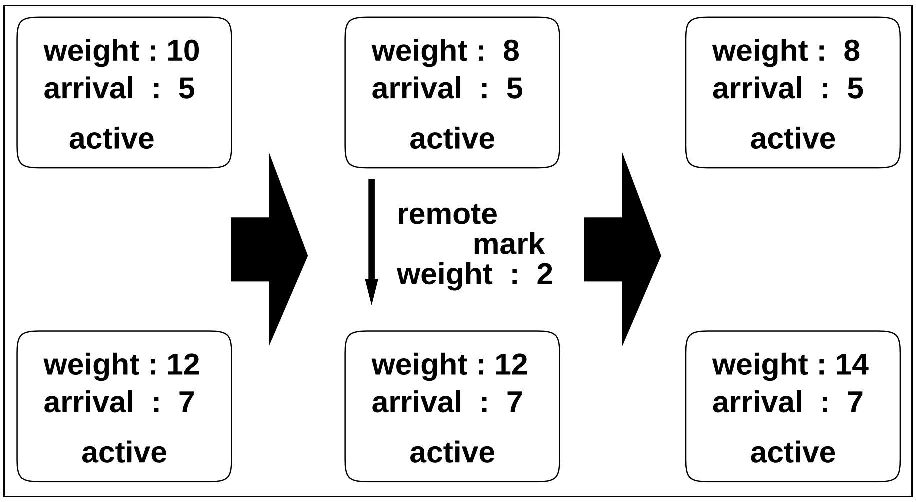

Figure 3.2: Remote Mark with Weight

mark messages and active local collectors as a unit. This weight is provided by GC host and each collector returns weight to GC host at becoming quiet. The sum of weight in network and local collectors on each node, that is sum of weight of remote mark messages and active local collectors, are equal to provided weight by GC host. Remote mark messages and active local collectors are kept to have positive weight in our algorithm (except the case described **3.3.4**. Naturally termination condition of our garbage collection is attained when GC host receive all weight provided before.

At staring of global GC, each node gets weight from global GC host with announcement message of starting of global GC and adds the weight to its weight. At sending remote mark messages, local collectors send them with positive weight and decrement their own weight. If collectors weight becomes zero by decreasing weight, sending remote mark is blocked until request of weight to GC host is done and enough weight is provided from GC host. Receiving remote mark messages, local collectors add weight of mark messages to their own weight and add referenced pointer into their marking set and continue marking. At becoming quiet, nodes return their weight to GC host.

A quiet local collector becomes active when possibly old messages or remote mark messages arrive. In case of remote mark messages, local collectors get messages' weight and become active

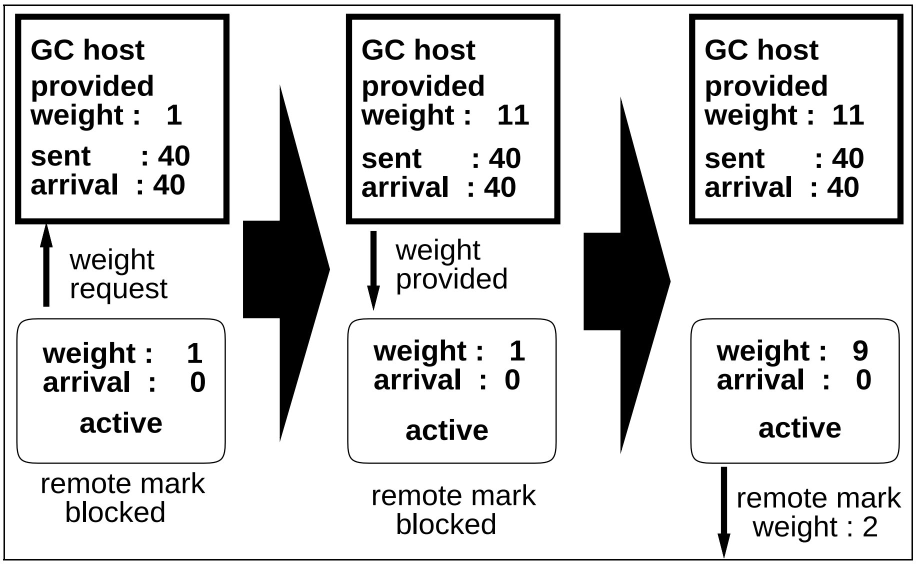

Figure 3.3: Requesting Weight to GC Host

with positive weight. In case of possibly old messages, local collector becomes active with zero weight.

Thus weight in network and all nodes are equal to weight provided by GC host. After confirming the arrival of all pending messages, weight of remote mark messages and local node collectors are kept positive because active collector with zero weight becomes quiet until confirming of arrival and no zero weight active collector will not be created after confirming. (More formal proof of this property is described in **4.3**.)

Termination condition of global marking is attained when following two conditions are achieved.

- Confirming the arrival of all pending messages is done.
- GC host received all weight providing before.

