TOWARDS PERFORMANCE ANALYSIS OF NON-BLOCKING CONCURRENT DATA STRUCTURES IN THE LINUX KERNEL

By

Pijush Chakraborty

Roll: 153050015

Guided By:

Prof. Sriram Srinivasan

Prof. Purushottam Kulkarni

Contents

Lock Free Data Structures

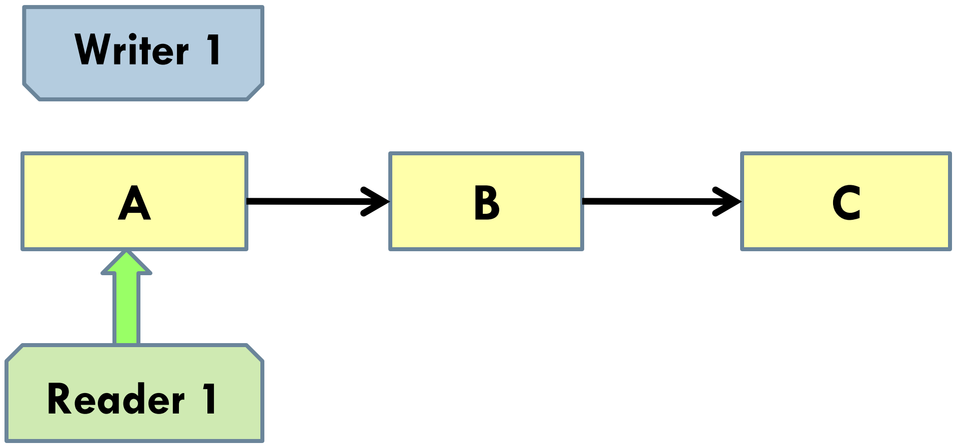
- Allows multiple threads to work in parallel on a particular data structure of interest.
- No wastage of CPU-cycles for waiting for a mutex lock.
- Other major problems to consider:
  - ABA problem
  - Memory Reclamation

Read Mostly Lock Free Data Structures

- Lock Free Data Structures are often compiled with memory reclamation methods.
- Research for Read Mostly Lock Free Data Structures
  - Read Copy Update Mechanism
  - Read Log Update Mechanism

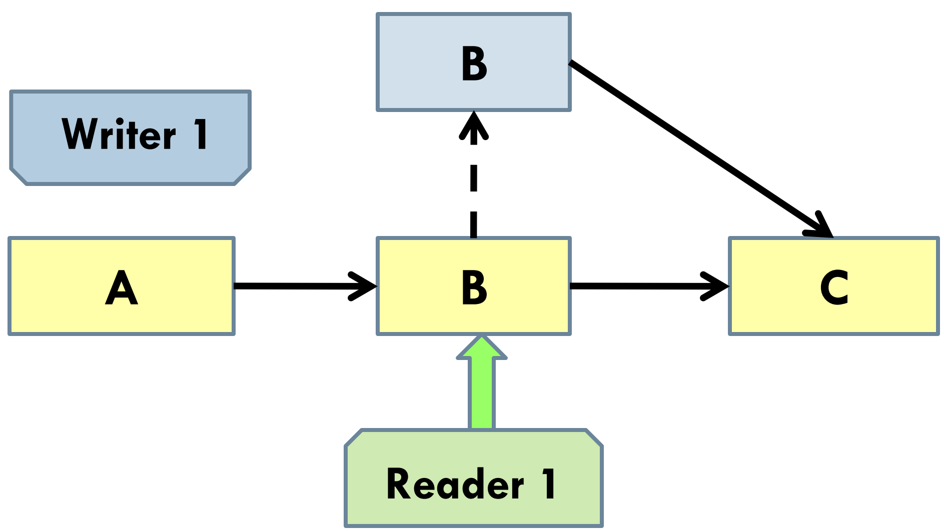
Read Copy Update

- Never Block Readers
- Writer synchronization left to the developers
- The Mechanism:
  - Read:



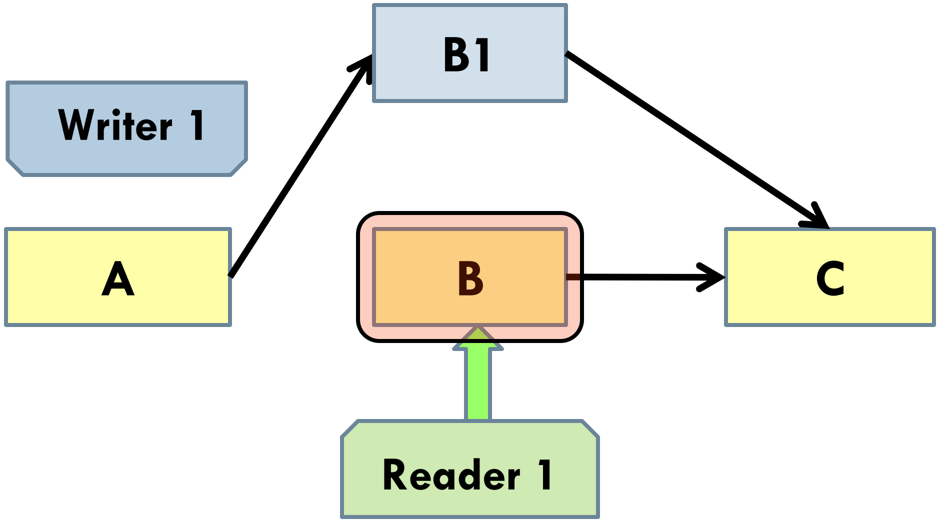
Read Copy Update

- Never Block Readers
- Writer synchronization left to the developers
- The Mechanism:
  - Copy:



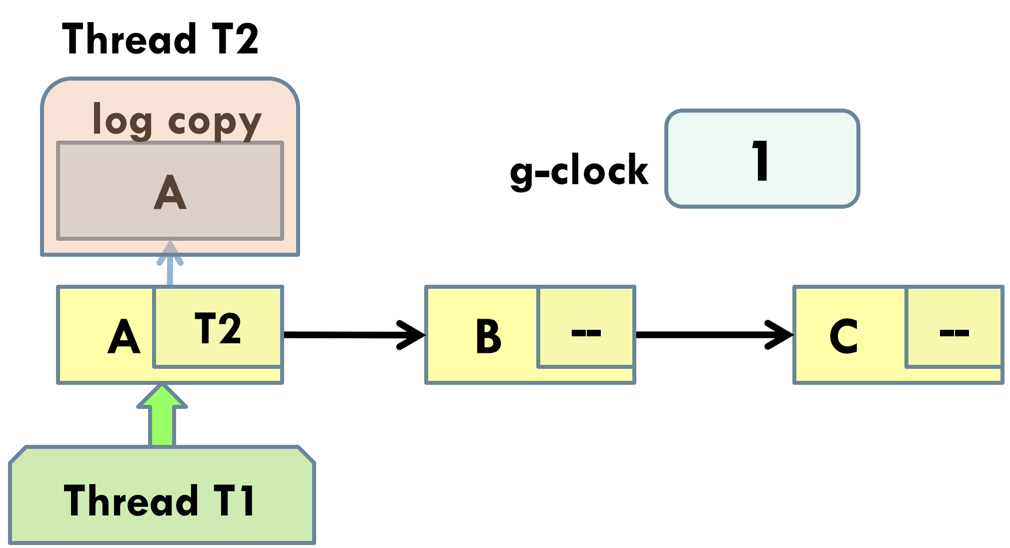
Read Copy Update

- Never Block Readers
- Writer synchronization left to the developers
- The Mechanism:
  - Update:



