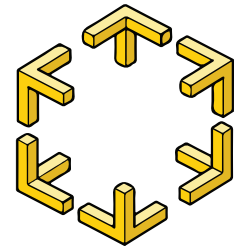


Lock-Free Search Data Structures: Throughput Modeling with Poisson Processes

Aras Atalar, Paul Renaud-Goud, Philippas Tsigs

Chalmers University of Technology

Concurrent Data Structures

- Concurrency:
  - Concurrency is the overlapped executions of processes
  - Interleaving of steps of processes
  - Synchronization to avoid interleavings that lead to unintended states
- Lock-based concurrent data structures:
  - Rely on mutual exclusion to work in isolation
  - Limitations: deadlocks, priority inversion and programming flexibility (difficult to compose)
- Lock-free concurrent data structures:
  - Guarantee system-wide progress
  - Employ optimistic conflict control
  - Limitations: harder to design and implement

Related Work

- Theoretical results:
  - Focus on retry loop conflicts and hardware conflicts (exist when operations overlap in time and memory location)
  - Amortized analyses parameterized with a measure of contention
  - Model asynchrony with adversarial scheduler
  - Target worst-case execution times
- Empirical results:
  - Compare the performance of different implementations
  - Help to grasp the hardware-software interaction
- In this work:
  - Study the throughput performance of lock-free search data structure
  - Propose analytical tools that provide estimations that is close to what we observe in practice

Lock-free Search Data Structures

- Search data structure is a collection of $\langle key, value \rangle$ pairs which are stored in an organized way to allow efficient search, delete and insert operations (*e.g.* Hash table, binary tree, skip list, linked list)
- Formed of basic blocks (Nodes)
- Accessed with *Read* and *Modify (CAS)* events
- Retry loop conflicts are very improbable ($Nodes \gg Threads$)

Algorithm Skeleton

Output of the analysis: Data structure throughput ($\mathcal{T}$), *i.e.* number of successful data structure operations per unit of time

Procedure AbstractAlgorithm

```
1 while ! done do
2 |   key ← SelectKey(keyPMF);
3 |   operation ← SelectOperation(operationPMF);
4 |   result ← SearchDataStructure(key, operation);
```

- Key $\in [1, \mathit{Range}]$ and Operation $\in \{\mathit{Search}, \mathit{Insert}, \mathit{Delete}\}$
- Memoryless and stationary key and operation selection process

Algorithm Skeleton

Output of the analysis: Data structure throughput ($\mathcal{T}$), *i.e.* number of successful data structure operations per unit of time

Procedure AbstractAlgorithm

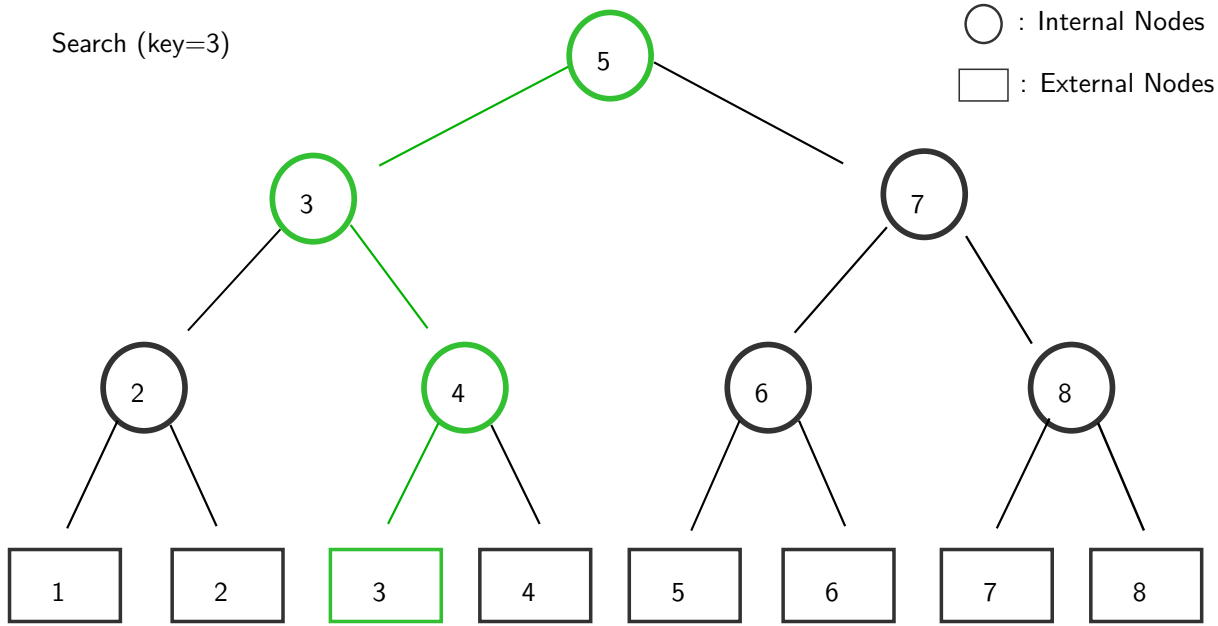
```
1 while ! done do
2 |   key ← SelectKey(keyPMF);
3 |   operation ← SelectOperation(operationPMF);
4 |   result ← SearchDataStructure(key, operation);
```