We now informally compare this algorithm with above mentioned ones using acknowledgment messages. This algorithm requires remote mark messages to have weight information but above ones require them to have pointer of referencing object instead. In our algorithm returning of

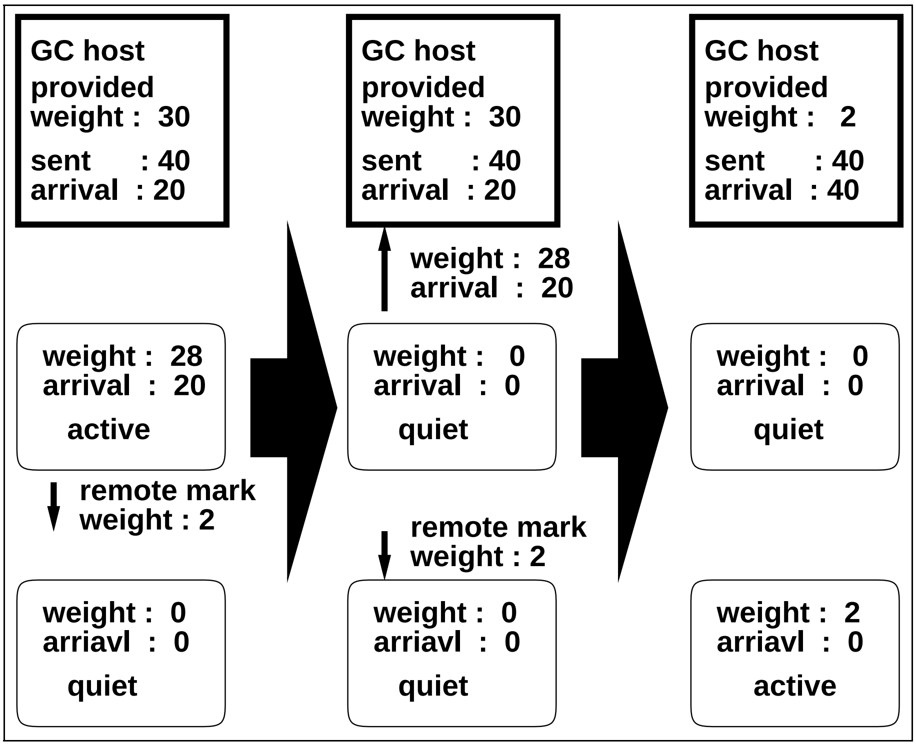

Figure 3.4: Remote Mark and Returning Weight

weight is done only when local collectors become quiet. Request message for weight and its reply is pure overhead messages to implement this termination detection algorithm. Comparing with above algorithm using acknowledgment messages for each remote mark message, message traffic is fairly lower. Furthermore, requests of weight occur seldom because weight is managed as a local collector as a unit. GC host judges termination of global marking only when weights are returned and do not have to send messages to watch local collectors' states. Considering of local collectors' overhead, in our algorithm collectors only have to deal weight and states for remote mark and have very simple data structure compared to above ones using acknowledgment.

3.3.4 Concurrent With Mutators

One of major problem on implementing concurrent GC with mutators is mutators' writing in unmarked space. Writing in unmarked space may change untraced reference relation of global

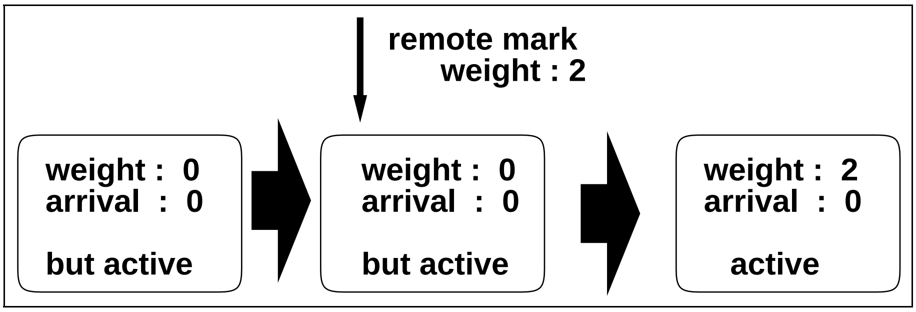

Figure 3.5: Active Collector with Zero Weight

snapshot at starting of global GC.

Two ways of coping with this difficulty are well known.

(1) Mutators are allowed to execute only in marked object whose reference is considered already.

(2) Conservatively marking from erased reference is done.

This time one of our goal of distributed garbage collection is not to block mutators and select **(2)** solution.

Some difficulties are raising from selecting **(2)**. Quiet local collector can become active without arrivals of remote mark messages or old messages. However this solution does not destroy our termination detection algorithm. We write informal proof here. (More formal proof is described in **4.3**)

Assuming of a written unmarked object O in node N. O is alive because it is accessible from live object, and if it is not unmarked it must be exist before global snapshot as new objects are created marked, and marked before termination condition is formed. Next we prove termination of marking from O as root before termination condition is formed. For marking from O as root out of N must be managed with positive weight. For marking from O as root in N must be finished because O has to be marked and the marking must be one from outside root (including old messages) and N has to receive old messages or remote mark messages. If the situation is achieved, following of this informal proof is same as described in **3.3.3**

Chapter 4

Algorithm

We formalize our algorithm based on discussions on chapter 3. Section 4.1 presents one version of our DGC algorithm that use message count scheme. Section 4.2 presents another version using bulldozing method. Section 4.3 describes proof of our algorithms.

In this chapter marking means traversing references of an object and add referenced object to marking set. Marking from A as root means traversing graph of reference relation as A as root and mark all referenced objects.

4.1 Algorithm : Using Message Count

This scheme uses message counters to confirm the arrival of all pending messages.

