FORMAL SPECIFICATION AND MODEL CHECKING OF RAFT IN SPIN

Supervisor: Dr Srinivas Pinisetty

Efforts By: Akshit Dudeja (21CS01026)
Department of Computer Science and Engineering
School of Electrical and Computer Sciences

AGENDA

01 **Introduction to Formal Verification**

Understanding the role and significance of consensus in distributed systems.

02 **Introduction to Consensus Algorithms**

Understanding the role and significance of consensus in distributed systems.

03 **Overview of Raft Algorithm**

Detailed examination of the Raft consensus algorithm and its mechanisms.

04 **Research Objectives**

Outline of the primary goals and aims of the research study

05 **Modeling Approach**

Discussion of the methodologies used for modeling the Raft algorithm.

06 **Safety Properties**

Identification of critical safety properties relevant to the Raft algorithm.

07 **Verification and Results**

Presentation of verification outcomes and analysis of results obtained.

08 **Conclusions and Future Directions**

Summary of findings and prospects for future research in this area.

FORMAL VERIFICATION OVERVIEW

Definition

Formal verification is a mathematical approach to demonstrate the correctness of systems, ensuring they meet specified requirements.

Testing vs Verificaion

It proves absence of errors compared to testing which only shows presence.

UNDERSTANDING MODEL CHECKING

Definition of Model Checking

Model Checking is an automated technique that verifies if a system model satisfies a formal specification, ensuring correctness in system design

Temporal Logic Formulas

Properties are expressed using temporal logic formulas allowing for precise verification of system behaviors over time, such as LTL (Linear Temporal Logic).

Finite-State Model Representation

The system is represented as a finite-state model, where all possible states are explored to validate system properties against specifications

Mathematical Formulation

The model M is represented as a transition system (S, S_0, R), where S is the state set, S_0 are initial states, and R is the transition relation connecting states.

SPIN Approach to Verification

The SPIN approach involves converting the negation of property $\neg\varphi$ into a Büchi automaton and checking for empty intersection with the system automaton, ensuring property validity.

CONSENSUS ALGROTHMS

Consensus Algorithms

Protocols used in distributed systems to ensure that multiple nodes agree on a single value or state

Goals

Agreement
Termination
Validity

Types of Consensus

Crash Fault Tolerance (CFT) for node failures

Byzantine Fault Tolerance (BFT) for malicious behavior

RAFT ALGRORITHM

- *Raft allows a set of nodes on a network to choose a leader and agree on the state of the network in the form of exchanges messages and keeping a replicated log of transactions.*

- *Created as an understandable alternative to Paxos.*

RESEARCH OBJECTIVES

01

Formally verify Raft's safety properties

Verified important properties like liveness, uniqeness, stability etc

02

Support for arbitrary number of servers

Understanding the role and significance of consensus in distributed systems.

03

Network layer abstraction

Added a network layer for proper communication between servers

04

Monitoring capabilities

Added centralized monitoring layer

05

Heartbeat messages

Heartbeat messages from leader to other servers

06

Unified communication channels

SIngle and uniqued network channel per server

07

Improved non-deterministic event handling

Segregated code logic based on server states rather than functionality

RAFT CORE MECHANISMS

Terms

Raft divides time into terms of arbitrary length, numbered with consecutive integers. Each term begins with an election, during which one or more candidates attempt to become a leader.