- Key $\in [1, Range]$ and Operation $\in \{Search, Insert, Delete\}$
- Memoryless and stationary key and operation selection process

- **Inputs of the analysis:**
  - *Platform parameters*: Data and TLB cache hit latencies, *CAS* latency, in clock cycles
  - *Algorithm parameters*: *PMF*s for the key and operation selection, Key range ($\mathcal{R}$), Total number of threads (P), Expected latency of key and operation selection

Impacting Factors

- An operation triggers a number of node accesses (Which nodes?)
- Latency of the operation: sum of the latencies of accesses



Impacting Factors

- Identify the factors that impact the latency of an access:
  - Capacity misses in data and TLB caches (both in sequential and concurrent executions)
  - Coherence misses (only in concurrent executions)
  - Execution time of *CAS* and stall time due to others' *CAS* (only in concurrent executions)

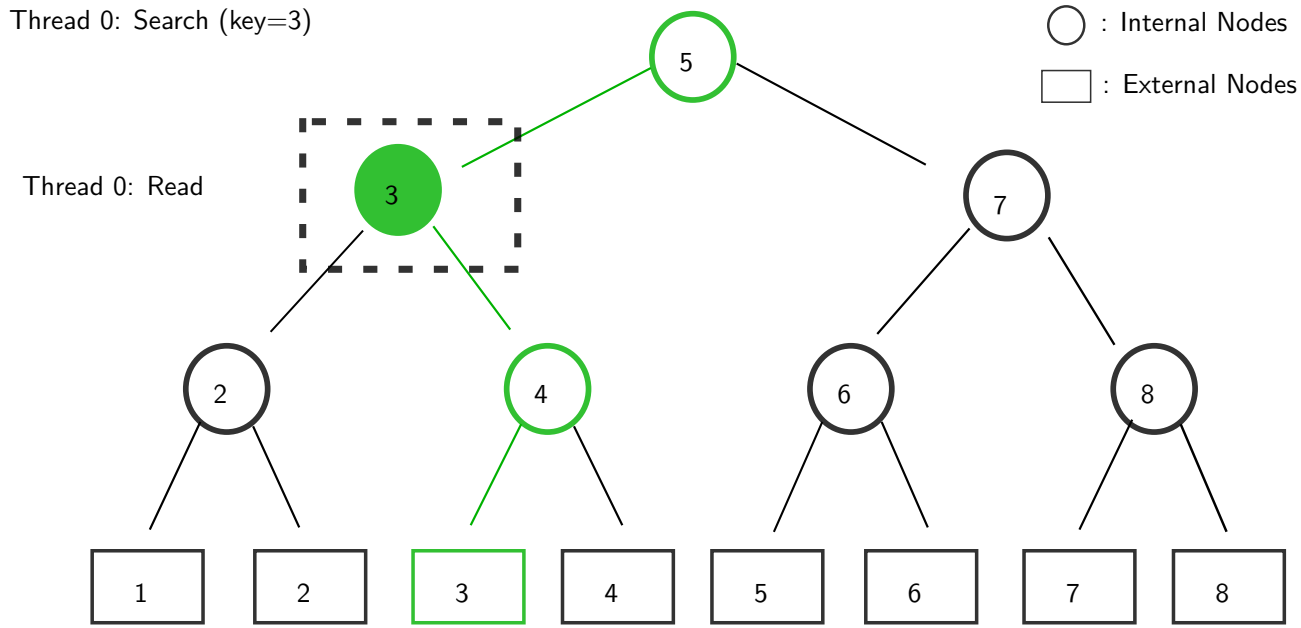
- Define access latency of node N_i:

$$Access_i = t^{cmp} + CAS_i^{exe} + CAS_i^{stall} + CAS_i^{reco} + \sum_{\ell} Hit_i^{cache_\ell} + \sum_{\ell} Hit_i^{tlb_\ell} \quad (1)$$

