

Remote Objects: The Next Garbage Collection Challenge*

Mulyadi Oey, Witawas Srisa-an, Sebastian Elbaum

ESQuaReD Laboratory

Department of Computer Science and Engineering

University of Nebraska – Lincoln

*Partially supported partially by the NSF under award CNS-0411043

OUTLINE

- MOTIVATION
- OBJECTIVE
- BACKGROUND
- IMPLEMENTATION
- BENCHMARK
- RESULT
- DISCUSSION
- CONCLUSION
- FUTURE WORK

MOTIVATION

- More large-scale server and distributed applications are developed in systems that rely on efficient garbage collection (e.g. Java and .NET) [Nutt05, Ram02, Mason].
- Garbage collector performance depends on the object behavior.
 - Thorough study of local objects.
 - Currently, there is no profile of the behavior of remote objects.
- Today's distributed systems are based on the old norm: most objects die young.
 - .NET and JDK use generational garbage collection.
 - "Does the old norm still hold?"

OBJECTIVE

- Specifically, we are driven to answer the following research questions (RQs):

RQ#1: Is the lifespan of remote objects different from local objects?

RQ#2: What are the effects of remote objects on generational GC?

BACKGROUND: SSCLI

- Shares the same code base with the commercial .NET Common Language Runtime (CLR) [Stutz03].
 - Subset of CLR.
 - Full support of remoting, threading, JIT compilation, and GC.
 - No support for ADO.NET and ASP.NET.
- Garbage collection
 - 2-generation heap.
 - Young or ephemeral generation (gen0): copying collection.
 - Mature generation (gen1): mark-sweep collection.
 - Objects are initially allocated in the young generation.
 - Survived objects are promoted (copied or moved) to the mature generation.
 - Few survived objects = less costly.