All mutators' messages have their colors used to be distinguished whether messages are sent before global snapshot or after. Each node has two kinds of message counter that counting the number of sent messages and arrival messages by message color. Each local collector has its color and sent messages are colored by its color.

Local collector and GC host's state are 2 sided, **normal** state and **marking** state. Local collectors in marking state with non-empty marking sets are called **active**, and other collectors in marking are called **quiet**.

4.1.1 Behavior in Normal State

Local Collector

- On sending a message, a local collector increments their sent_message_count. If the message includes local object pointers, they are added to exported set. The message is sent with color of the collectors.
- On receiving a message, a local collector increments arrival_message_count of message color. If message's color is not same as local collector's color, local collector must become active state in GC. The message is received after becoming active.
- On receiving remote mark messages or announcement of staring of global GC start, local collector becomes active. Remote mark message is received after becoming active.
- If a collector wants to start local GC, local collector starts garbage collection as described in **3.2**. If a collector wants to start global GC, it sends request message to GC host.

GC Host

- GC host becomes active state on receiving request of global GC.

4.1.2 Behavior in Global Marking

Local Collector

- At becoming marking state, collectors must change their color, clear exported set, set root set and send message count.
- Collectors with non-empty marking set continue marking from their marking set.

- Marking remote object, at first collector check the object is remotely marked from own node. If it is not marked form own node, collectors send remote mark message with positive weight. Collectors' weight decreased and , in the condition their own weights become zero, sending remote mark is blocked until request for weight is done and enough weight is provided from GC host.
- Sending a message is same as normal state.
- Receiving a message, a local collector increments arrival_message_count of message color. If message's color is old, object pointer included in messages are added to marking set.
- Receiving a remote mark message, weight of message is added to collector's one and object pointer included in messages is added to marking set.
- If new object is allocated, it is created as marked object.
- If mutator write in unmarked objects, erased pointer informations are added to marking set.
- If marking set become empty, collectors send weight and arrival_message_count of old color to GC host.
- Announcement of starting of global marking is discarded.
- Receiving an announcement of termination of global marking, each node starts to reclaim unmarked space. Collectors become normal state.

GC host

- At becoming marking state, GC host broadcasts announcement of starting of global GC.
- Requests to start global GC are discarded.

- Receiving sent_message_count, collectors add it to sum of sent_message_count. Number of receiving sent_message_count is counted.
- Receiving weight and arrival_message_count, collectors add them to sum of each. Continue to termination detector.
- Termination of global marking is detected if following two conditions are established.

  1. Sent_message_counts are received from all nodes.
  2. Sums of sent_message_counts and arrival_message_counts are equal.
  3. All weight is returned to GC host.

  If these conditions are established, GC host broadcasts messages to each node to start reclaiming of unmarked memory space. GC host becomes normal state.

4.2 Algorithm : Using Bulldozing Method

This scheme uses a bulldozing method to confirm the arrival of all pending messages. Bulldozing is done concurrently with global marking.

We present two algorithms, one uses message color to distinguish old messages and another do not use message color and messages received before an announcement of termination of bulldozing are judged as possibly old messages.

Local collectors and GC host have 2 sided state as message count scheme.

4.2.1 Behavior in Normal State

Local Collector

- Sending a message including local object pointers, they are added to exported set.

- Receiving remote mark messages or announcement of global GC start (or new colored messages at using color), local collector becomes active. The message is received after becoming active.

- If a collector wants to start local GC, local collector starts garbage collection as described in **3.2**. If a collector wants to start global GC, it sends request message to GC host.

GC Host

- GC host becomes active state on receiving request of global GC.

4.2.2 Behavior in Global Marking

For Bulldozing

- Bulldozing starts after cells become marking state.

- Termination of bulldozing is announced to all cells and GC nodes.

- Bulldozing algorithm depend on machine architecture and our algorithm using WO at implementing on AP1000 is presented in **5.3**.

Local Collector

- At becoming marking state, collectors clear exported set, set root set. Some kinds of acknowledgement is done for bulldozing.

- Collectors with non-empty marking set continue marking from their marking set.

- Marking remote object is as message count scheme.

- Sending a message is same as normal state.

- Receiving a message before arrival of announcement of bulldozing termination, local collectors add object pointers included in messages to their marking set. (If using color, local collectors add object pointers included in old colored messages to their marking set.)
- Collectors neglect messages after arrival of announcement of bulldozing termination.
- Receiving remote mark messages, weights of messages are added to collector's one and object pointers included in messages are added to marking set.
- New objects are created as marked object.
- If mutators write in unmarked objects, erased pointer informations are added to marking set.
- If marking set become empty, collectors send weight to GC host.
- Receiving announcement of termination of bulldozing, collectors send acknowledgement message for this announcement. If collectors have no weights, acknowledgement is done after requesting weight is done and weights are provided.
- Receiving bulldozing messages, collectors play their role in bulldozing algorithm.
- Announcement of starting of global marking is discarded.
- Receiving announcement of termination of global marking, each node starts to reclaim unmarked space. Collectors become normal state.

GC host

- At becoming marking state, GC host broadcasts announcement of starting of global GC.
- Requests to start global GC are ignored.

- Receiving weight and arrival_message_count, collectors add them to sum of each. Continue to termination detector.

- Receiving an acknowledgement message for announcement of bulldozing termination, termination detector is continued.