Impacting Factors

Over a sequence of operations: *Coherence Miss*

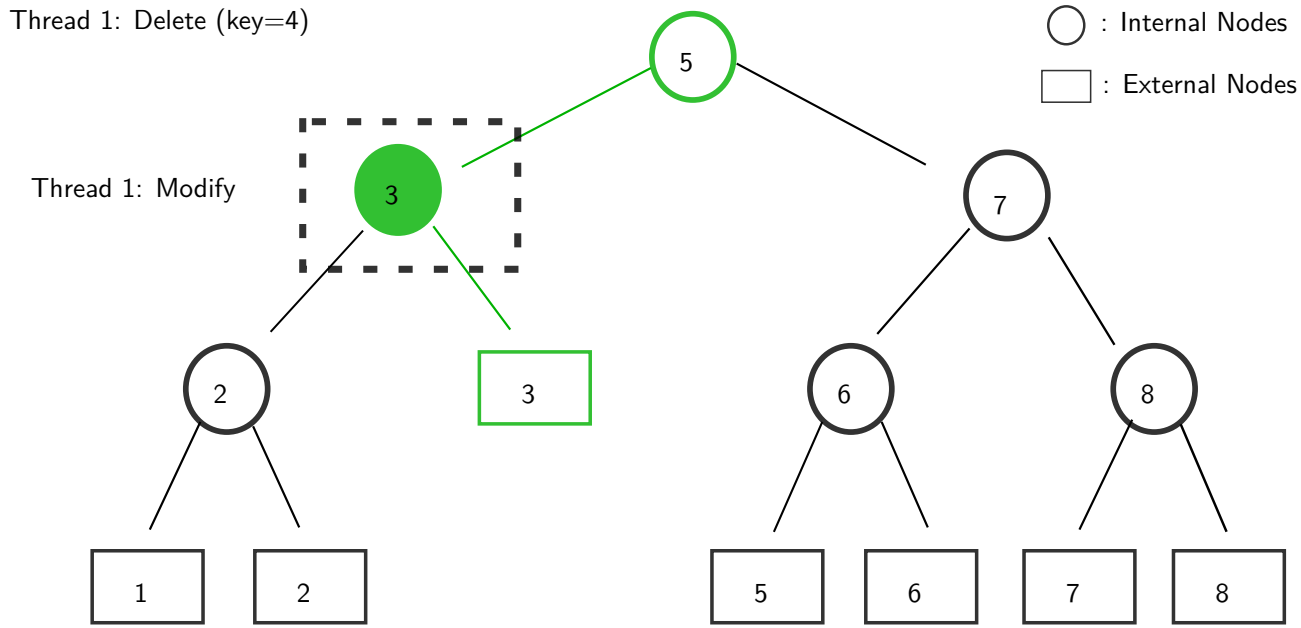
- Step 1: P_0 reads $IntNode_{key=3}$ (brings a valid copy to P_0)



Impacting Factors

Over a sequence of operations: *Coherence Miss*

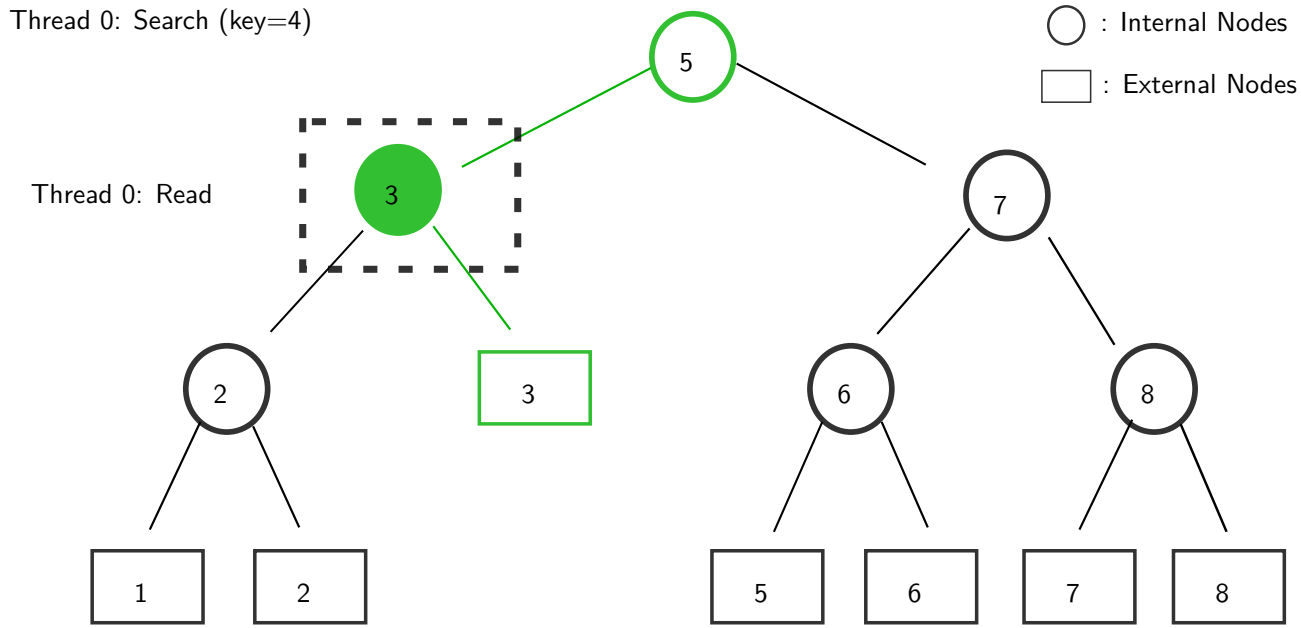
- Step 1: P_0 reads $IntNode_{key=3}$ (brings a valid copy to P_0)
- Step 2: P_1 modifies $IntNode_{key=3}$ (invalidates the copy of P_0)