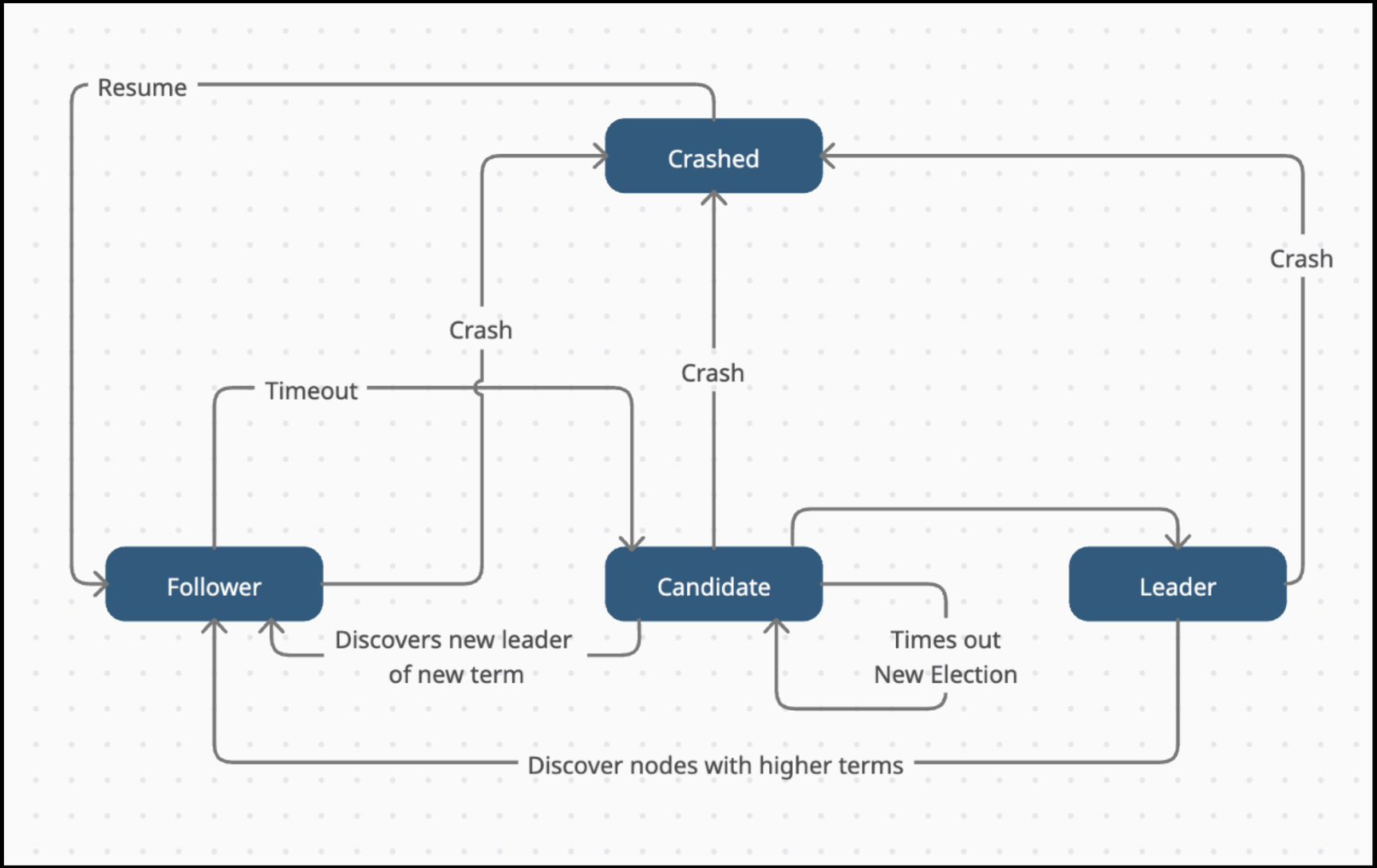
Leader Election Stage

Nodes in the network compete to become the leader and leader is elected based on election. Followers accept the leader if it has the most up-to-date log.

Log Replication

Once a leader is elected, it sends heartbeat messages to all the followers. The leader receives requests from clients and appends them to its log. It then replicates the log to followers, who apply the changes to their own logs to maintain consistency

STATES AND TRANSITIONS



STATES AND TRANSITIONS

FROM\TO	LEADER	CANDIDATE	FOLLOWER
LEADER	If it discovers a server with a higher term (AppendEntries or RequestVote RPC).	If it receives votes from a majority of the servers.	-
CANDIDATE	-	If the election timeout elapses without a majority, it restarts the election with an incremented term	If the election timeout expires without hearing from a leader.
FOLLOWER	If it discovers a server with a higher term (in AppendEntries or RequestVote RPC).	If it receives an AppendEntries from a valid leader (with a higher or equal term).	If it receives an AppendEntries (heartbeat) from a valid leader.
CRASHED	-	-	Upon Recovery