- Termination of global marking is detected if following two conditions is established.

  1. Acknowledgment messages for announcement of bulldozing termination are received from all nodes.

  2. All weight is returned to GC host.

  If these conditions are established, GC host broadcasts to all nodes to start reclaiming of unmarked memory space. GC host becomes normal state.

4.3 Correctness Proof

4.3.1 Correctness and Assumption

We must satisfy following properties to prove the correctness of our distributed garbage collection algorithm.

- **Safety Theorem** :

  1. Non-garbage objects will not be collected during garbage collections.

  2. User programs will not be influenced by garbage collections.

- **Liveness Theorem** :

  1. Garbage collection algorithm terminates in finite time.

  2. Every Garbage Object will eventually be collected.

To prove our distributed garbage collection, we use following assumption.

- Assumption : Garbages will not become live objects again.

We also assume about network as follows:

- Network Assumption : All messages arrive at their destination in finite time.

4.3.2 Lemmas around Confirming the Arrival of Old Messages

This section presents and proves following lemmas around confirming of old messages used in proof of correctness.

> **Lemma 1.1.1** : At message count scheme or bulldozing scheme using color for messages, a consistent global snapshot has been obtained by our algorithm.
>
> **Lemma 1.1.2** : At bulldozing scheme not using color for messages, a consistent global snapshot has existed.
>
> **Lemma 1.2** : After confirming the arrival of all old messages, no old message is in network.
>
> **Lemma 1.3** : Marking from all old messages is done.

Lemma 1.1.1 : At message count scheme or bulldozing messages using color for messages, a consistent global snapshot has been obtained by our algorithm.

Proof :

When a remote message or an announcement of starting of global marking or a new colored message arrives, local collectors changed their color and receive the message. Messages from new colored collectors are received by new color collectors only and consistency is gained. Announcement of starting global marking is received before announcement of termination of global marking and its arrival is acknowledged during global marking phase. As announcement of starting of global marking arrives in finite time, snapshot is obtained in finite time.□

Lemma 1.1.2 : At bulldozing scheme, a consistent global snapshot has existed during marking phase.

Proof :

Bulldozing scheme guaranteed the fact that no message sent from nodes before bulldozing arrives at nodes after bulldozing, and global time of a node before bulldozing is earlier than one of another node after bulldozing. We can get consistent global snapshot by following definition.

- Global time before bulldozing is before global snapshot.
- Global time after bulldozing is after global snapshot.
- If a node receives a message sent after global snapshot, global time of its node is after global snapshot.
- If a node's global time is not after global snapshot with above algorithm, global time of its node is before global snapshot.

A snapshot above definition has consistency. Supposing the condition where consistency is broken by a message sent after global snapshot arrives at a node before global snapshot, the node is before bulldozing. This makes contradiction with guarantee of bulldozing method. □

Lemma 1.2 : After confirming the arrival of all old messages, no old message is in network.

Proof on message count scheme :

From Network Assumption, sent messages arrive at destination. It is obvious that sum of sent messages' number is equal to sum of arrival ones. Our algorithm confirms the arrival of all old messages if following conditions are attained.

- Sent message counts are returned from all nodes.
- Sums of sent message counts and arrival message counts are equal.

Sent message count is returned when local collectors on each node change their colors and sent message count of old color never increase. Sum of arrival message count is increased monotonically and when sum of arrival message count is equal to sums of sent ones all arrival message count is returned. This guarantees the arrival of all old messages.□

In actual implementation message color needs only 2 color as changing color is not done before confirming of old objects.

Proof on bulldozing scheme :

Bulldozing algorithm may changes and informal proof of this time bulldozing is done in section 5.3.

Lemma 1.3 : Marking from all old messages is done.

Proof on message count scheme :

In this scheme, all messages are colored. As local collectors checking message color and marking from old messages before the arrival of all old messages are confirmed. From Lemma 1.2 no old messages are in network after confirming all messages' arrival. Local collectors continue checking messages until termination of global marking is announced, and no old message will arrive in future. This guarantees the fact that collectors mark from all old messages. □

Proof on bulldozing scheme :

In our bulldozing scheme, local collectors mark from all messages before bulldozing terminates. Therefore marking from all old messages is done. □

4.3.3 Proof of Safety Theorem 1

We also use following lemmas to prove safety theorem 1.

Lemma 2.1 : All live objects will be marked in global marking in our algorithm.

Lemma 2.2 : Local collectors acknowledges arrival of old messages with non-empty marking set and no weight.

Lemma 2.3 : Termination condition is not achieved until global marking terminate.

Lemma 2.1, Lemma 2.2, Lemma 2.3, Safety Theorem 1 are proved in this order.

In our DGC live objects are defined as follows.

- Active objects are live objects.
- Root objects are live objects.
- Objects are alive if other live objects know their object address.

Our algorithm marks following objects.

1. Active objects and root objects are marked as root set.
2. Objects referenced by messages of global snapshot are marked.
3. Objects referenced by marked objects are marked.
4. When mutators write in unmarked objects, the written object and referenced objects by it are marked.

Lemma 2.1 : All live objects will be marked in global marking in our algorithm.