Impacting Factors

Over a sequence of operations: *Coherence Miss*

- Step 1: P_0 reads $IntNode_{key=3}$ (brings a valid copy to P_0)
- Step 2: P_1 modifies $IntNode_{key=3}$ (invalidates the copy of P_0)
- Step 3: P_0 read $IntNode_{key=3}$ (coherence miss of P_0)



Approach

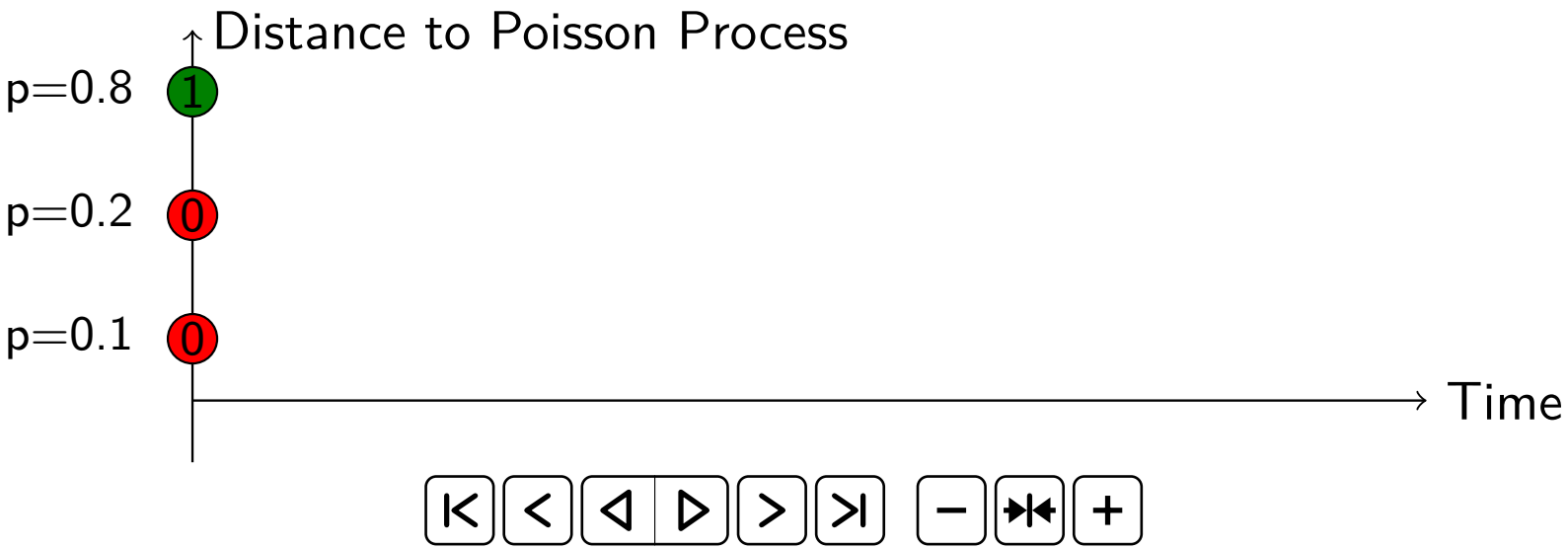
Observation: Latency of a node access depends on the interleaving of accesses

To estimate the latency of an access on node N_i:

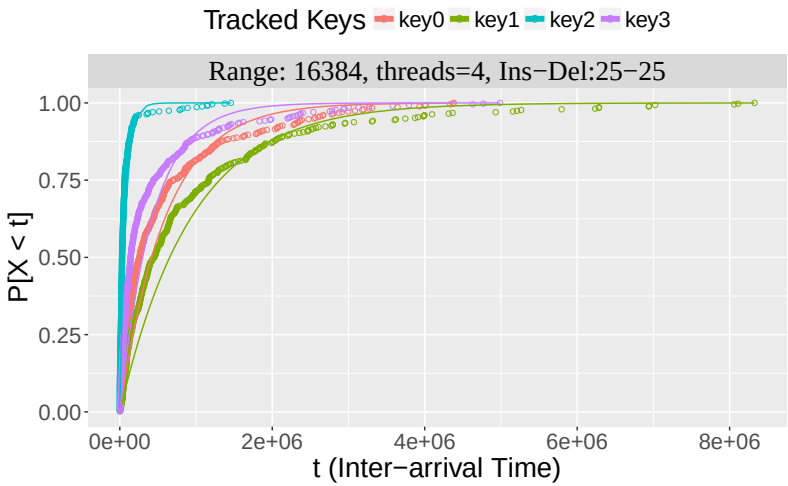
- Follow the sequence events (*Read* and *Modify* seperately) on N_i by *a thread*, when $N_i \in DS$
- Slice the execution into consecutive intervals, where an interval begins with a call to an operation by the thread
- Each interval potentially includes a Read event (resp. Modify) at N_i
- Think of a static structure: Stationary and memoryless access pattern $\rightsquigarrow$ Bernoulli Process

