IMPROVING APPLICATION FAULT-TOLERANCE WITH DIVERSE COMPONENT REPLICATION

João Soares, Nuno Preguiça
CITI – DI / FCT / Univ. Nova Lisboa

Motivation

- Software Bugs compromise system/application **availability** and **reliability**
- Causing applications to crash or produce erroneous results

Motivation

- Developers rely heavily on third party components, these present a great source of software bugs
  - Mostly designed for generic use
  - Testing does not contemplate specific usage scenarios

Motivation

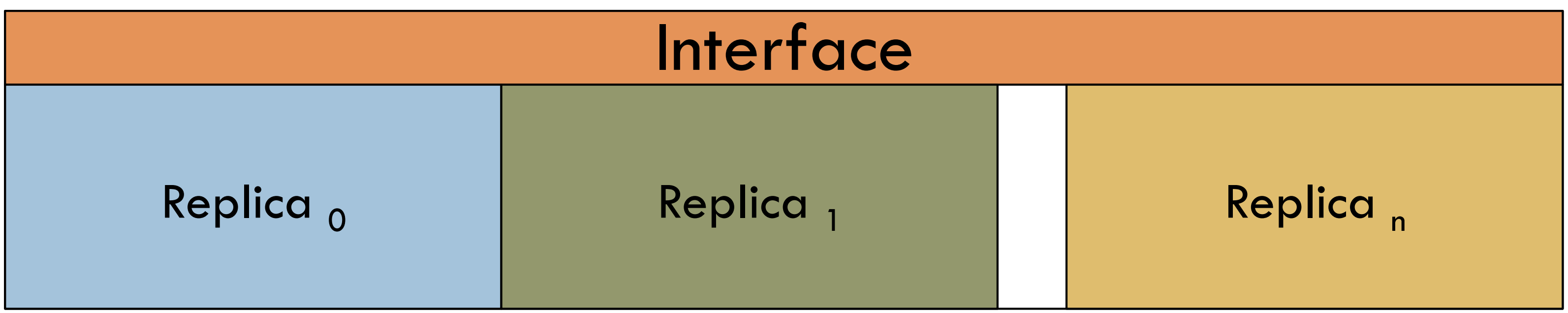
- **Replication** & **Diversity** have been used as mechanisms to deal with these faults
  - Replication prevents fail-stop faults
  - Diversity detects and prevents additional faults

Objective

- Provide run-time fault detection and prevention
  - For single machine multi-core systems
- Create a framework for developing fault-tolerant components
  - Relying on **existing third party components**
- Improving application fault-tolerance
  - Minimum impact during software development

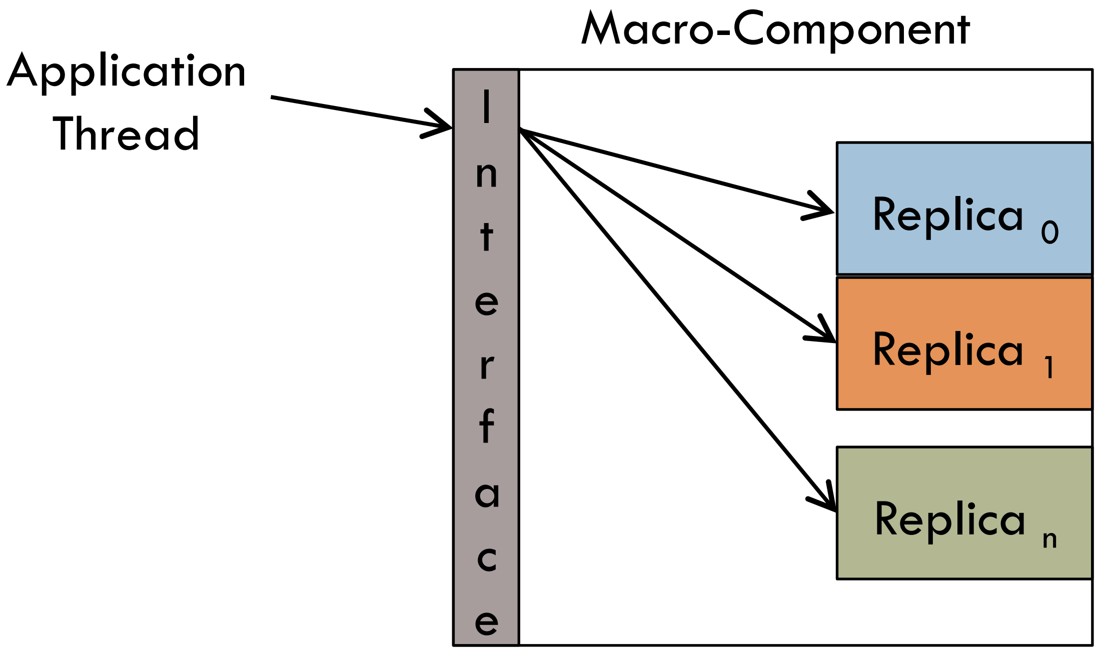
Macro-Component (MC)