Proof

It is obvious that all old live objects are traced on global snapshot with 1 - 3 type marking only. Our algorithm works concurrently with mutators. To avoid any untraced reference to be erased, erased references are traced conservatively before reference relation is changed. We traced reference relations on global snapshot only and no unmarked reference relation is erased during marking phase, all live objects at global snapshot are marked using Lemma 1.3. Old live objects were live at global snapshot using above Assumption and marked during marking phase. As new object is created as marked object during GC, it is proved that all live objects are marked during phase. $\square$

Lemma 2.2 : Local collectors acknowledge arrival of old messages with empty marking set or positive weight.

Proof :

In message count scheme, local collectors acknowledge arrival of old messages only when their marking set become empty. In bulldozing scheme, local collectors with non-empty marking set acknowledge announcement of termination of bulldozing with positive weight. If a local collector has no weight with non-empty marking set, acknowledgment is done after collectors request weight and GC host provides weight. In both cases Lemma 2.2 is kept. $\square$

We define some sets for proof of Lemma 2.3.

- Old marked objects are member of set A if it can be traced from root set or old messages with references traced during marking phase.
- Old marked objects are member of set B if it is not member of A.

Lemma 2.3 : Termination condition is not achieved until global marking terminate.

Proof Weights in network and in local collectors are kept equal to weight provided by GC host in our algorithm. As weights do not have negative value, Returning all objects to GC host guaranteed the fact that no remote mark messages are in network and no collectors have positive weights.

Local collectors' weight can be zero only if its marking set become empty and collectors return weights to GC host.

Assuming the situation where terminate condition is achieved and some local collectors continue marking, the collector must continue marking with no weight.

A local collector with no weight marked an object O as member of set A, O is traced from any root objects or old messages.

Case 1 : Suppose O can be traced from root set of same nodes, O is marked before marking set becomes empty and the local collector has positive weight. This supposition makes contradiction.

Case 2 : Suppose O can be traced from old messages arrived at this node, O is marked before marking set becomes empty. This means that O is marked by collectors with no weights and non-empty marking set. This supposition contradicts with Lemma 2.2.

Case 3 : Suppose O can be traced from root set at other nodes or old message arrived at other nodes, a traced reference exists that can reach to O. Remote marking to this node was done and O will be marked until marking set becomes empty. However, local collectors on the nodes receive positive weight from remote mark message and keep positive weight until marking set becomes empty. This supposition makes contradiction.

Therefore we can say O is not member of A. O is marked objects and O must be traced from unmarked objects written in by mutators. WO must be live objects and member of A. Suppose WO and O is not at same nodes, some remote references exist between O and WO and contradiction occurs as case 3. Suppose WO and O is at same nodes and reference to WO was traced by collectors with positive weights, O must have been marked before marking set becomes empty by collectors with positive weight. This also makes contradiction. Supposing that WO will be marked by collectors with no weight after confirming the arrival of all old messages, and same contradiction occurs as Case 1 - 3.

Consequently we can say first assumption is not correct and Lemma 2.3 is proved.□

Safety Theorem 1 : Non-garbage objects will not be collected

Proof :

Using Lemma 2.1 and Lemma 2.3, all live objects are marked during global marking phase and as local collectors do not collect marked objects, all live objects will not be collected.

4.3.4 Proof of Safety Theorem 2

Safety Theorem 2 : User programs will not be influenced by garbage collections.

Proof

Our algorithms do not changed reference relations of objects and not block marking and not collect live objects from Safety Theorem 1. Our algorithm will not influence user programs.

4.3.5 Proof of Liveness Theorems

Liveness Theorem 1 : Garbage collection algorithm terminates in finite time.

Proof :

We prove it according to following steps. (1) finite number of markings as reference traversal is done. (2) markings are done in finite time. (3) termination detection is done in finite time.

This time we trace only old objects. It is obvious that number of messages in network and objects of global snapshot at starting global marking. All messages and objects have finite number of reference relation and Number of reference relations at global snapshot is finite. Once a reference is traced, it is not traced twice and marking is done in finite number of times at most.

Marking has two types, local marking and remote mark. Marking are blocked only when local collectors do not have enough weight. At this case providing of weight is done in finite time as it is not blocked. Remote mark messages arrive in finite time as Network Assumption, Each type of marking is done in finite time. In addition all old messages arrive in finite time, all marking is done in finite time.

(3) is proved at each algorithm, message count scheme and bulldozing scheme. As message counts scheme, local collectors have no weight and no message counts when marking set is empty. At bulldozing scheme, local collectors have no weight when marking set is empty and as bulldozing messages are not blocked by local collectors or mutators bulldozing terminate in finite time. For both cases, as marking is done in finite time and all weight and message count is returned to host, GC host noticed achievement of terminate condition, if no message is in loss in network as Network Assumption.

Liveness Theorem 2. : Every Garbage Object will eventually be collected.

Proof :

If an object is garbage at global snapshot at starting global marking, it is not accessible from any live objects or messages and it is not marked during marking phase.

If an object becomes garbage during global marking phase, it is marked and not collected during the marking phase. However it is garbage at beginning of next global GC, and collected by the global GC or previous local GC.

Chapter 5

Performance Evaluation

This chapter presents implementation of our distributed garbage collection algorithm on AP1000 and evaluate two schemes, message count one and bulldozing one are compared.

5.1 Machine Environment

We chose AP1000 as a target machine. AP1000 consists of 64 - 1024 cells and host computer as SUN-4/330 one CPU workstation. Each cell has one SPARC processor with 16 MByte memory and 25 MHz cycle clock.