Approach

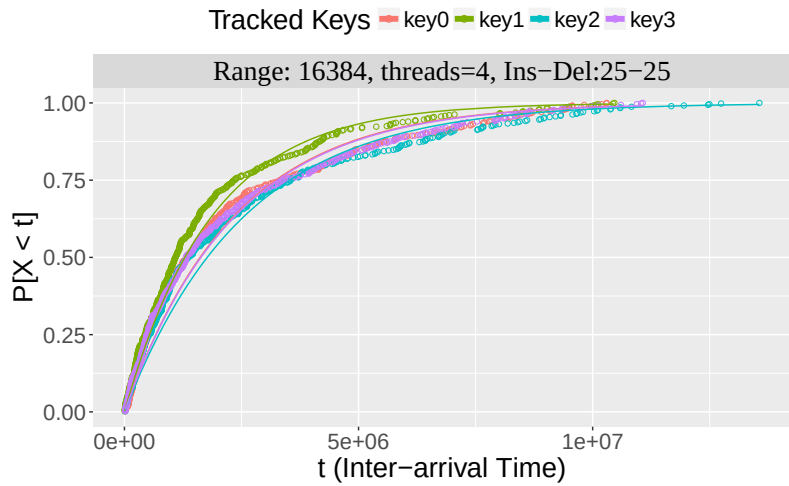
- Poisson Process approximation is well-conditioned if the success probability is small
- Dynamicity: DS change state with insertions and deletions
- Bernoulli trials with different success probabilities $\rightsquigarrow$ Poisson Process (if p_j are small)
- Key characteristic: set of nodes that are accessed in an operation is small in front of all nodes