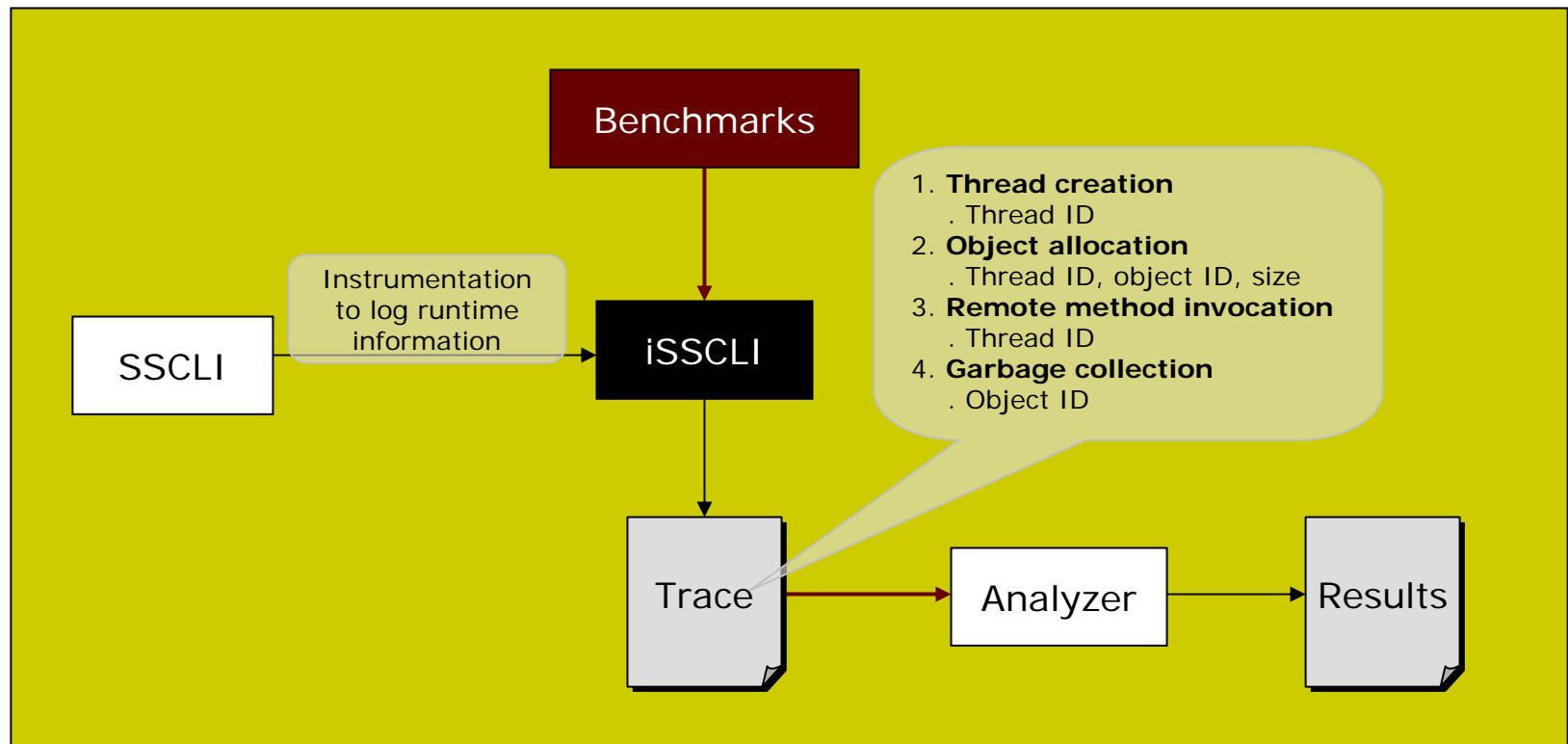
BACKGROUND: REMOTE OBJECTS

- .NET Remoting
 - Similar to Remote Method Invocation (RMI) in Java.
 - A framework for objects to interact with one another across application domains [McLean03].
 - Via remotely-accessible (remote) methods.
 - Used increasingly in distributed systems [Nutt05, Mason].
 - COM+ Web Services.
 - Legacy distributed applications are ported to .NET platform [Abram02, Buss, Silver].

- Remote objects: objects that are created to serve remote requests.
 - In Remoting & RMI: remote requests = remote method invocations.
 - Objects created within the remote methods are considered remote. Otherwise, local.

IMPLEMENTATION

- How do we identify local and remote objects?



IMPLEMENTATION

- How do we measure GC efficiency?
 - We focus on the efficiency of ephemeral collection, but in the process to study the mature collection as well.

Total garbage collected (in ephemeral generation)

Total objects allocated (in ephemeral generation)

- Ephemeral GC efficiency is inversely proportional to total objects that are promoted to mature generation.
- We monitor and calculate the size of promoted objects during each ephemeral collection.