FORMAL MODELING APPROACH

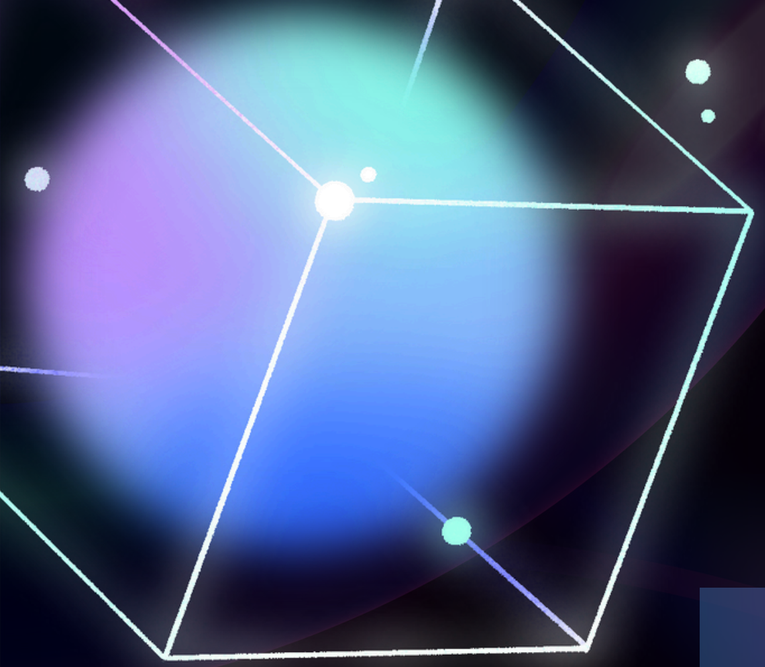
Tool:- SPIN model checker with Promela language.

Why formal verification?
Provides mathematical assurance of correctness.

Elements modeled:
- Nodes as Promela processes
- States as variables (follower, candidate, leader, crashed)
- RPCs as message channels
- Events as variable changes
- Transitions as control flow

MESSAGE ABSTRACTION AND CHANNEL

```
mtype:MessageType = {APPEND_ENTRY, APPEND_ENTRY_RESPONSE,
                     REQUEST_VOTE, REQUEST_VOTE_RESPONSE,
                     HEARTBEAT};
chan toNodes[NUM_SERVERS]=[MSG_CAPACITY] of {Message};
```



NETWORK AND MONITOR LAYER

Network process

Handles message routing between servers

- Processes messages from servers
- Routes to appropriate recipients
- Sends copies to monitor

Monitoring

Logs all system events for analysis

HANDLING TIMEOUTS

```
do
:: (serverTimeouts[sid]==0) -> atomic {
    // Handle Timeout Logic
   };
:: (serverTimeouts[sid]!=0) -> serverTimeouts[sid]--;
od;
```

SAFETY PROPERTIES LTL SPECIFICATION

Core Protocol Properties

- **Election Safety:** Ensures that at most one leader can be elected in a given term.
- **Log Matching Property:** If two logs have an entry with the same index and term, all prior entries must match.
- **Leader Completeness:** If a log entry is committed in a term, then it must be present in the logs of all future leaders

Liveness and Stability Properties

- **Stability:** Once a leader is elected, it should remain one until crashed.
- **Uniqueness:** There can be at most 1 leader at all times.
- **Liveness:** Eventually some server becomes leader

Crash and recovery conditions

- **Crash Safety:** After any crash event, the system must still guarantee that no two servers are leaders in the same term.
- **Eventual Leadership:** A previously crashed server should be able to recover and become the leader eventually.
- **Recovery Log Consistency:** After recovery, a server's log must match the current leader's log.