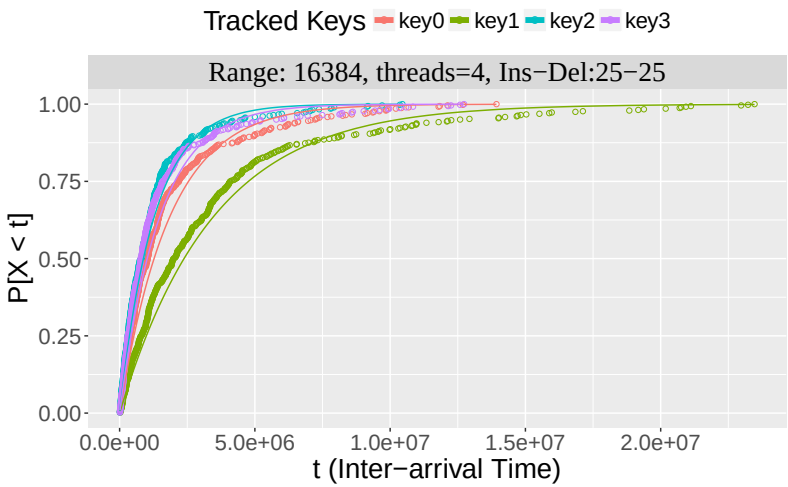
Statistical Test: Kolmogorov–Smirnov



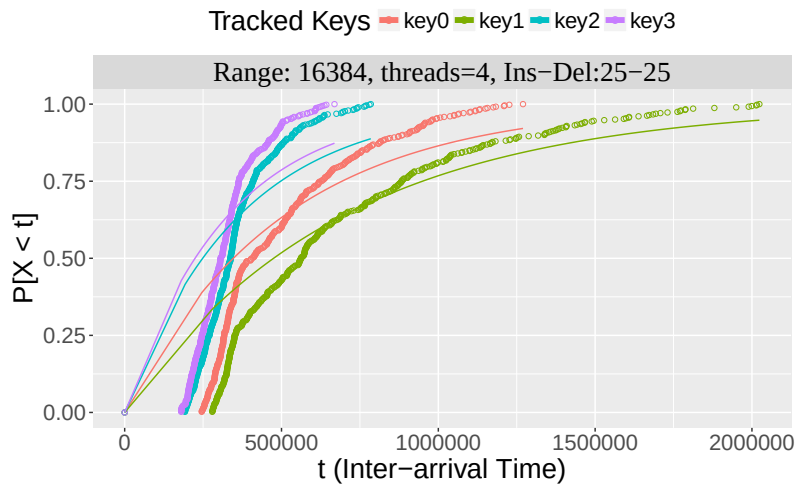
(a) Read Events for Skiplist



(b) Read Events for Hash Table

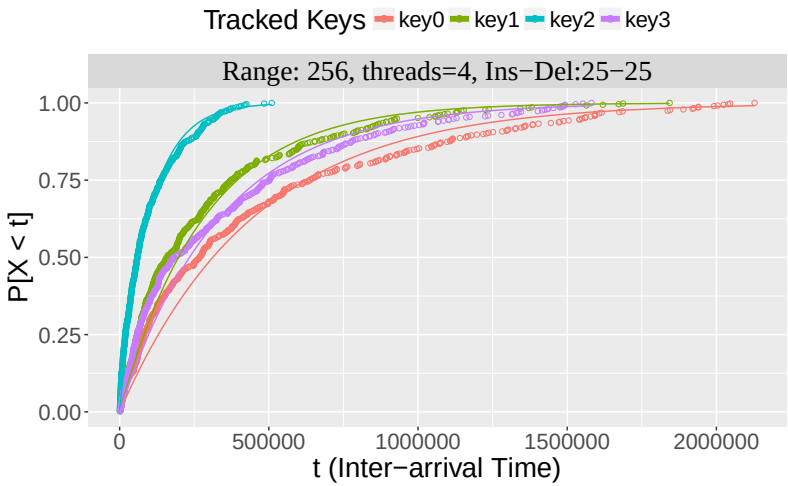


(c) Read Events for Binary Tree

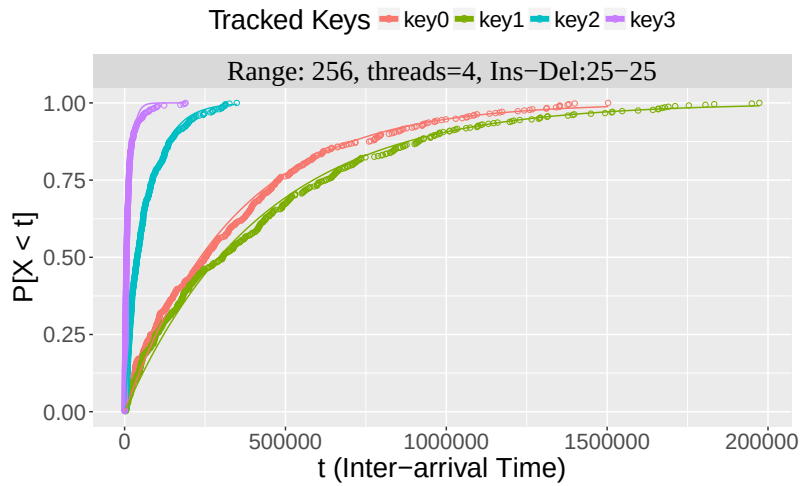


(d) Read Events for Linked List

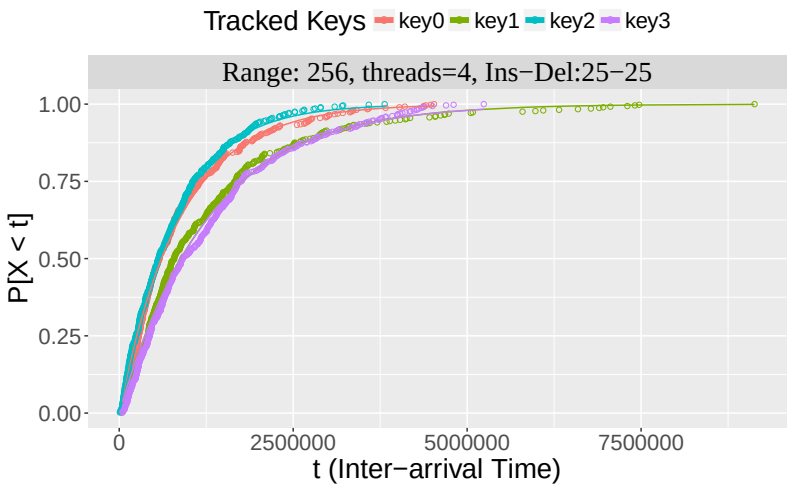
Statistical Test: Kolmogorov–Smirnov



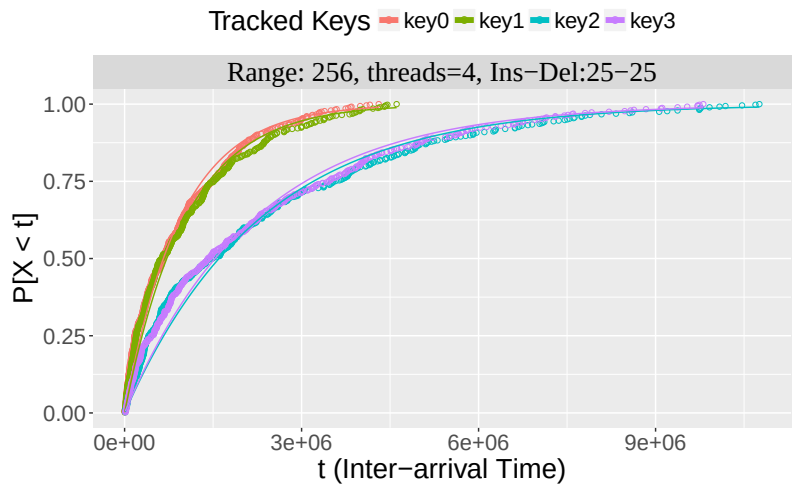
(a) CAS Events for Skip list



(b) CAS Events for Hash Table



(c) CAS Events for Binary Tree



(d) CAS Events for Linked List

Poisson Rates

- Extract the rate of events on N_i (by a thread) based on a random operation at a random time as a function of throughtput ($\mathcal{T}$):