- **Abstraction** that encapsulates several diverse implementations of the same interface
  - Called Replicas



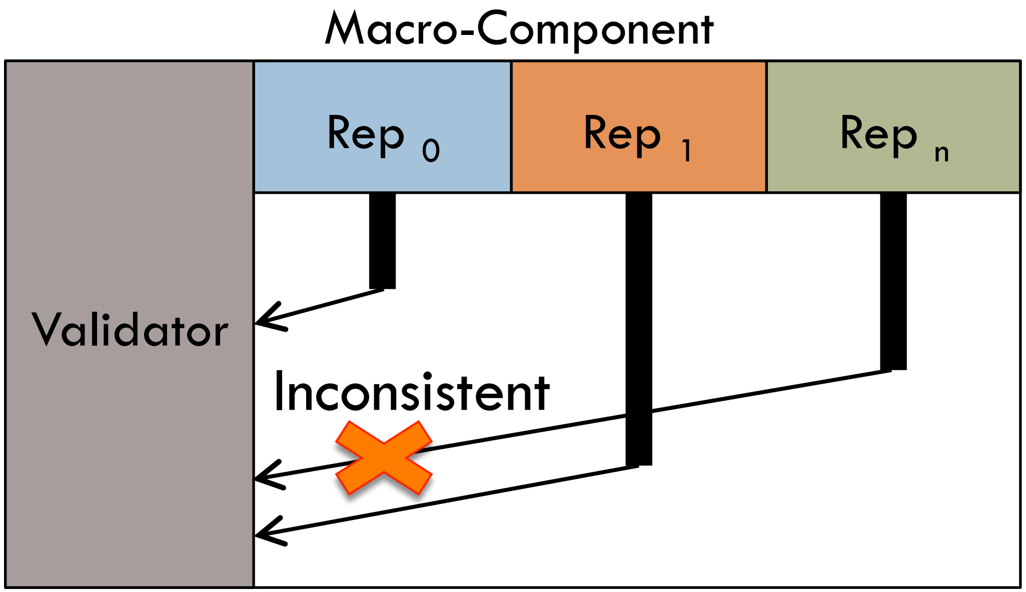
Macro-Component (MC)

- Faults are detected by
  - Executing operations on all Replicas



Macro-Component (MC)

- Faults are detected by
  - Executing operations on all Replicas
  - Comparing the set of obtained results
    - Results contradicting the majority are considered faulty



Requisites

- Operations need to execute in the same order on all Replicas
- Guaranteeing Replica state consistency
  - Allowing detection and prevention of faulty behavior

Possible Approaches

- Sequential Update Approach
  - Update operations are executed sequentially on each Replica, ordered at calling time
  - Read operation are executed concurrently
- Guarantees execution order in all Replicas (+)
- Restricts performance (–)
  - Different operations can have different performances
    - Faster operations can be held by slower ones

Possible Approaches

- Concurrent Update approach
  - Read and Update operations are executed concurrently on the Replicas
- Reduces performance constraints (+)
- Does not guarantee execution order (–)
  - Replicas can offer different performance for the same operation
    - Operations can execute faster on some Replicas

Possible Approaches

- Concurrent Update approach
  - Read and Update o… y on the Replicas
- Reduces performance co… (+)
- Does not guarantee execution order (–)
  - Replicas can offer different performance for the same operation
    - Operations can execute faster on some Replicas

Need to use a mechanism for totally ordering operations on **all Replicas**

Our Approach

- Operations on MCs are mapped into Transaction Groups
  - Operation on a Replica is wrapped by a transaction
  - Group them into Transaction Group (TG)
- Executed concurrently on the Replicas

Our Approach

- We still need to preserve transaction order on Replicas
  - All transactions of a TG need to execute in the same order
    - i.e., TGs need to be (totally) ordered