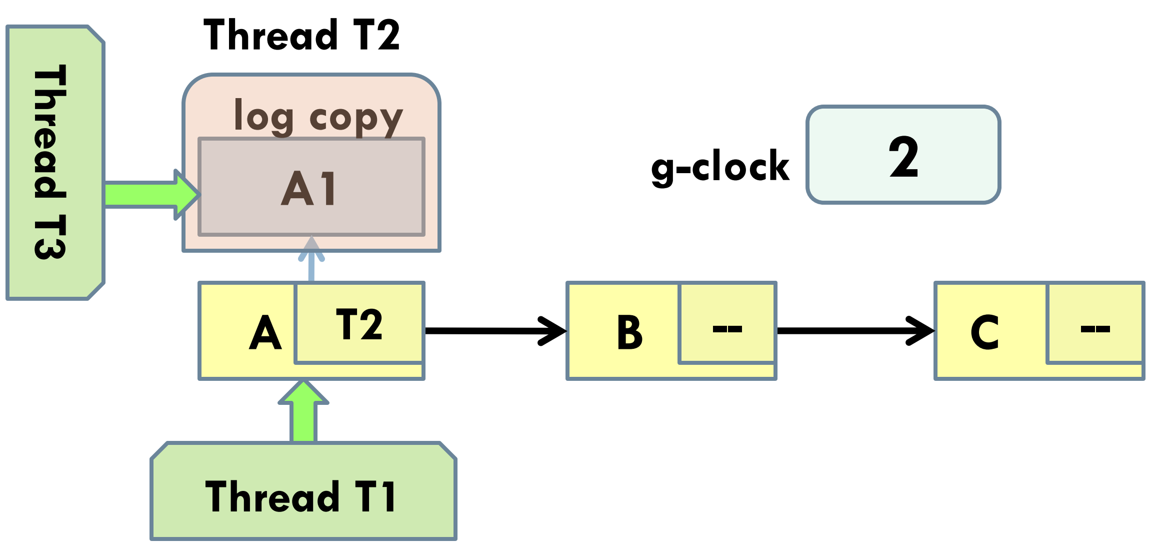
Read Log Update

- Provides a per thread log for concurrent writers
- Easy to use writer synchronization
- The Mechanism:
  - Read-Log:



Read Log Update

- Provides a per thread log for concurrent writers
- Easy to use writer synchronization
- The Mechanism:
  - Update:



Contents

Problem Statement

- Compare existing Read Mostly Lock Free Mechanisms.
- Answers to find:
  - Should writers be helped to improve overall performance?
  - Can we improve existing list based semantics in use to improve cache utilization?

Contents

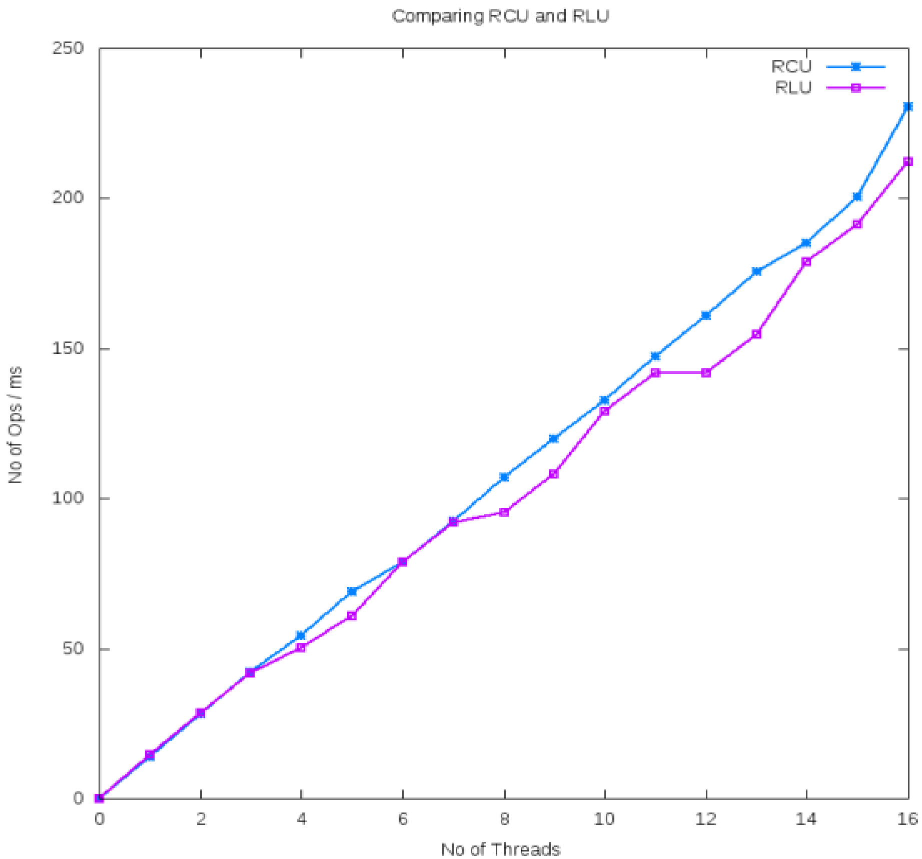
- Introduction
- Problem Statement
- Establishing Claims
- Conclusion and Stage-2 Plans