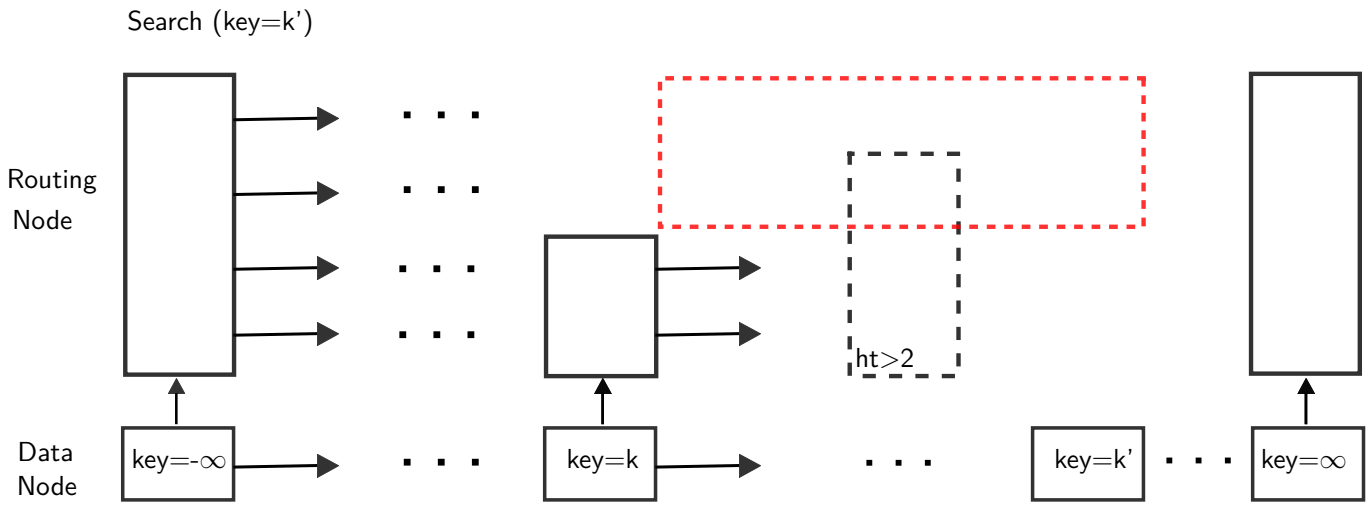
$$\forall e \in \{cas, read\} \ :$$

$$\lambda_i^e = \frac{\mathcal{T}}{P} \times \sum_{o \in \{ins, del, src\}} \sum_{k=1}^{\mathcal{R}} \mathbb{P}\left[Op = op_k^o\right] \times \mathbb{P}\left[op_k^o \rightsquigarrow e(N_i) \mid N_i \in D\right]$$

- Throughput per thread: $\frac{\mathcal{T}}{P}$
- Probability of operation of type o and key k: $\mathbb{P}\left[Op = op_k^o\right]$
- Instantiate $\mathbb{P}\left[op_k^o \rightsquigarrow e(N_i) \mid N_i \in D\right]$ based on the particularity of data structure

Poisson Rates

$\mathbb{P}\left[op_{k'}^{src} \rightsquigarrow e(N_k) \mid N_k \in D\right]$ for Skip list:



- N_j is in the structure if the latest operation on N_j is an insert
- Obtain the probability of a node to be in D (Thanks to memoryless and stationary access pattern)

Access Latency

- Applying expectation to the access latency of N_i:

$$\mathbb{E}\left[Access_i\right] = t^{cmp} + \mathbb{E}\left[CAS_i^{exe}\right] + \mathbb{E}\left[CAS_i^{stall}\right] + \mathbb{E}\left[CAS_i^{reco}\right] + \mathbb{E}\left[\sum_{\ell} Hit_i^{cache_\ell}\right] + \mathbb{E}\left[\sum_{\ell} Hit_i^{tlb_\ell}\right]$$

- Express each term according to the rates at every node $\lambda_\star^{cas}$, $\lambda_\star^{read}$
- Useful properties of Poisson Processes: Superposition and Thinning

Access Latency

Estimate the expected access latency $\mathbb{E}[Access_i]$ for N_i:

- A thread encounters a *coherence miss* while accessing N_i if the previous event of the thread on N_i is followed by CAS of another thread:
  (i) Events from the given thread $= \lambda_i^{cas} + \lambda_i^{read}$
  (ii) Superpose (Merge) CAS events from any other thread $= \lambda_i^{cas}(P-1)$

$$\mathbb{P}[\text{Coherence Miss on } N_i] = \frac{\lambda_i^{cas}(P-1)}{\lambda_i^{cas} P + \lambda_i^{read}}$$