Ordering Approaches

- TGs can be order *a priori*
  - When the operation is called on the MC
- TGs can be order during at the commit phase
  - The first transaction of the group to commit defines the order for all group transactions (i.e., the TG order)

Ordering Compromises

- *A priori* order
  - Less complex solution (+)
    - Transaction only start after previous ones
  - May compromise performance (–)
    - Faster transactions can be held by slower ones
- Commit phase ordering
  - Does not compromise performance (+)
    - Slower transaction do not held faster ones
  - More complex (–)
    - May increase transaction abort rate

Preliminary Studies and Results

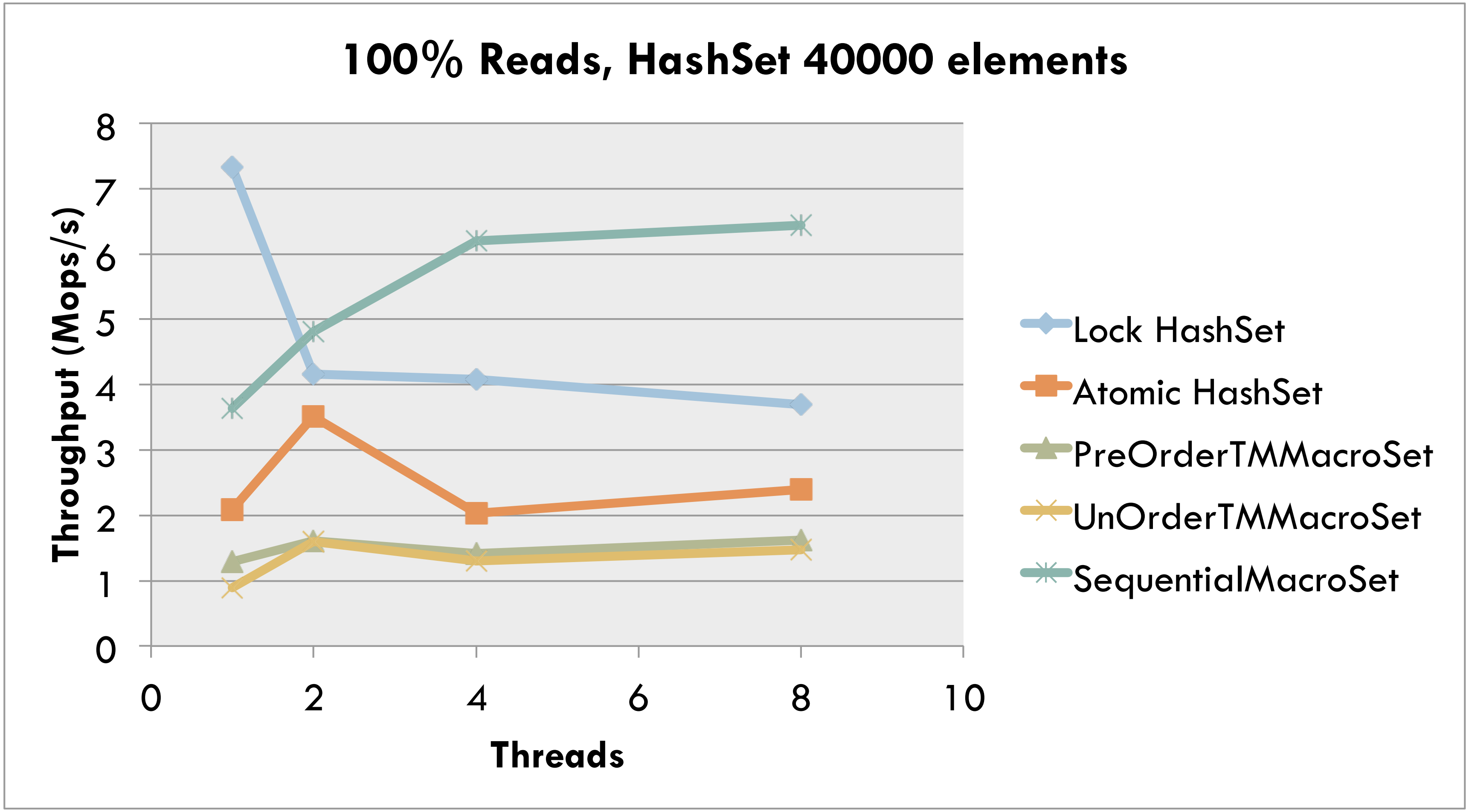
- Study the impact for possible approaches
  - Used a Micro-Benchmark
  - Executing a fixed number of operations on different implementations of the same component
    - Collection
- Macro-Components use identical Replicas
  - Without result validation
  - All Replicas perform the same number of read/write operations