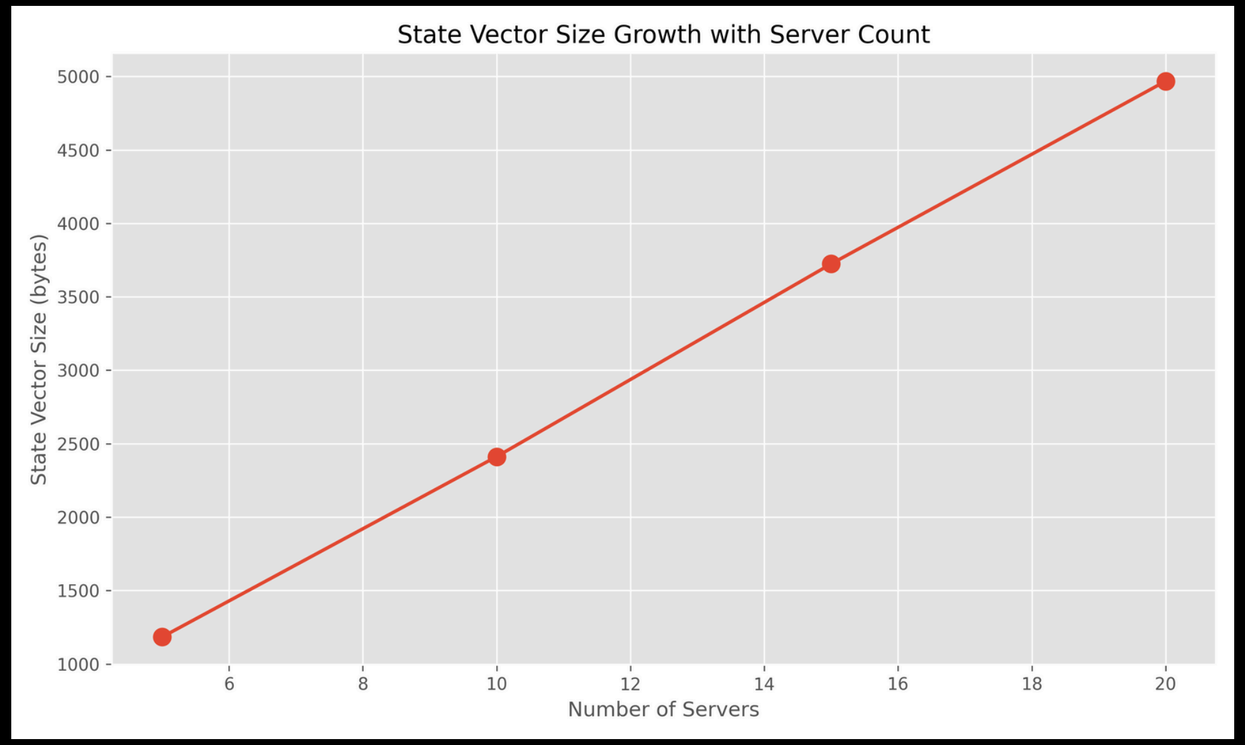
VERIFICATION

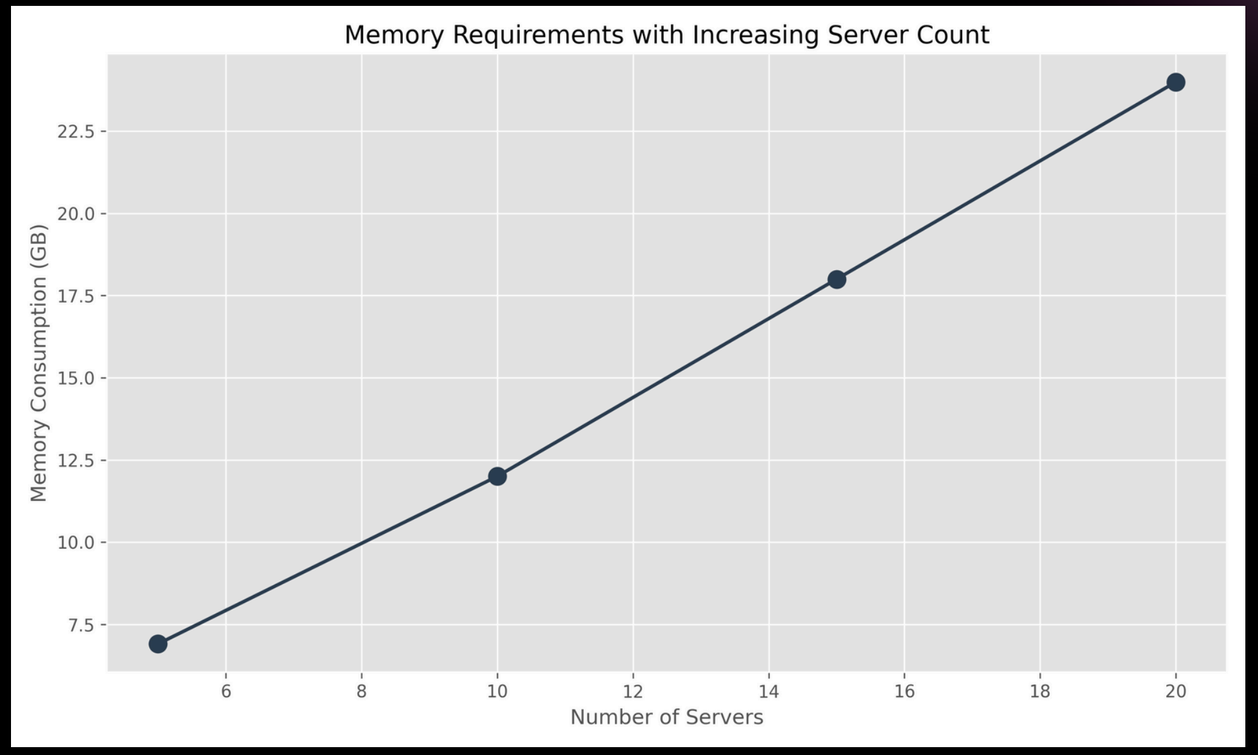
Server configurations

- Verified and Tested the model with multiple server counts ranging from 5 to 20

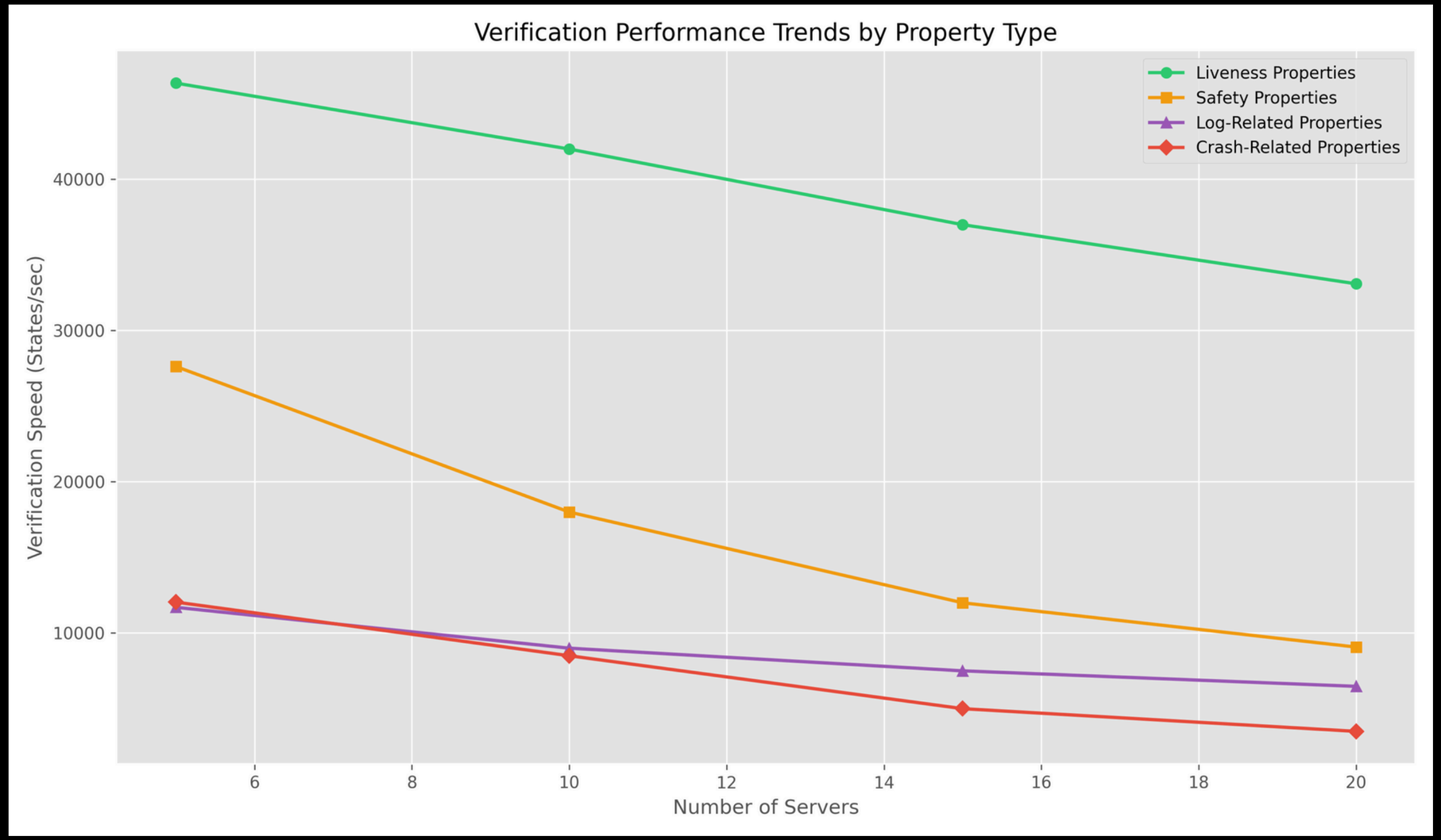
- **Setup**
  - **Search depth**: 100 and 1000
    - Limits the depth of the search
  - **DVECTORSZ**=20000
    - Sets the maximum number of unique states Spin will track.

ANALYSIS


State Vector Size Growth with Server Count
State Vector Size (bytes)
5000
4500
4000
3500
3000
2500
2000
1500
1000
6
8
10
12
14
16
18
20
Number of Servers


Memory Requirements with Increasing Server Count
Memory Consumption (GB)
22.5
20.0
17.5
15.0