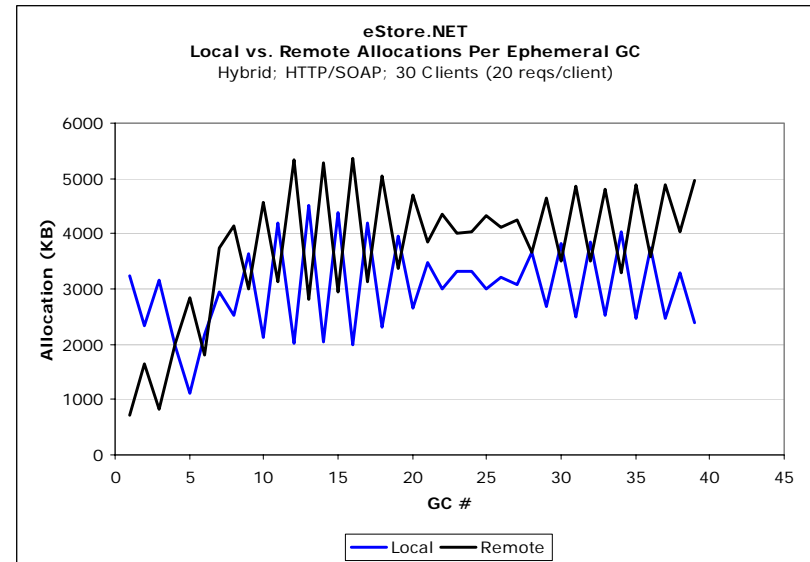
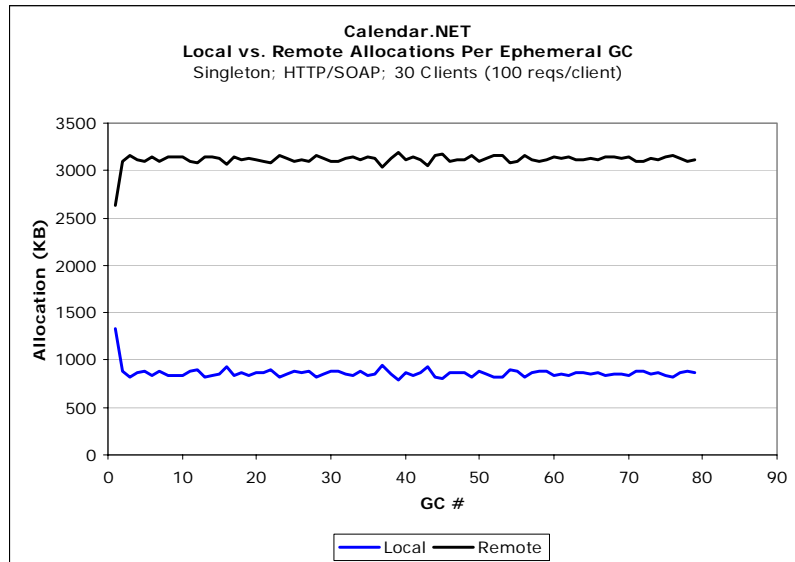
BENCHMARK

- Calendar.NET (1137 LOC).
 - 3-tier calendar application: client, server, XML-based database.
- eStore.NET (1206 LOC).
 - 3-tier online store application: client, server, MySQL database.

	Calendar.NET	eStore.NET
Activation Mode	Singleton	Hybrid
Channel/Formatter	HTTP/Binary	HTTP/SOAP
Workload	30 clients (100 reqs/client)	30 clients (20 reqs/client)