Test bed

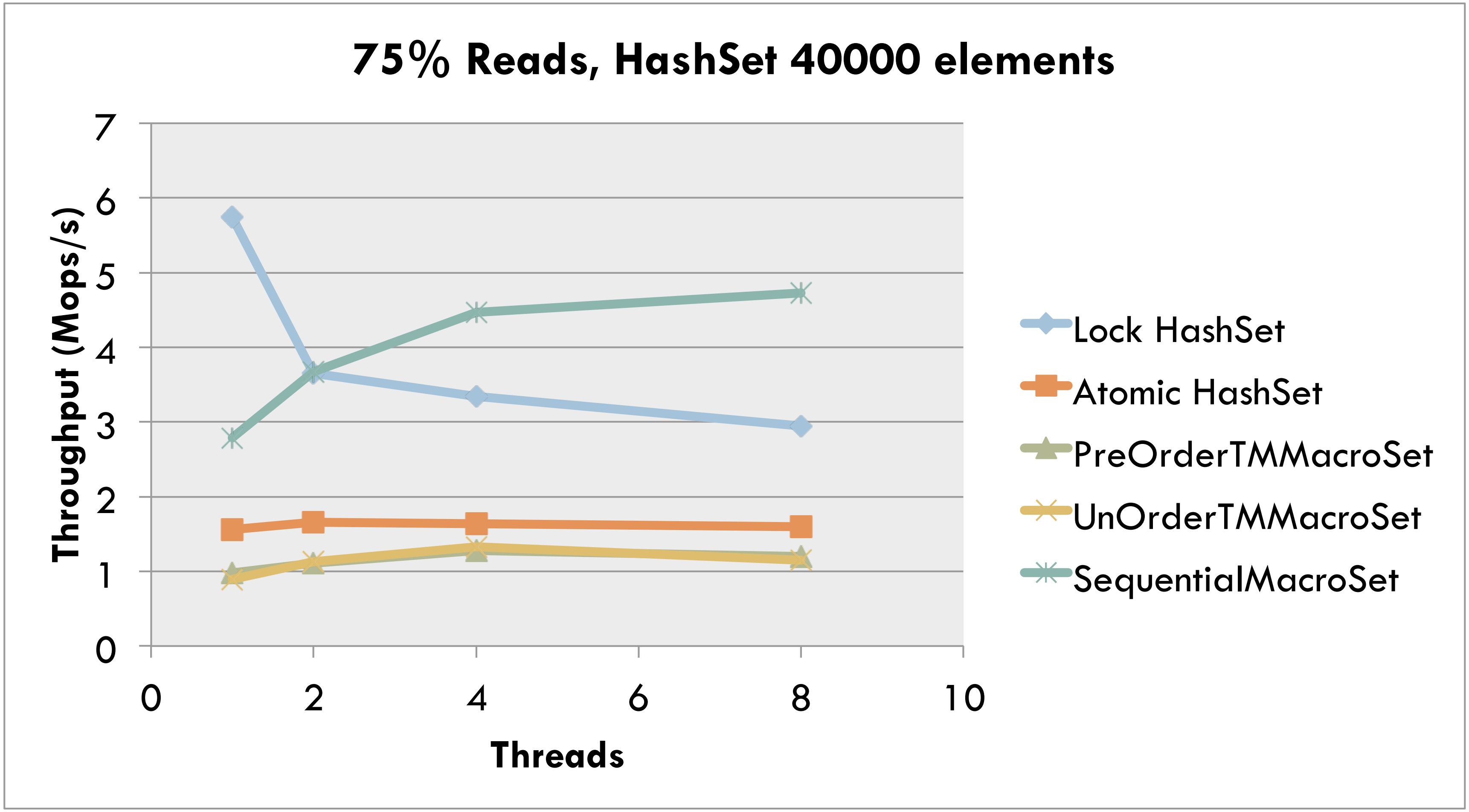
- Sun Fire X4600 M2 x86-64 machine
  - 8 dual-core AMD Opteron Model 8220 processors
  - 32 GByte of RAM, running Debian 5 (Lenny) OS

