Distributed Algorithms

Assignment 05

Group 11	Xugang Zhou	352032
	Fangzhou Yang	352040
	Yuwen Chen	352038

November 27, 2013

1 Mutual Exclusion

2 Distributed Garbage Collection

- JAVA RMI: RMI uses a fairly straightforward mechanism for garbage collection, which is explained in detail in [1]. Any program which has a reference to an object must obtain a "lease" for the object.The lease, which is literally represented by a Lease object, entitles the program to use the object for a certain period of time.
- Microsoft DCOM: DCOM differentiates between interfaces and objects. Each has its own type of garbage collection support.
  - Interfaces: Garbage collection of interfaces is handled through a manual reference counting mechanism. RemAddRef and RemRelease respectively add and release references to remote objects. To reduce the overhead by "multiplexing references", a single reference can actually stand for many references within a single program.
  - Objects: Garbage collection of Objects send keepalive messages periodically to objects as a way of pinging the objects to let them know they are still needed.

3 Reference Counting

4 Mark and Sweep

Mostly-concurrent algorithm [2] is aimed to avoid to freeze the system permanently during the garbage collection. The general idea is to trade off complete concurrency for better throughput, by allowing root locations to be written without using a barrier. The algorithm comprises four phases, which can be found in [2]:

- Initial marking pause: Suspend all mutators and record all objects directly reachable from the roots (globals, stacks, registers) of the system.
- Concurrent marking phase: Resume mutator operation. At the same time, initiate a concurrent marking phase, which marks a transitive closure of reachable objects. This closure is not guaranteed to contain all objects reachable at the end of marking, since concurrent updates of reference fields by the mutator may have prevented the marking phase from reaching some live objects. The algorithm also arranges to keep track of updates to reference fields in heap objects.
- Final marking pause: Suspend the mutators once again, and complete the marking phase by marking from the roots, considering modi

  ed reference fields in marked objects as additional roots. Since such fields contain the only references that the concurrent marking phase may not have observed, this ensures that the final transitive closure includes all objects reachable at the start of the final marking phase. It may also

include some objects that became unreachable after they were marked. These will be collected during the next garbage collection cycle.

- Concurrent sweeping phase: Resume the mutators once again, and sweep concurrently over the heap, deallocating unmarked objects. Care must be taken not to deallocate newly-allocated objects. This can be accomplished by allocating objects "live" , at least during this phase.