ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 5, Issue 11, November 2017

Modular Software Model Checking For Distributed Systems

Amit Singh, Ruchi Singh, Astha Gautam

M.Tech Scholar, Dept. of CSE, Invertis University, Bareilly, U.P., India

Assistant Professor, Dept. of CSE, L.R. Institute of Engineering & Technology, Solan, H.P., India

Assistant Professor, Dept. of CSE, L.R. Institute of Engineering & Technology, Solan, H.P., India

ABSTRACT: Model checking has been demonstrated as powerful at recognizing discreet bugs in real distributed framework implementations. This research work presents amplified cache-based model checking methodology to conquer the shortcomings of existing model. This methodology is applicable for peer-to-peer (P2P) applications, which keep up numerous links at once, and where the request of messages normally influences their conduct. This paper explores the issue for part based software frameworks from four perspectives. In the first place, the entire range of QoS qualities is characterized. Second, the logical and physical fundamentals for QoSqualities are analyzed and solutions to accomplish them are proposed. Third, previous work is grouped by QoS attributes and after that acknowledged by unique reconfiguration methods. Fourth, the FIFO demanding of request is assured while reconfiguration. The proposed work assuredthatthe characterized QoS qualities can be completely accomplished under some satisfactory stipulations.

KEYWORDS: Model Checking, Modular approach, Cache- based approach, QoS, P2P

I. INTRODUCTION

Message-passing is an extensively utilized communication and programming standard as a part of the outline of concrete distributed frameworks. Though, given the intricacy imminent about because of concurrency and errors, message-passing frameworks are inclined to discreet bugs. Thus, a variety of formal procedures is upheld for ascertaining protocolaccuracy. A broadly utilized formal system for discovering bugs or demonstrating their deficiency is model checking, i.e., the automated and comprehensive investigation of the framework's state space. The proceeding with principle weakness of model checking is that the span of the full state space (and the relating time of investigation) is immovably extensive actually for small frameworks, i.e., state space explosion. [1]

A compelling measure against state space explosion is deliberation, the division of the rational, *protocol* level state space from the low-level procedure. An execution of *protocol* level develops characterizes a "one-to-many" (see in figure 1) mapping among *protocol* and procedure level states and interchanges. Once the accuracy of the implementation (i.e., the mapping all in all) is checked, another protocol can be checked on the decreased protocol level state space only [2].In the event that the execution is not demonstrated accurate, the properties that hold at protocol level are still important, e.g., to support the conceptual design, however they don't exchange to the implementation.

An alternate genericstate space reductionmethod is *partial order reduction* (POR). POR accept that the framework is characterized in terms of transitions, i.e., atomic operations that change the condition of the framework. In message passing frameworks, are the sending or accepting of messages for instance, interchanges. The assumption of POR is that the successive execution of "autonomous" transfers prompts the same state independent of the relative request of the transfers and, regularly, the intermediatestates don't affect the properties of interest. In this way, it suffices to investigate a representative execution request of such interchanges. [3]

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 5, Issue 11, November 2017

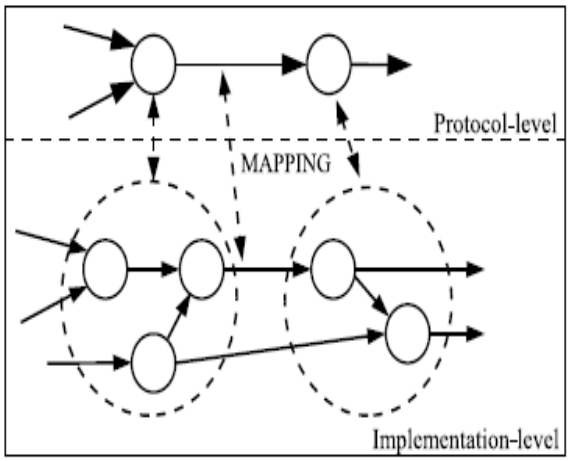

Fig. 1: Illustration of protocol and implementation-level states

Errors created by concurrency are generally hard to identify and duplicate since they just happen on certain timing. Testing does not cover each conceivable way that a framework can be executed; regardless of the possibility that it runs the framework multiple times. Simultaneous faults in this manner may in any case be remaining, significantly after the application has finished a thorough test suite. Model checking is a system to discover property intrusion in a simultaneous framework by investigating each conceivable execution way. Accordingly, each conceivable state of the framework is checked against given properties. This system is particularly valuable for quality affirmation of security perceptive frameworks and core algorithms/protocols of substantial frameworks. Model checking was initially created for hardware checking, yet the idea of state-space investigation has been connected to an extensive variety of software design frameworks also [7].