RESULT

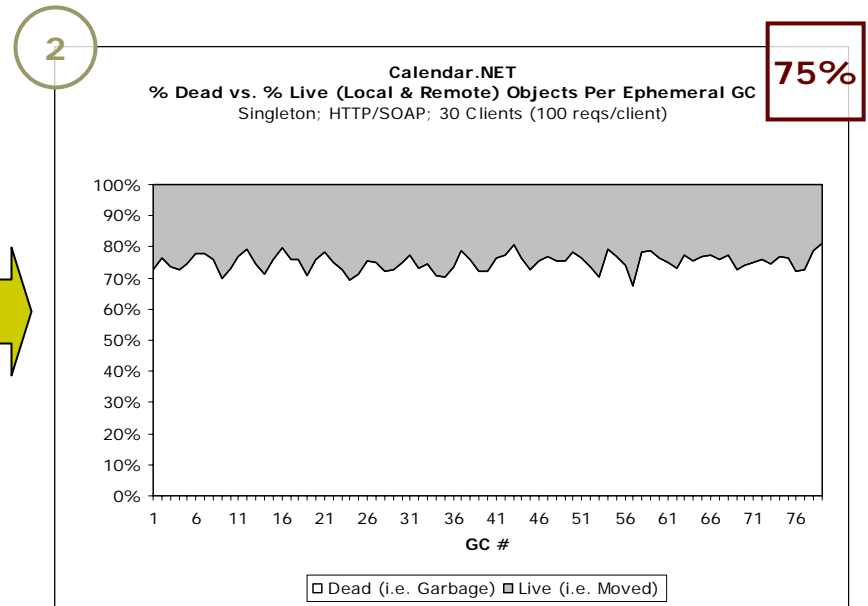
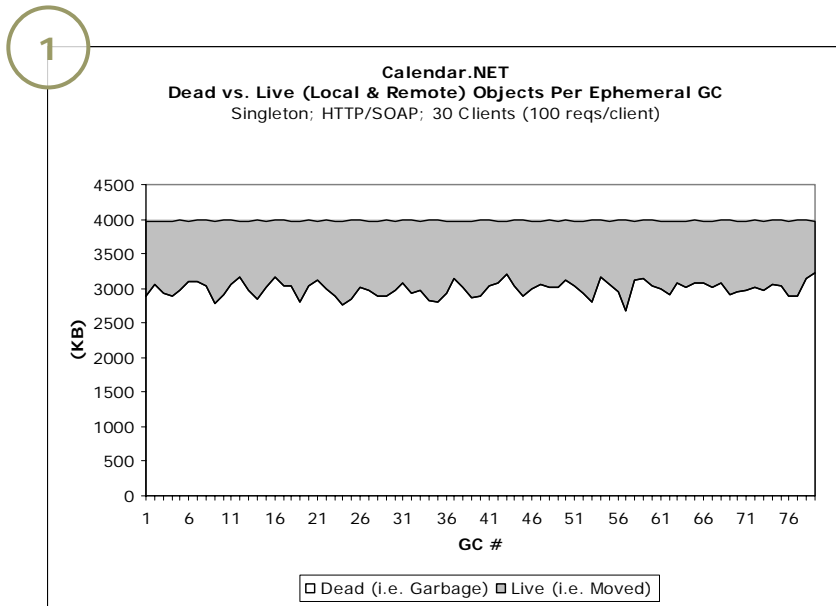
- Local objects are mostly created during program startup.
- Remote objects are created as the program continues to run.



RESULT

- Remote objects have different lifetime characteristics from local objects.
 - Remote objects tend to live longer.
- Generational GC does not perform efficiently.
 - The inefficiency is mainly affected by the remote objects.
 - Opportunities for improvement.

RESULT: Calendar.NET

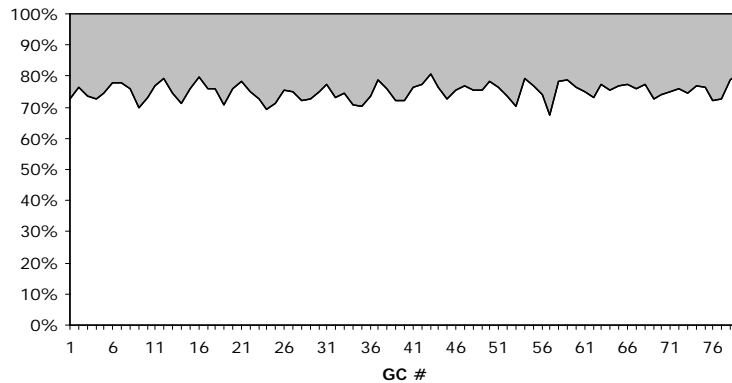


RESULT: Calendar.NET

2

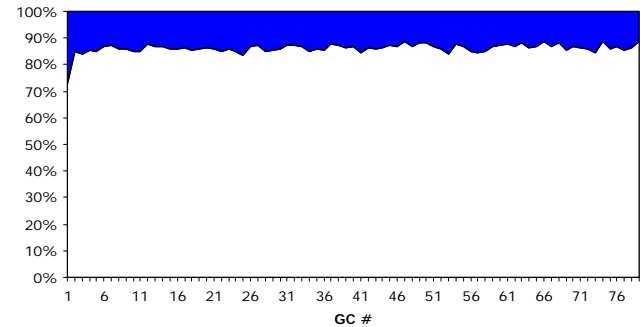
Calendar.NET
% Dead vs. % Live (Local & Remote) Objects Per Ephemeral GC
Singleton; HTTP/SOAP; 30 Clients (100 reqs/client)