Verifying Read-Log-Update Claims

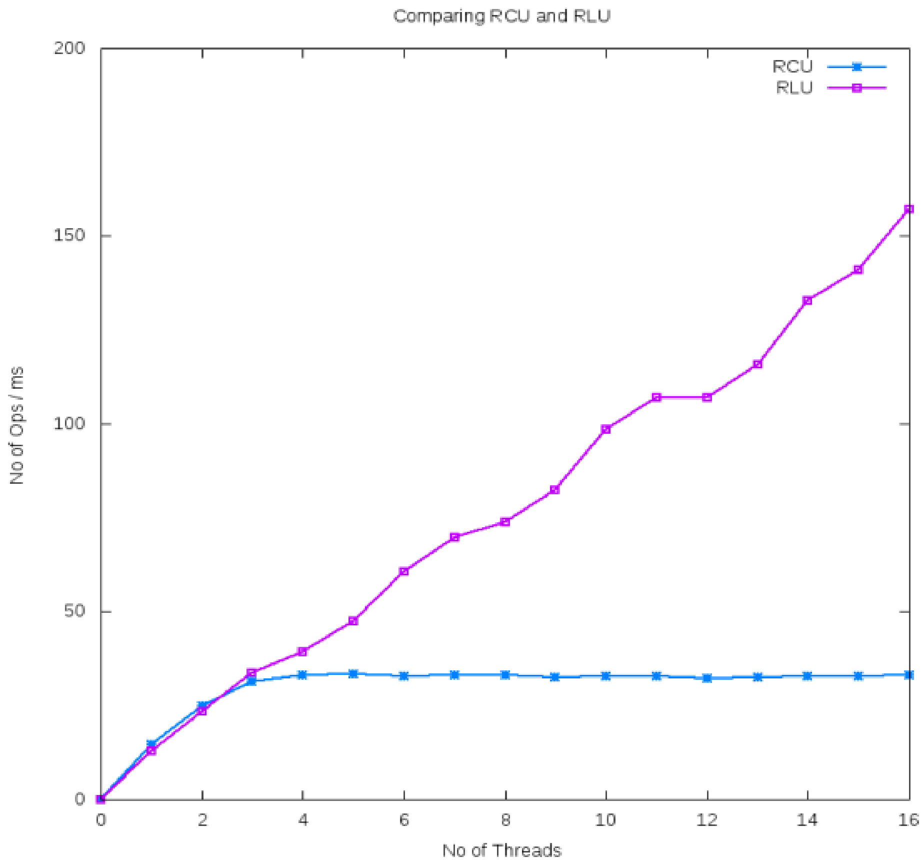
- Questions:
  - Is RLU better than RCU?
  - Does Node-Size matter in the comparisons?

- Experiments for Comparing RCU and RLU
  - System: 16 core blade server(Intel Xeon) supporting 16 Hardware threads.
  - Compared using benchmark used by RLU authors