12.5
10.0
7.5
6
8
10
12
14
16
18
20
Number of Servers

ANALYSIS


Verification Performance Trends by Property Type
Liveness Properties
Safety Properties
Log-Related Properties
Crash-Related Properties
Verification Speed (States/sec)
40000
30000
20000
10000
6
8
10
12
14
16
18
20
Number of Servers

KEY FINDINGS AND LIMITATIONS

- All safety properties verified successfully with lower state space and partially verified with high search space.
- No violations found in millions of states examined with high search space.

- **Scalability challenges:**
  State space explosion when servers increase.

CONCLUSION & FUTURE WORK

Contributions

- Extended Promela model with arbitrary server counts
- Partial Verification across different cluster sizes
- Detailed performance analysis with scaling insights
- Validation of crash fault tolerance

Future Work

Proper results with higher state exploration (Requires a high capacity machine to run on)

REFERENCES

[1] *Ongaro, D., Ousterhout, J. "In search of an understandable consensus algorithm."* <Link>

[2] *Qihao Bao, Bixin Li, Tianyuan Hu, and Dongyu Cao, "Model Checking the Safety of Raft Leader Election Algorithm"* <Link>

[3] *Modelling the Raft Distributed Consensus Protocol in mCRL2* <Link>

[4] *Raft Consensus(Github)*<Link>

[5] *Raft Spin(Github)* <Link>

[6] *Spin Manual* <Link>

THANK YOU!