- Modified TL2 STM
  - Using Deuce framework
  - With additional states
    - Pre-commit after validation

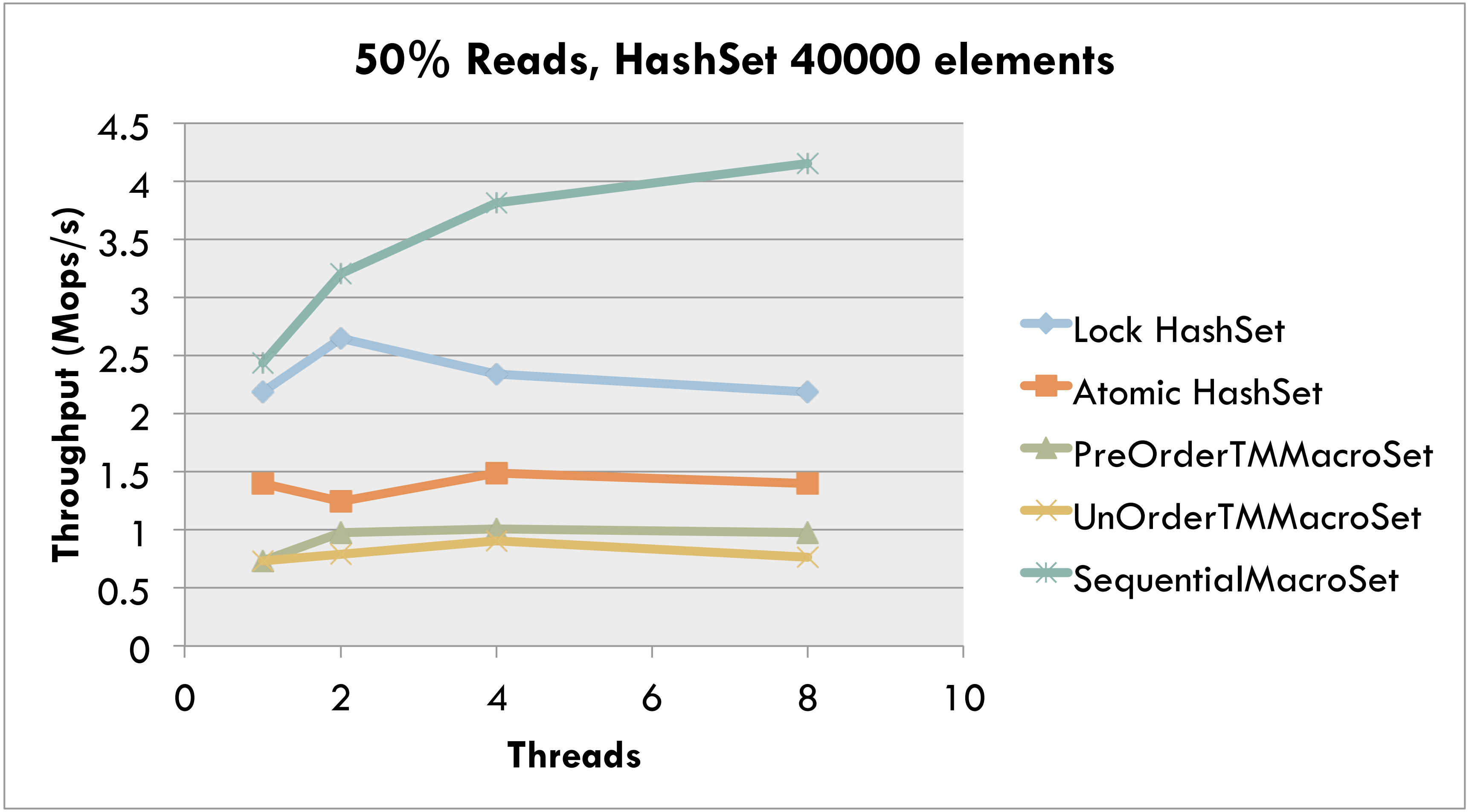
Preliminary Results



Preliminary Results



Preliminary Results



Discussion

- Sequential Update approach show good results
  - At least for small complexity components
  - More complex components should also be tested

- Benefits of TM usage and different ordering of operations
  - May provide improvements for operations with different complexities/performances
    - Components need to be developed for TM usage!

- Thank you!