75%



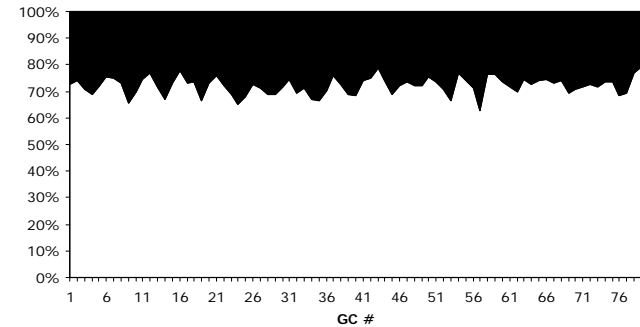
Calendar.NET
% Dead vs. % Live Local Objects Per Ephemeral GC
Singleton; HTTP/SOAP; 30 Clients (100 reqs/client)

86%

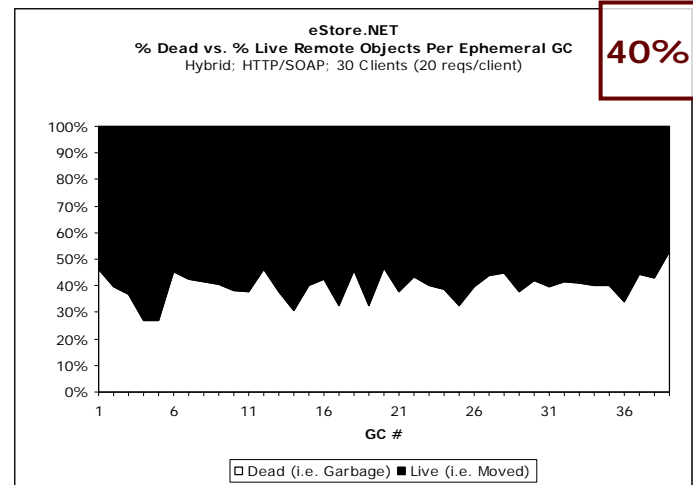
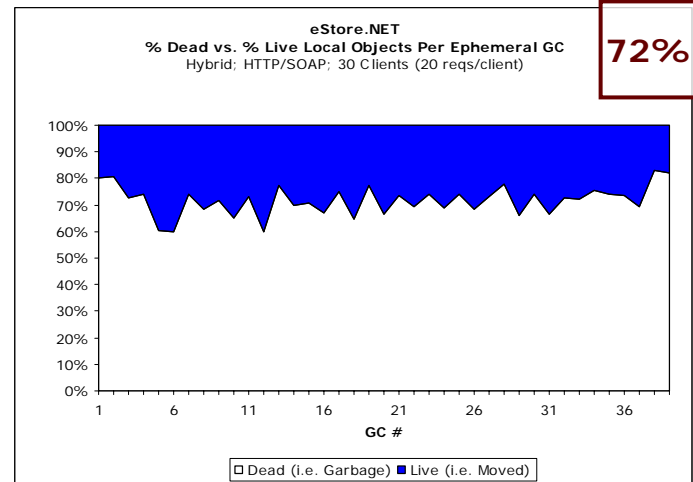
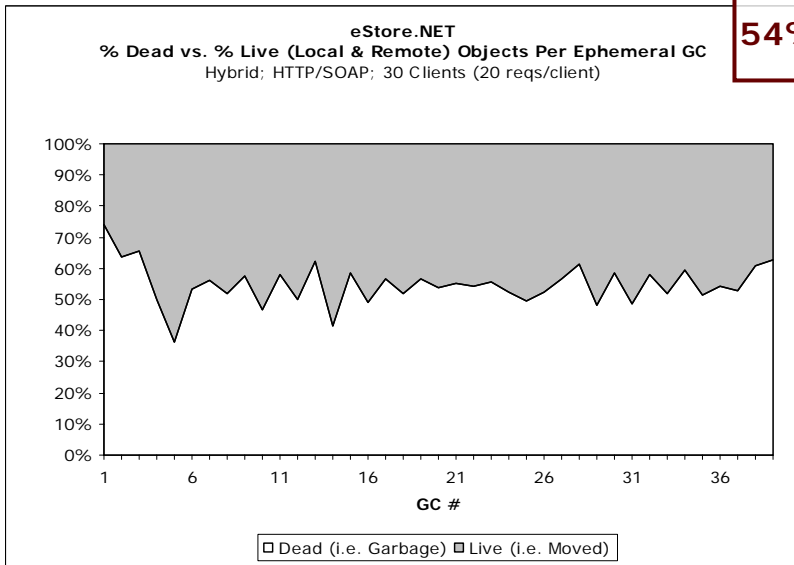