In the customary methodology, a framework to be checked is abstracted into a data languagesupported by the model checker. The model checker then creates a navigatedgraph that expresses to a state space of the framework. It marks the graph and checks if the sought properties hold at each state. This program is indicated to as state investigation. After verification of the model, the framework is normally usually implemented in a programming language, based on the confirmed model. [4]Although model checking initially obliged an unique model, recently in the group applies model checking specifically to an execution; this action is frequently called software model checking.

II. PROBLEM DESCRIPTION

Check of distributed software frameworks by model checking is not a clear task because of inter-procedure communications. Numerous software model checkers just investigate the state space of a self-contained multi-threaded procedure. Recently works proposes a procedure that applies a cache to capture communication between the primary process and its peer partner processes, and permits the model checker to finish state-space investigation. Though earlier work handles non-deterministic return in the principle process, any peer program is obliged to create deterministic return.

This exploration work presents a methodology check- pointing tool. The mix of caching and process check-pointingmakes it conceivable toward handle non-determinism on both sides of message-passing. Peer states are spared as checkpoints and restored when the model checker backtracks and produces an appeal not accessible in the database. We additionally acquaint the idea of systems with control the development of checkpoints and the overhead created by the check-pointing device.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 5, Issue 11, November 2017

III. RELATED WORK

Andreas Classenet. al. present SNIP, a productive model checker for software product lines (SPLs). Variability in software product lines is for the most commonly communicated regarding devices, and the quantity of potential items is exponential in the quantity of peculiarities. Though traditional model checkers are just equipped for checking properties against every individual item in the product line, SNIP exploits particularly designed algorithms to weigh all items in a one step. This is carried out by utilizing a compact scientific structure for product line conduct that explores similitude and expresses to the conduct of all items in a minimal way. Determination of a SPL in SNIP depends on the balance of two particular languages: TVL to depict the variability in the product line, and Promela to represent the behavior of the individual items. SNIPis in this manner one of the first devices operational with detail languages to formally express both the variability and the practices of the results of the product line. [1]

Ong'salgorithmfor checking higher-request recursion plans is somewhat complex and likely hard to understand: The algorithmdiminishes the model-checking issue to an equality game over variable profiles, and its accuracy confirmation depends on game semantics [3]. Hague et al. [4] gave optional confirmation through a decrease of the model checking of recursion plans to that of collapsible pushdown automata; their diminishment is likewise focused around game semantics.

Kobayashi [5] demonstrated that given a Buchi tree device with a trivial acknowledgement condition (a class which Aehlig [6] has called trivial automata), one can build a intersection sort framework in which a recursion plan is sort proficient if, and just if, the tree produced by the plan is acknowledged by the automaton. The favorable circumstances of the sort framework are that the perfection of the algorithm is much easier, and it is simpler to streamline the algorithm in various exceptional cases, by standard techniques for type deduction. Particularly, Kobayashi [5] has demonstrated that under the thought that the sizes of sorts and the automaton are limited above by a consistent, the checkingalgorithm runs in time direct in the measure of the recursion plan.

PRISM is a probabilistic model checker, an apparatus for formal validating and analysis of frameworks that show arbitrary or probabilistic conduct. It has been utilized to investigate frameworks from various application spaces, including communication and media conventions, security protocols, randomized distributed algorithms, organic frameworks and numerous others.

IV. REEARCH METHODOLOGY

Our work show that FIFO demanding of request is assured while reconfiguration.

Cache-based Verification

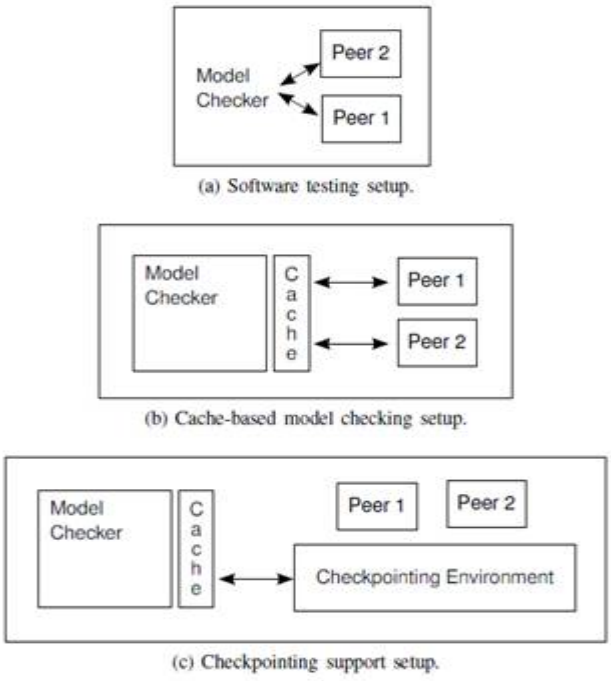

Fig. 2: Three configurations: testing, cache-based model checking and model checking with check-pointing support