Cells and host are connected with T-net and B-net and S-net. B-net is a broadcast network that is used communication between host and cells. S-net is for synchronization between cells. These two networks are not used in our implementation. T-net is a two-dimension torus network for communications between cells. Node to node FIFO communications and X-dimension, Y-dimension, XY-dimension broadcasts are provided. It is controlled by the routing chip called RTC [3]. RTC's transfer rate is 25MB/s per channel (connection between next node) and its transfer delay is $160 + 160 \times distance + 160 \times message - length(word)$ (ns).

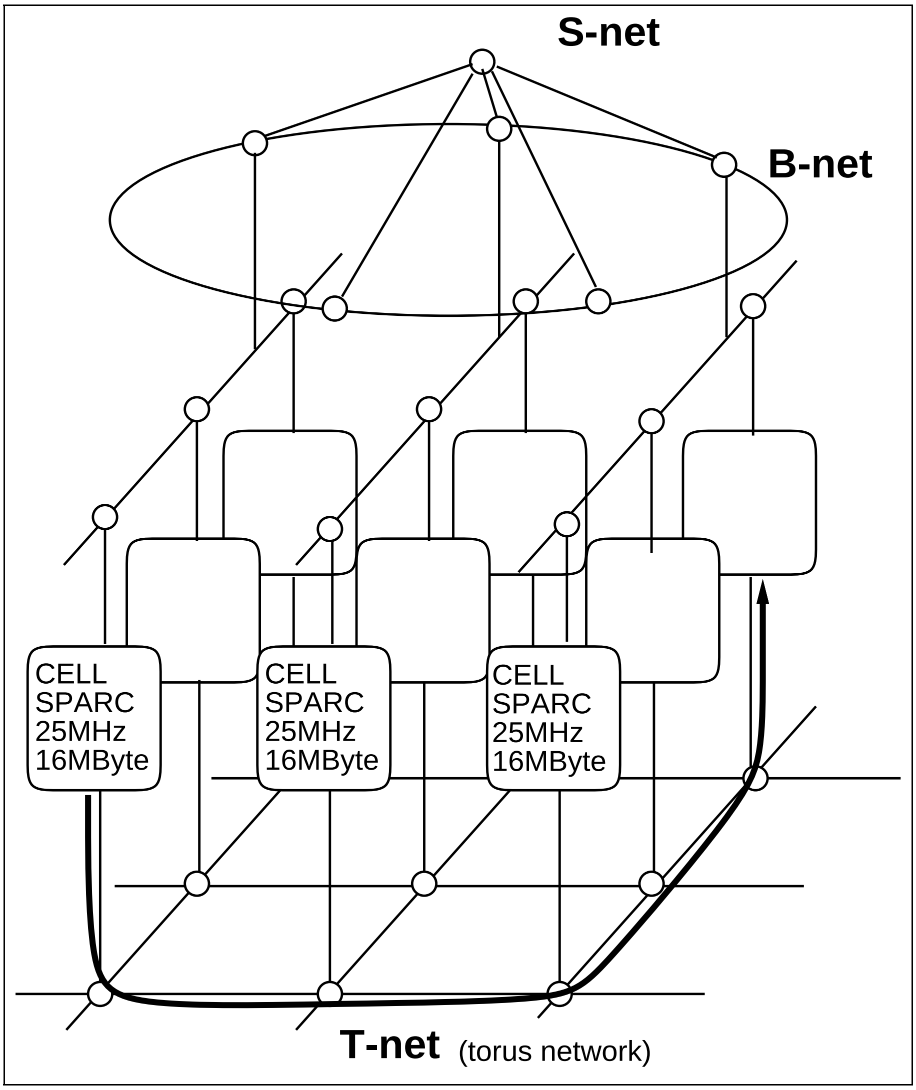

Figure 5.1: AP1000 Architecture

5.2 Memory Allocation

Our local collectors have 2 generations to reduce overhead of GC. When local collector does GC over generation, generations of all live objects are raised to old generation. On global garbage collection we can not get consistency between each node's generation, GCs over generation are done.

AP1000 does not have enough memory protection mechanism to do incremental copying GC because it is not actual to check whether it is marked or not for all reading. Our collector seldom change addresses of objects.

Objects are only space that is referred from other nodes, their addresses are not changed. Their allocation is done with fixed size chunk with bidirectional link. Marking to chunks are done with copying from new space to old space with changing link and no sweeping is needed and We can expect effects of doing generational GC with requiring no sweeping.

We can use heap to allocate arrays and records. References to heap area is managed through indirection tables to enable compaction at GC over generation. GC for heap area is done with mark and sweep. we can expect effects of doing generational GC with compacting old generations.

Each allocated object has tag bit for GC. Two tag bit is used to represent contents of one word, as contents have 3 data type, object pointer or heap or non-pointer.

5.3 Bulldozing Algorithm

Routing scheme of AP1000's B-net is as in [3] and FIFOness of communication between nodes are guaranteed. Messages are routed through X-dimension and then Y-dimension. Shorter side of torus network is used.

AP1000 is implemented as messages may pass messages from another node in physical network.

To bulldozing X-dimension messages, it is not enough to send bulldozing messages to go around. This time we use following algorithm to do bulldozing because messages are not passed by ones from the same nodes in AP1000. Node's position is presented as (X, Y). GC host is assumed to be $(0, 0)$.

