Fully Concurrent Garbage Collection of Actors on many Cores
S. Clebsch, S. Drossopoulou

Jesse Badash

Background

Actor Model Languages:

- Actors are the basic unit of computation
- All communication is based on message passing
- Actors encapsulate memory, computation.
- Fantastic for concurrency

Background

Garbage Collection

- Some languages provide you with the ability to explicitly end the actors lifecycle.
- Others utilize hardware for poor performance
- Focus on many small, short lived actors

Goals

Create a garbage collection system:

- Soundness: Only collect dead actors
- Completeness: All dead actors are eventually collected
- Concurrency: Does not require stopping, shared memory, thread coordination etc.

Terminology

- Blocked - Actor’s state when it has complete execution and have no messages left. Send BLK with reference count and external set
- Dead - Actor’s state when it is blocked and all of the actors with a reference to it are blocked.

Current Solutions

- Many actor languages do not garbage collect, you are expected to.
- Convert Actor graph, to an object graph, and run known algorithm in ActorFoundry
- SALSA uses reference listing

Topology

- Graph of all the actors
- Each actor has a “local topology”
- Eventually it will be correct

Deferred Reference Counting

- External Set: An over approximation All references an actor contains
- Allows lazy reference counting
- “Local garbage collection”

Cycle Detection

- When an actor blocks it send BLK
- When it processes a message, it send UNB
- Cycle detector is eventually correct
- If a ‘dead cycle’ exists it will be removed
  - If there is a conflict, the cycle is not removed

Conf-Ack

- Cycle Detector sends a CNF to the members of a perceived cycle
- If the topology agrees actor sends ACK
  - If the cycle detector receives a UNB before an ACK all perceived cycles are updated
- Provides confirmation in a truly concurrent way

Casual Messaging

- Messages adhere to casual messaging
- Sending a message and enqueuing it can be done in a single atomic operation

Formal Model

- Operational semantics for MAC
- Each is accompanied with a formal logic representation of the rule.
- Used for proofs and proper representation

Completeness

- If a cycle exists, every actor has sent BLK
- Eventually the cycle will be recognized
- If all workers are blocked, then all cycles will be found
- Terminates when:
  - no actors are executing
  - the message queue is empty
  - no cycles are detected

Robustness

- Upon failure of Cycle detector, garbage is not collected, but no live actors are collected
- Upon failure of actor, that cycle may stay alive, but all other cycles will be removed
- Termination conditions can still be reached

Soundness

- When every actor in a PC has confirmed, it is infact a true cycle
- When the cycle detector’s view of topology of the actors in the cycle is true, the PC is true
- When an actor confirms a PC, the actors topology, and its view of its topology agreed.

Implementation

- Library written in C
  - Error prone, not type safe
- In use at a large financial institution
- Scales linearly with increase of core count
- Casual messaging is easy on a single host

Comparison

- Tests against Erlang, Scala, libeppa, and 3 versions of MAC:
  - Disabled Cycle detection
  - Normal Cycle detection
  - Forced Cycle detection

Comparison

- Message handling
  - 3 million messages, 2 cores
- Actor creation
  - 2^19 actors, 2 cores
- Mailbox performance
  - 20 million messages, 4 cores
- Mixed
  - 50 million messages, 1000 actors, 4 cores

Limitations

- Seems unfair to expect erlang is a full language
  - This garbage collecting is completely useless in distributed systems.
- Demonstrate on a system with many short-lived actors