Access Latency vs. Throughput

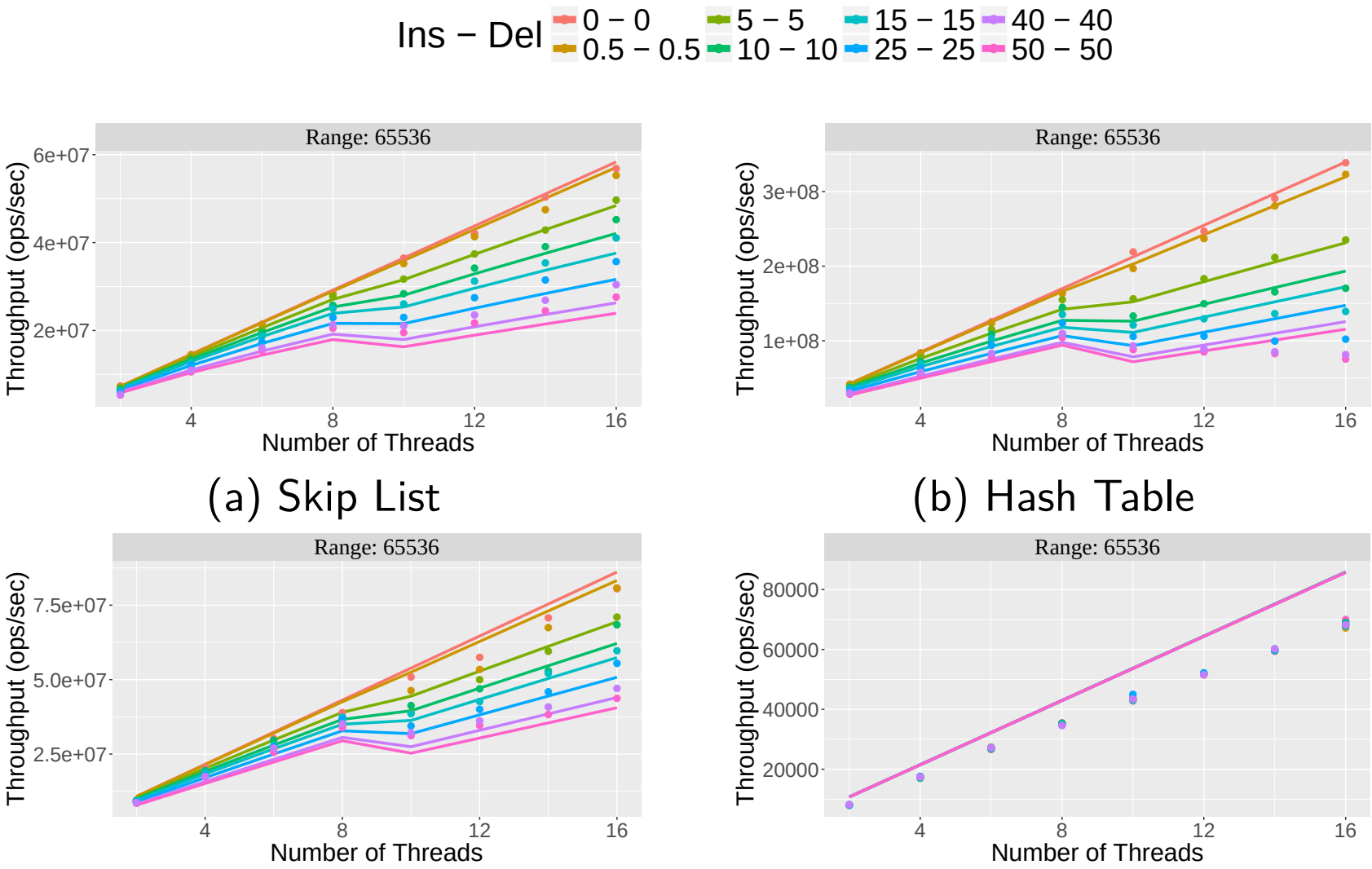
Link the access latencies and rates with throughput:

- Little's Law states that the expected number of threads accessing a node is the product of access rate and access latency
- Link latencies to throughput using Little's Law by summing over all nodes and application latency

$$P = \sum_{i=0}^{\mathcal{N}} (p_i \lambda_i^{acc} \mathbb{E}\left[Access_i\right])$$

P: *Total number of threads*
$p_i \lambda_i^{acc}$: *Average arrival rate to* N_i
$\mathbb{E}\left[Access_i\right]$: *Expected latency to access* N_i

Results



(a) Skip List

(b) Hash Table

(c) Binary Tree

(d) Linked List

Conclusion

- Analytical tools for throughput of lock-free search data structures
- Validate with: *hash tables*, *skiplists*, *linked lists*, *binary trees*
- Could be useful:
  - Compare different lock-free designs
  - Facilitates the design decisions
  - Drive the tuning process (*e.g.* memory aligment strategies)