1. GC host broadcasts announcement messages of starting of global GC and bulldozing to all nodes.

2. All nodes send bulldozing messages in two directions, $+$ and $-$ of X-dimension by half distance of X-dimension to bulldoze messages from each node.

3. Receiving X-dimension bulldozing messages, nodes positioning (X, Y) send acknowledgement messages to nodes in position $(0, Y)$.

4. Receiving all the acknowledgements of X-dimension bulldozing messages, node $(0, Y)$ knows termination of X-dimension bulldozing on this row. They send X-dimension broadcast messages to start Y-dimension bulldozing.

5. Receiving messages to start Y-dimension bulldozing, each node sends bulldozing messages in two directions, $+$ and $-$ of Y-dimension by half distance of Y-dimension to bulldoze messages from this row.

6. Receiving Y-dimension bulldozing messages, nodes positioning (X, Y) send acknowledgement messages to nodes in position $(X, 0)$.

7. Receiving all the acknowledgements of Y-dimension bulldozing messages, node $(0, Y)$ knows termination of Y-dimension bulldozing on this column. This guarantee no old messages will

arrive in this column in the future. Nodes announced termination of Y-dimension bulldozing on this column to local collectors on this column.

8. Receiving announcements, local collectors acknowledge to host. If local collectors have no weight and active in marking, local collectors have to request to GC host and then send acknowledgement messages.

9. Receiving all acknowledgement messages, GC host confirms the arrival of all pending messages.

We present informal proof of unproved lemmas in section 4.3.1.

Lemma 1.2 : After confirming the arrival of all old messages, no old message is in network.

Proof

Our algorithm depends on routing algorithm of AP1000 [3]. We use following features.

- Messages routed X-dimension and next Y-dimension.
- Messages routed through shorter pass for each dimension.
- Messages do not pass by another messages from same node.

At step 1, we confirm no old messages are send from each cell. At step 4, it is confirmed that no messages exist X-dimension network of each row. At step 7, it is confirmed that no messages exist Y-dimension network of each column, and no old messages are arrived at nodes of each column. At step 8, local collectors stop checking messages. At step 9, GC host confirms that no old messages are in network.

5.4 Implementation

This section presents a way of representing concurrent object-oriented programing environments used in this time implementation and presents implementation of DGC algorithm.

We use Remote Continuation Address Passing for remote messages as presented in [9]. A received message has only specifying a message handler address (remote continuation address) that depends on each specific message. In our DGC implementation, a message handler also check messages as a part of local collectors. In message count method, message handler check message's colors and increment arrival_message_count by each color. At global marking, marking from possibly old messages are done by message handler, in both cases, message count scheme and bulldozing scheme. As message handler knows tag information of messages, messages need not to have tag informations.

We also use low latency remote object creation mechanism presented in [9]. Each node gives to each node stocks of chunks for remote creation. At remote object creation, nodes get a chunk from given chunks and return its address as remote created objects address and send messages to create object to owner of chunk.

In our implementing DGC, local collectors and GC host communicates with message passing, each remote message are handled by each type message handler using remote continuation address mechanism.

This time we hand-compile N-queen programs to evaluate performance of DGC algorithm and compare two schemes, message count scheme and bulldozing scheme. However our DGC is on experiment stage and it is difficult to identify data type at stack, no method call is implemented using stack. Each method call is implemented as allocating a chunk and saving parameters and put the chunk to continuation queue.

5.5 Evaluation

Following evaluation is done at AP1000 with 64 (8×8) cell.

At first we evaluate overheads of global marking of each type algorithm, by measuring time of global marking in environments with no live objects. The result is as follows.

Table 5.1: Global Marking Time with No Live Object

message count method	bulldozing method	(for bulldozing only)
7.4(msec)	14.8(msec)	14.8(msec)

Next we evaluate numbers of messages to do global GC and detect its termination. This performance is done with running N-queen program.

At first we examine the case of message count scheme as shown in 5.2. Numbers of weights request and acknowledgments for termination are less than remote marking messages at averages. We can detect the termination of global marking with these messages only. We can say that our algorithm require less messages than using acknowledgement for each marking termination. According to weight request, it is quite less than remote message counts in this case.

Table 5.2: Messages at Global Marking : Message Count Scheme

	average	max	min
remote references	58615	120340	68
remote mark messages	37892	90202	58
marked object	234837	544209	267
old pending messages	2.23	11	0
global marking time	366.6(msec)	824(msec)	38(msec)
weight request	0.02	3	0
returing of weights	5016	13527	199
overhead msg/ remote msg	0.13		

Next we examine the case of bulldozing scheme, 5.3 shows number of messages at messages do not have their color and 5.4 shows the case of no color for messages. Comparing these two versions, we can find few deferents. Oppositely in both case bulldozing time is quite similar to global marking time. This means that terminations of global marking are often restricted by terminations of bulldozing. This occurs in the probability of 70This fact does not indicate bulldozing is not actual, because 5.1 shows that bulldozing can complete in short time. Each bulldozing message is sweeping via some nodes and at each nodes the message waits for being processed by message handler. Latency becomes long as each cell or network traffic become heavy. If some hardware supports that relay bulldozing message with low latency or that interrupts for some kind of remote messages as bulldozing, bulldozing becomes more actual.