Comparing Linked Lists



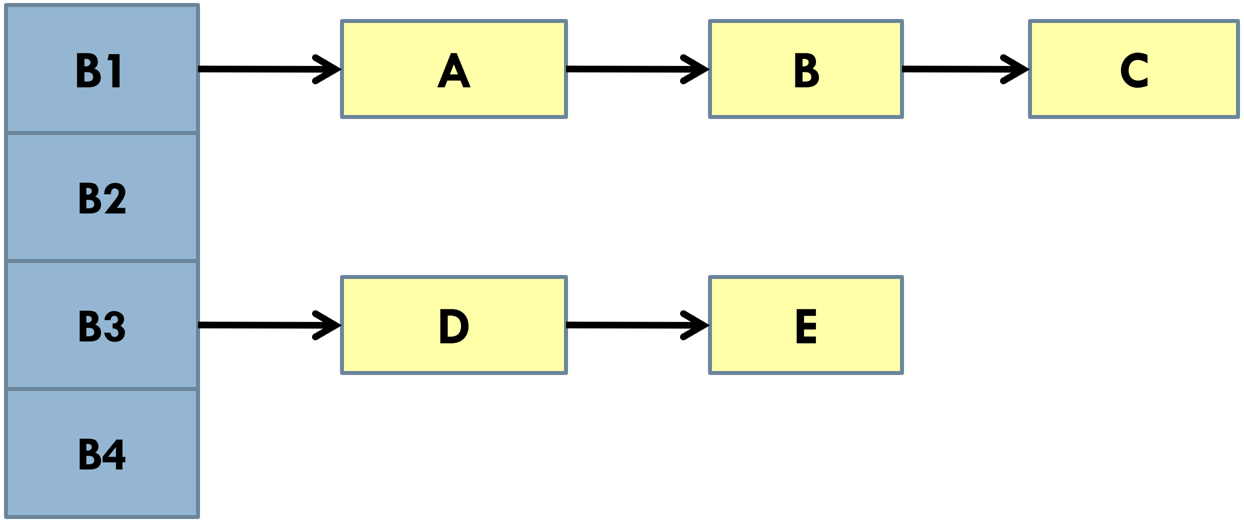
Comparison with no updates



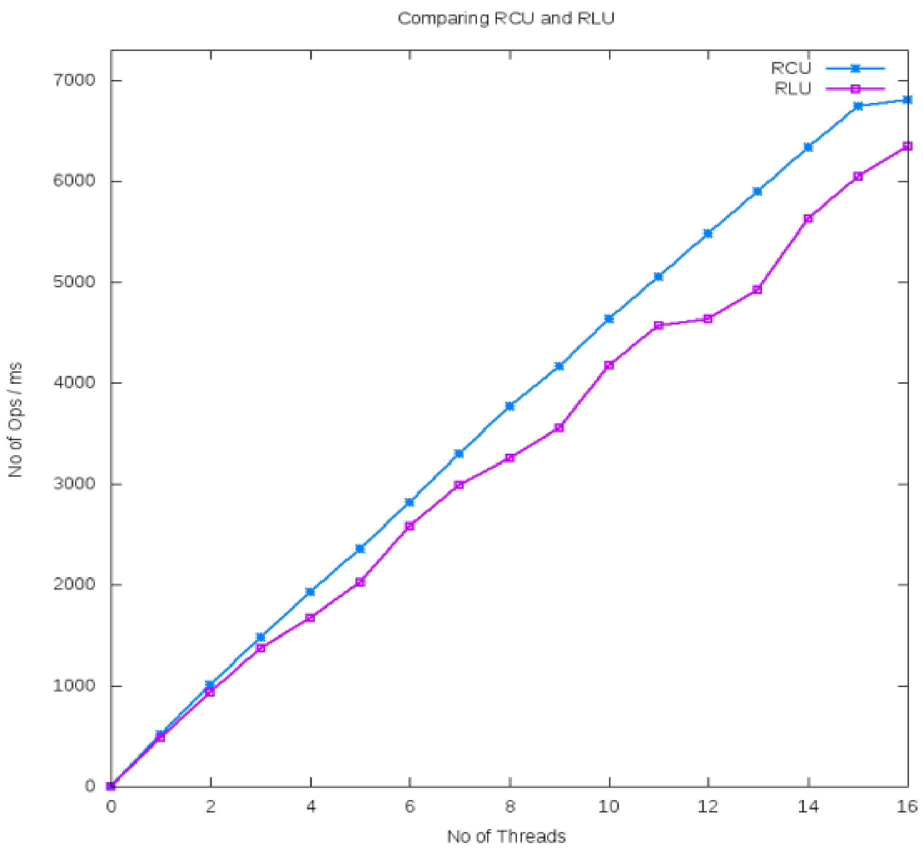
Comparison with 40% updates

Comparing Hash Lists

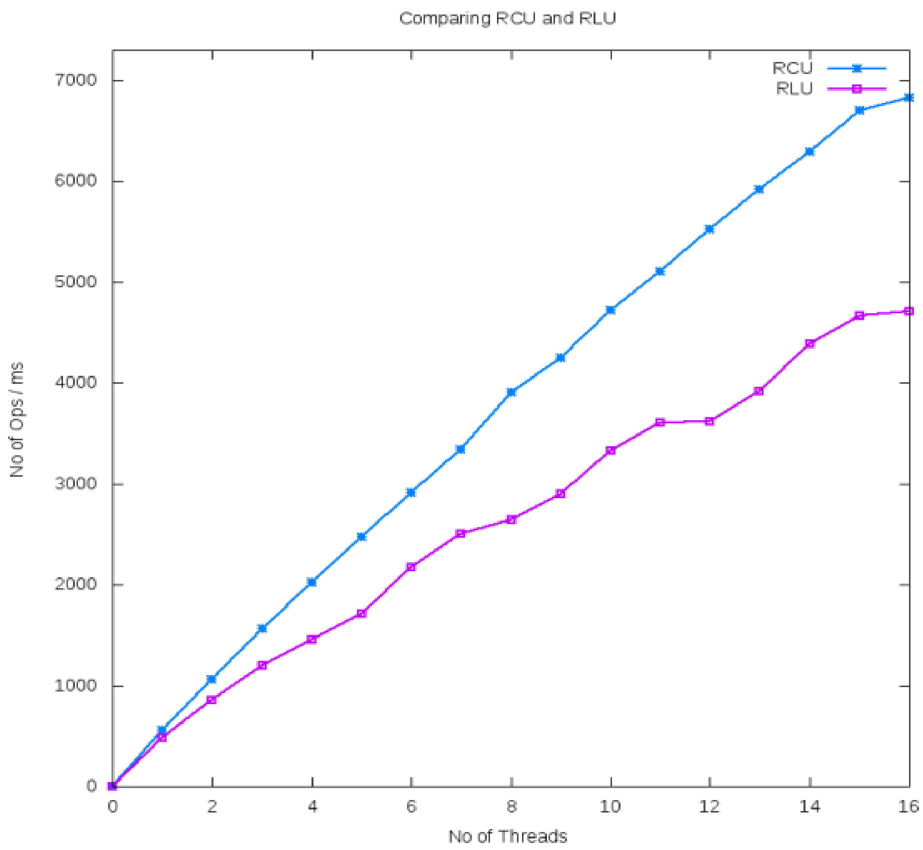
- Hash Lists protected by RCU and RLU in use



Comparing Hash Lists

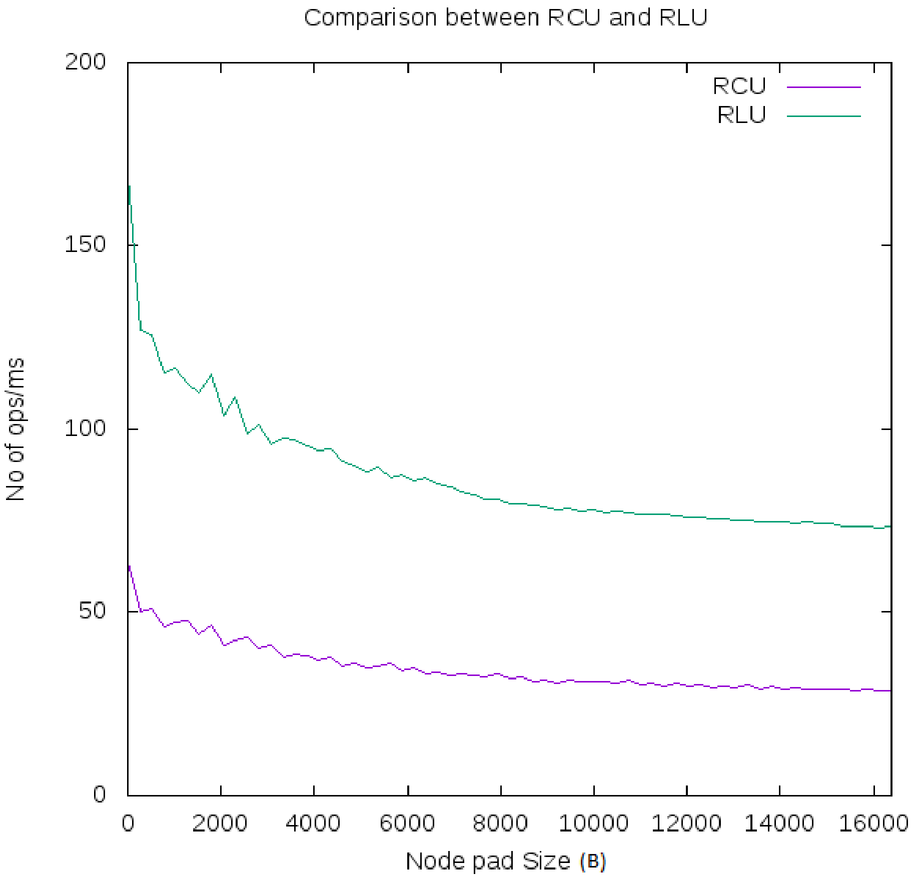


Comparison with no updates

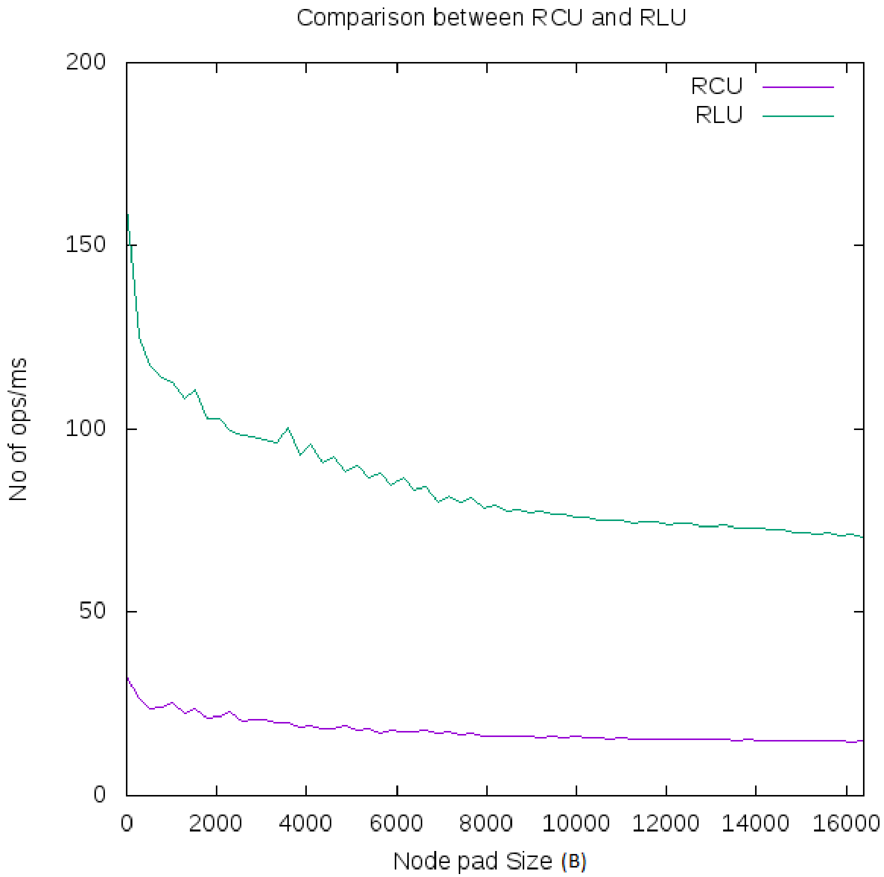


Comparison with 40% updates

Does Node Size Matter?