Calendar.NET
% Dead vs. % Live Remote Objects Per Ephemeral GC
Singleton; HTTP/SOAP; 30 Clients (100 reqs/client)

71%



RESULT: eStore.NET



DISCUSSION

- We might gain efficiency by separating the remote from the local objects.
 - Manage remote objects in a separate ephemeral heap.
 - We experiment with a modified generational scheme (simulation) that segregates local and remote objects.
- We use Calendar.NET as a preliminary study.
 - GC granularity (average): 2 KB.
- By segregating local and remote objects into two different ephemeral generations:
 - Average ephemeral GC efficiency for local objects increases from 72% to 84%.
 - Average ephemeral GC efficiency for remote objects increases from 69% to 82%.
 - Space traversal by mature collection is reduced by 38%.

CONCLUSION

- **RQ#1: Is the lifespan of remote objects different from local objects?**
 - Remote objects have different lifetime characteristics.
 - Remote objects tend to live longer compared to local objects.
- **RQ#2: What are the effects of remote objects on generational GC?**
 - Generational GC does not perform efficiently.
 - The inefficiency is mainly affected by the remote objects.

FUTURE WORK

- Investigate to answer the next research questions:
 - **(Future) RQ#1:** Will most the copied/promoted remote objects survive the full collection?
 - Are the remote objects truly long-lived or semi-long-lived (i.e. between short-lived and long-lived)?
 - **(Future) RQ#2:** What are the factors that determine the lifespan of remote objects?
 - Leasing time, workload, heap size?
- Implement a separate ephemeral generation for remote objects in SSCLI.
 - Performance comparison and analysis.

REFERENCE

- [Abram02] D. Abramson, G. Watson, and L. Dung. Guard: A tool for migrating scientific applications to the .NET framework. In *Proceedings of International Conference on Computational Science (ICCS 2002)*, pages 834-843, Amsterdam, The Netherlands, April 2002.
- [Buss] LTD. Buss. <http://www.buss.co.uk/buss/dnet.htm>.
- [Mason] R. Mason and W. Kelly. Peer-to-peer Cycle Sharing via .NET Remoting. <http://ausweb.scu.edu.au/aw03/papers/mason/>.
- [McLean03] S. McLean, J. Naftel, and K. Williams. *Microsoft .NET Remoting*. Microsoft Press, 2003.
- [Nutt05] G. Nutt. *Distributed Virtual Machines: Inside the Rotor CLI*. Addison Wesley, 2005.
- [Ram02] I. Rammer. *Advanced .NET Remoting*. Apress, 2002.
- [Silver] Silverfrost. http://www.silverfrost.com/21/ftn95/buss_casestudy.asp.
- [Stutz03] D. Stutz, T. Neward, and G. Shilling. *Shared Source CLI Essentials*. O'Reilly and Associates, 2003.