Table 5.3: Messages at Global Marking : Bulldozing Scheme (with No Color)

	average	max	min
remote references	49284	131154	5
remote mark messages	36383	104503	5
marked object	239218	552378	41
bulldozing time	380(msec)	887(msec)	31(msec)
global marking time	394 (msec)	887(msec)	35(msec)
weight request	0.015	1	0
returing of weights	4088	18743	68
bulldozing messages	320		
overhead msg/ remote msg	0.12		

Table 5.4: Messages at Global Marking : Bulldozing Scheme (with Color)

	average	max	min
remote references	45420	117447	101
remote mark messages	32096	88442	81
marked object	233612	536618	475
global marking time	387(msec)	810(msec)	34(msec)
bulldozing time	380(msec)	810(msec)	36 (msec)
weight request	0.017	1	0
returing of weights	3309	15148	136
bulldozing messages	320		
overhead msg/ remote msg	0.11		

In both cases, the number of messages which are sent for returning weights or requesting for weights is 11 - 13However, our DGC using bulldozing scheme takes more time than message count scheme, because remote marking sometimes terminates before the termination of bulldozing. Using bulldozing scheme on our algorithm is superior in the point that algorithm using bulldozing scheme can work without colors and any overheads are needed in sending remote messages. If bulldozing or some kind of network clear operations are supported, using bulldozing method in our DGC will be faster ones.

Chapter 6

Conclusion and Future Work

This thesis presents an distributed garbage collection algorithm that have local GC and mark-and-sweep type global GC, and not block mutators. To reduce overhead of global marking we use a message count scheme and a bulldozing scheme that do not send ACK messages to sender nodes. A termination detection algorithm is used that use weight as local collectors as units.

Our DGC algorithm gives weight to each marking activities and returning of weight or requesting for weight or confirming the arrival of all old messages are done as overhead. In our DGC algorithm, it is ascertained that number of messages for above overhead are less than that of remote mark messages through actual implementations. For our algorithm using bulldozing method, termination of remote marking sometimes catch up with bulldozing termination and bulldozing often becomes the bottleneck of our algorithm in our implementation on AP1000. This is because bulldozing messages often waits for being handled by processors. As bulldozing methods does not require coloring messages, it may be in actual use when some hardware supports are offered.

Our DGC algorithm is not fault tolerant because the termination condition of global marking in our algorithm does not consider any kind of fault. If message are lost, arrival_message_count or weight for GC are missed and our termination condition is not satisfied. Local collectors are waiting for GC termination, and can not start other GC. We have to devise a system recoverable from fault. We also need to consider reducing global marking overhead. Global marking over many nodes requires overhead and to reduce this overhead some sort of hierarchical system is required. Since our DGC does not require messages to have any information about sending nodes, some information must be carried in messages to do hierarchical DGC.

References

[1] Lex Augusteijn. Garbage collection in a distributed environment. In *Lecture Notes in Computer Science*, volume 259. PARLE Parallel Architectures and Languages Europe II, Spinger-Verlag, 1987.

[2] D I Bevan. Distributed garbage collection using reference counting. In *Lecture Notes in Computer Science*, volume 259. PARLE Paralell Architectures and Languages Europe Conference, Spinger-Verlag, 1987.

[3] Takeshi Horie, Hiroaki Ishihata, and Morio Ikesaka. Design and implementation of an interconnection network for the ap1000.

[4] Eric Jul, Henry Levy, Norman Hutchinson, and Andrew Black. Fine-grained mobility in the emerald system. ACM Transactions on Computer Systems, 2 1988.

[5] Bernard Lang, Christian Queinnec, and Jose Piquer. Garbage collecting the world. POPL, 1992.

[6] Niels Christian Luul and Eric Lul. Comprehensive and robust garbage collection in a distributed system. In Yves Bekkers and Jacques Cohen, editors, *Memory Management*, volume 637 of *Lecture Notes in Computer Science*. Spinger-Verlag, 1992.

[7] Lsabella Puaut. Distributed garbage collection of active objects with no global synchronization. In Yves Bekkers and Jacques Cohen, editors, *Memory Management*, volume 637 of *Lecture Notes in Computer Science*. Spinger-Verlag, 1992.

[8] Marcel Schelvis. Incremental distribution of timestamp packets: A new approach to distributed garbage collection. October 1989.

[9] Kenjiro Taura, Satoshi Matsuoka, and Akinori Yonezawa. An efficient implementation scheme of concurrent object-oriented languages on stock multicomputers. In *Proceedings of the ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming PPOPP*, 1993. to appear.

[10] Nalini Venkatasubramanian. Hierachical garbage collection in scalable distributed systems. Master's thesis, University of Illinois at Urbana-Champaign, 1992.

[11] Nalini Venkatasubramanian, Gul Agha, and Carolyn Talcott. Hierarchical garbage collection in scalable distributed systems. Technical Report UIUCDCS-R-92-1740, University of Illinois at Urbana-Champaign, Department of Computer Science, 1992. UILU-ENG-92-1720.

[12] Nalini Venkatasubramanian, Gul Agha, and Carolyn Talcott. Scalable distributed garbage collection for system of active objects. In Yves Bekkers and Jacques Cohen, editors, *Memory Management*, volume 637 of *Lecture Notes in Computer Science*. Spinger-Verlag, 1992.