Linked List with 20% updates



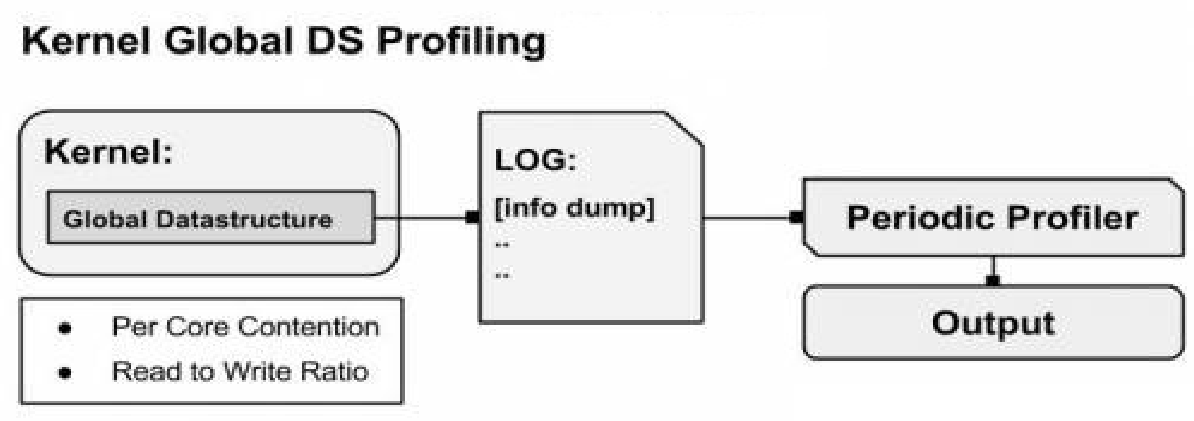
Linked List with 40% updates

Kernel Side of the Story

- Question:
  - Do the writers need any optimization?
- Understand the usage behavior of the currently used RCU protected data structures
  - Read-Write Ratio
  - Write Contention

Experiments:

- System: 4 core Intel i5 processor(1.2GHz), 8GB Ram
- Design:



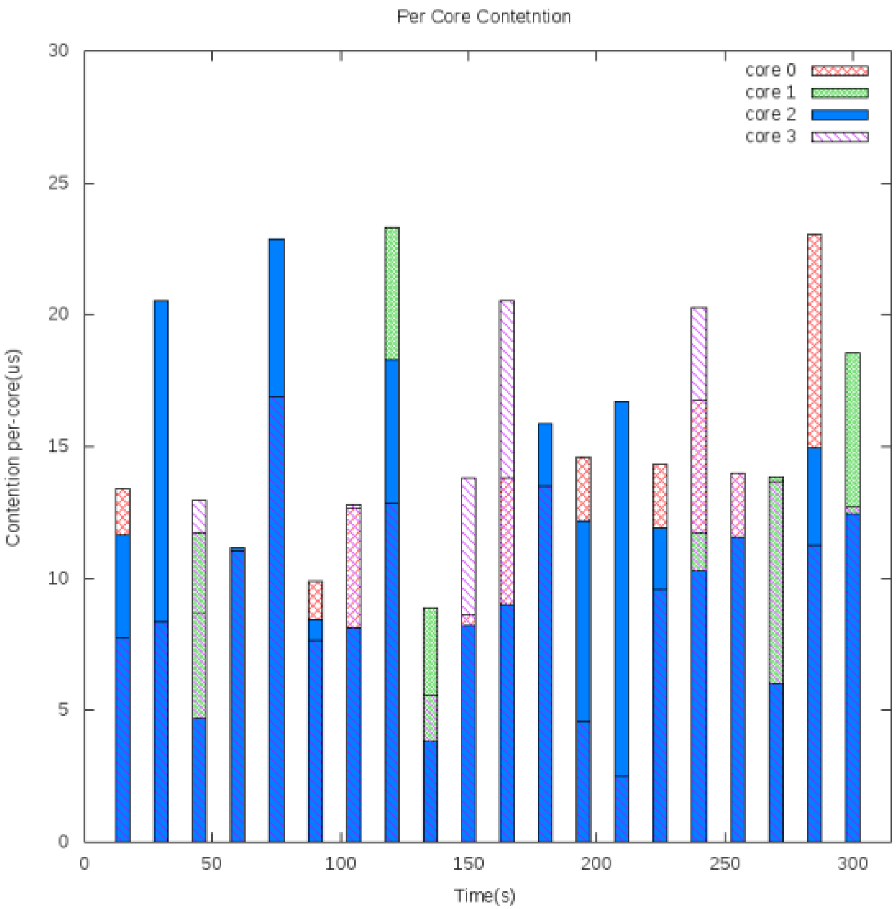
- Workload:
  - Web Server Workload
  - Process List Stress Workload

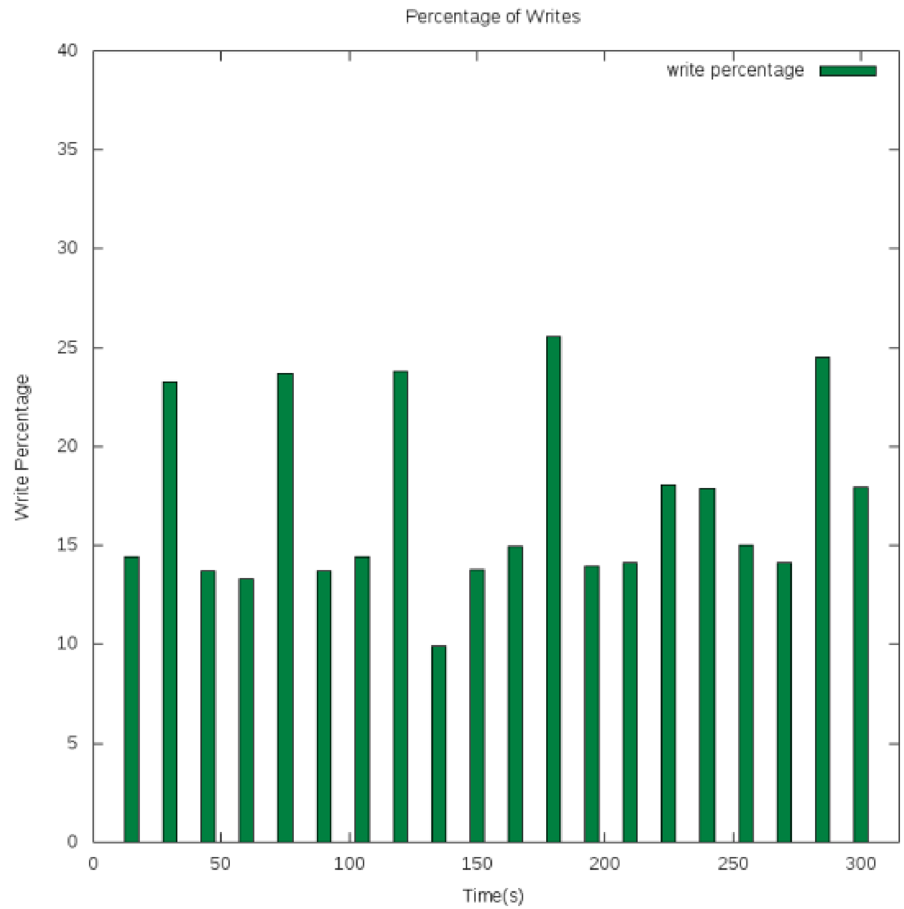
Profiling Data Structures

- List to Track: epoll file descriptor list
- Server in use:
  - Nginx local server that uses epoll to serve multiple client requests
- Client Statistics:
  - 1000 parallel clients created every 5 seconds
  - Each client making 10000 requests in parallel to localhost.

Profiling Data Structures

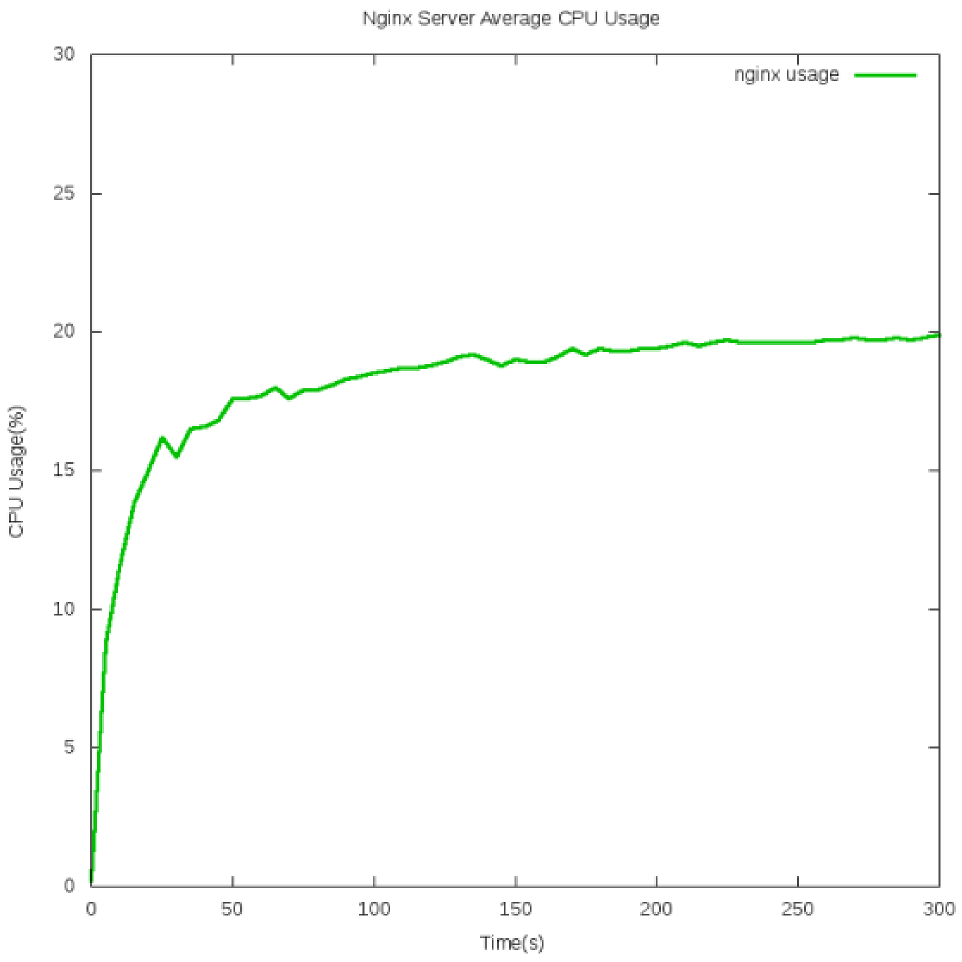
- Workload Load: Web Server Workload





Profiling Data Structures

- Workload Load: Web Server Workload

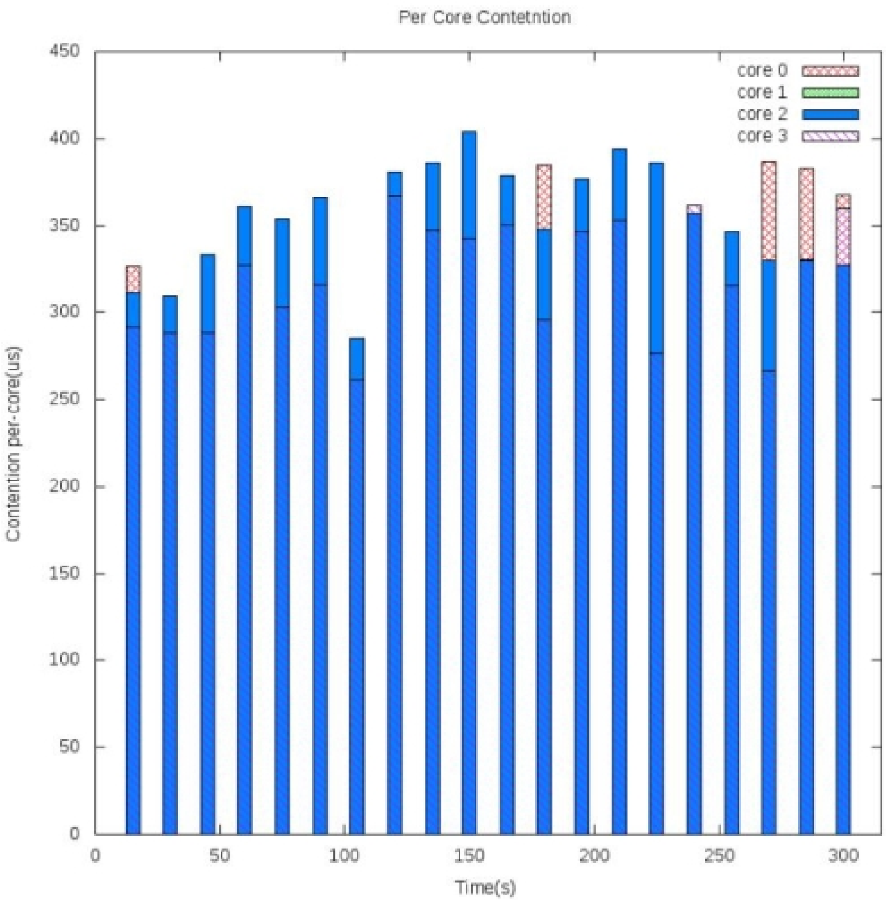


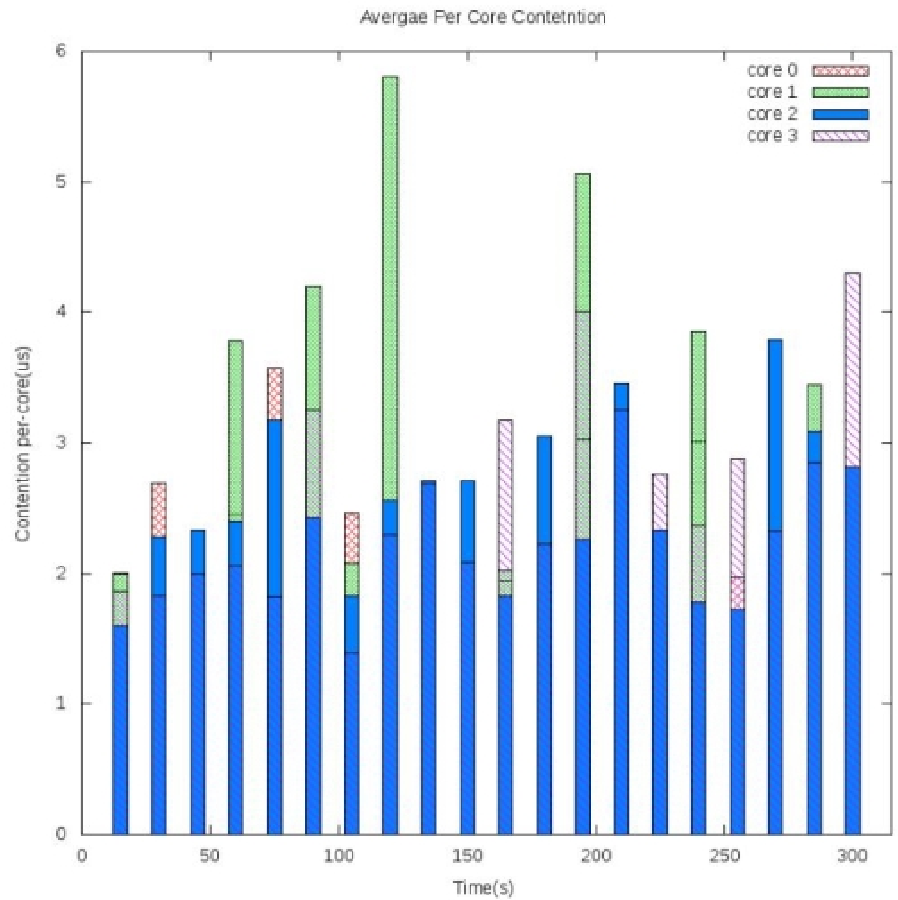
Profiling Data Structures

- List to Track: List of all processes
- Synthetic Workload:
  - 250 parallel threads
  - Each thread creating 500 empty threads every 5 seconds

Profiling Data Structures

- Workload Load: Synthetic Stress Workload





Helping the Readers

- Questions:
  - Can we provide a cache friendly way of traversal?
  - Should the list based semantics be changed?
- Kernel data structures uses list based semantics
- But, arrays provide better cache utilization than lists

Observations

- Are the lists ordered?

Name of List Head	Insertion Behavior	Deletion Behavior
List of process with same thread-id	Inserts at head	Delete anywhere
List for epoll file descriptors	Inserts at tail	Delete anywhere
List of all running processes	Inserts at head	Delete anywhere

Table 1: Usage Behavior Observations

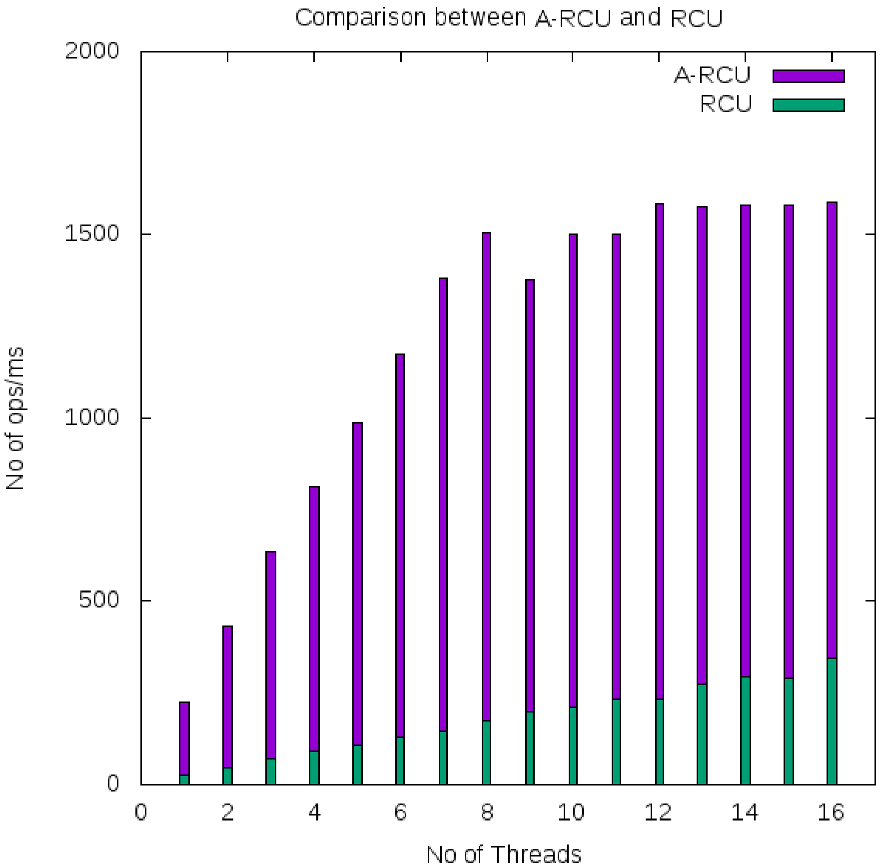
Usage Statistics	Linked List	Hash List
No of such RCU protected data structures in the Kernel	236	71
Inserts at list tail	109	–
Inserts at head	127	71

Table 2: Usage Statistics Assumptions

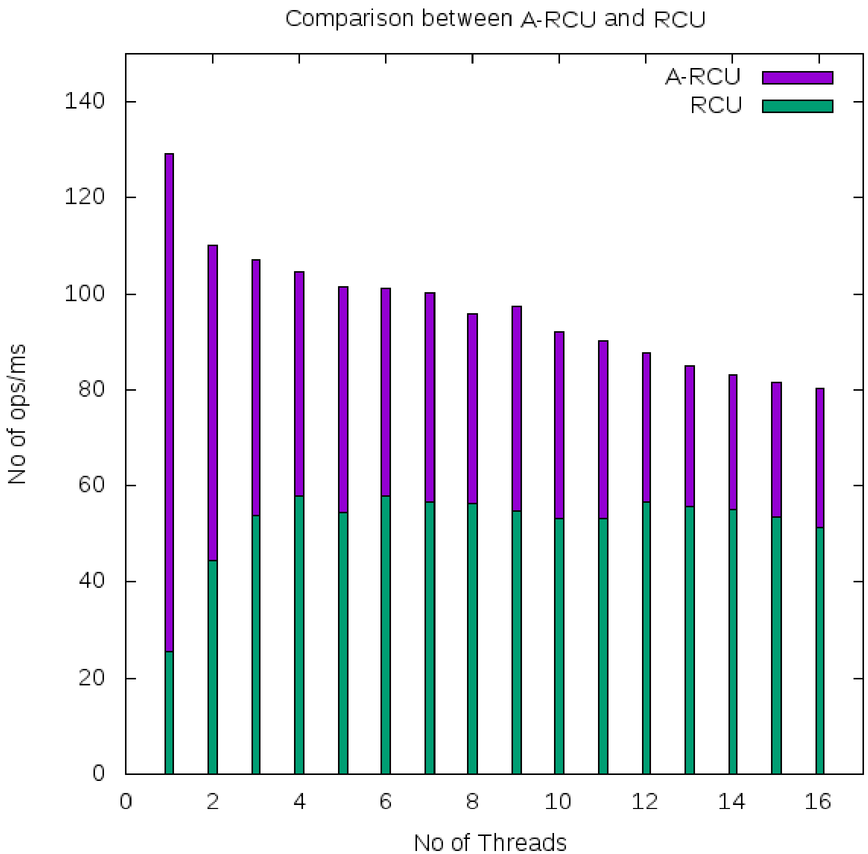
RCU Protected Arrays

- Can RCU protected Arrays prove better?
- Experiment:
  - Comparison using RCU array(A-RCU) and RCU protected linked list
  - Benchmark used by RLU authors for comparing RCU and RLU
  - System: 16 core Xeon Blade Server processor supporting 16 hardware threads.

RCU Protected Arrays

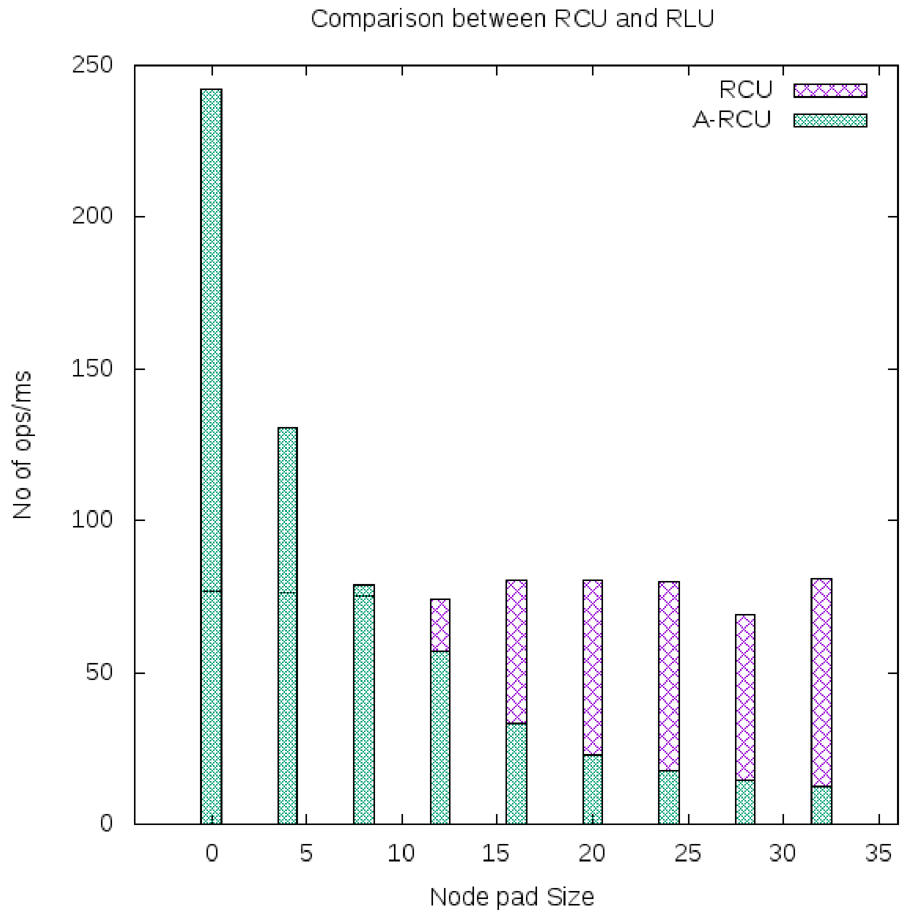


Comparison with no updates



Comparison with 40% updates

Does Node Size Matter?



Comparison with 20% updates

Conclusion

- Studied the behavioral aspects of RCU Usage in the Linux Kernel.
- Need to look at Reader Perspective in Stage-2.
- Design a lock free mechanism from that supports:
  - More cache utilization
  - Supports larger node size
- Change a particular Kernel subsystem to use the array based design and compare with existing model.

References

- [1] Timothy L. Harris. A pragmatic implementation of non-blocking linked-lists. Proceedings of the 15th International Conference on Distributed Computing, pages 300–314, October 2001.
- [2] Alexander Matveev, Nir Shavit, Pascal Felber, and Patrick Marlier. Read-log-update: a lightweight synchronization mechanism for concurrent programming. 2015.
- [3] Paul E. McKenney. Read-copy update (RCU) usage in Linux kernel. Available: http://www.rdrop.com/users/paulmck/RCU/linuxusage/rculocktab.html, October 2006.
- [4] Paul E. McKenney. What is rcu: https://lwn.net/articles/262464/. 2007.
- [5] Paul E. McKenney. Rcu linux usage log: http://www.rdrop.com/ paulmck/rcu/linuxusage/linux- 4.3.rcua. 2015.
- [6] Paul E. McKenney. Read-mostly research in 2015: https://lwn.net/articles/667593. 2015.
- [7] Paul E. McKenney. Some more details on read-log-update: https://lwn.net/articles/667720. 2015.
- [8] Paul E. McKenney and John D. Slingwine. Read-copy update: Using execution history to solve concurrency problems. In Parallel and Distributed Computing and Systems, pages 509–518, Las Vegas, NV, October 1998.
- [9] Maged M. Michael. Hazard pointers: Safe memory reclamation for lock-free objects. IEEE Transactions on Parallel and Distributed Systems, 15(6):491–504, June 2004.
- [10] John D. Valois. Lock-free linked lists using compare-and-swap. , Proceedings of the 14th annual ACM symposium on Principles of distributed computing, pages 214–222, August 1995.