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 5, Issue 11, November 2017

On a very basic level, dynamic software check can be completed by two methodologies: testing and model checking. Figures 2a and 2b look at the setups of both methodologies in the checking of adistributed application. Testing executes both the FIFO and peers in the ordinary execution environment. [8]
Figure 2c is checkpointing setup which is single execution way of the FIFOto be practiced for each one run in this setup. Though, model checking executes the FIFO inside an environment where the system state may be stimulated back. In this way, the FIFO can be efficiently determined through every conceivable execution way. In the case of a multi-process application, the I/O cache is obliged to interface with the FIFO for the peers.
The I/O cache captures each request packet, an information packet sent by the FIFO, and stores it in a private information structure. Correspondingly, response packets returning from the associates are put away in the I/O reserve also. Each one request packet is matched with its comparing response packet, if any. The I/O cache utilizes this data to emulate peer practices. Therefore, the FIFOcomes across the same communication with the I/O cache that would experience with the real peers. The single-procedure model checker then can finish the investigation of the FIFO state space. In doing this way, it stays away from a costly investigation of the full state space of each one peer. Like a partial-order reduction, this decreases the state space fundamentally. By exploring the full state space of the FIFO consolidated with just a couple of (as opposed to all) peer executions, cache based verification permits frameworks to be explored that were formerly out of scope for model checking. [8]

V. EXISTING WORK

Existing technique displays the idea of cache based model checking, which focuses on checking a single process in a distributed framework. An arrangement of methodologies, based on this idea, is clarified to help a model checker in managing distributed applications. Cache based model checking copies the conduct of nature by recording the communication between the SUT and the surrounding. Communication messages are consumed in a cache that interoperates with the model checker. The cache adjusts itself as per changes of the SUT state so it can communicate with the SUT on behalf of the real peer forms. Cache based model checking offers an adaptable system to check a single process in a framework since the state space of one procedure is investigated as opposed to the composite state space of all procedures. It additionally has the profit of permitting analyzers to check a framework with an associate process that is hard to model [9].
Besides, process checkpointing is acquainted into software model checking with capture nature state in specific situations where the cache is not sufficient to imitate the earth conduct. The analysis both applies the complete cache based model checking approach and cache based model checking with the procedure checkpointing capacity. It demonstrates that cache based model checking with procedure checkpointing help, given suitable improvement, is more capable than, yet as adaptable as immaculate cache based model checking [10].
Methodology of existing work [11]

1. Proposes routines to synchronize a methodology controlled by a software model checker with outside procedures;
2. Formalizes and arranges applications by I/O determinism;
3. Presents cache based model checking, which enhances the execution of single-methodology check;
4. Integrates check-pointing into software model checking to overcome the constraint of caching;
5. Evaluates the execution and appropriateness of each one methodology;
6. Introduces the idea of trace convergence, which keeps a single methodology model checker from finding each fault in a project.

Drawbacks of existing work as follows

- A synchronization mechanism is needed to maintain the consistency of the system.
- More optimization is possible by omitting unnecessary checkpoints.
- The current cache-based approach is limited to applications in client-server architecture.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 5, Issue 11, November 2017

VI. PROPOSED METHODOLGY

Java Pathfinder (JPF) [12] is a model checker for projects written in Java. It is utilized as the model checker and the run-time environment for FIFO reconfiguration in this work. The perfect I/O cache approach without checkpointing capacities was produced as an expansion of JPF called net-io-store for checking organized applications. One checkpoint filecommunicates to the condition of one hub, so for every checkpoint order, the quantity of checkpoints made is equivalent to the quantity of hubs presently running.

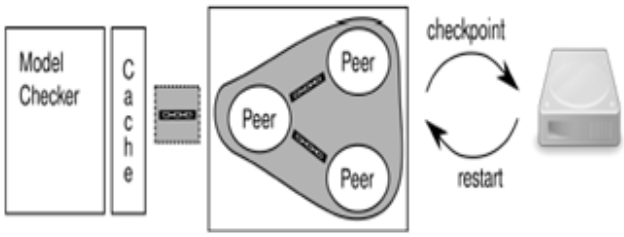

Fig. 3: The state of a FIFO and a group of peers is managed by the model checker and I/O cache, respectively

The checkpoint records contain sufficient data to restart the peer of procedures at a state where each one procedure is consistent with each other. The end goal to make work with the I/O cache, we adjust some portion of I/O cache and register callback capacities to capture the events inside the peer procedure.

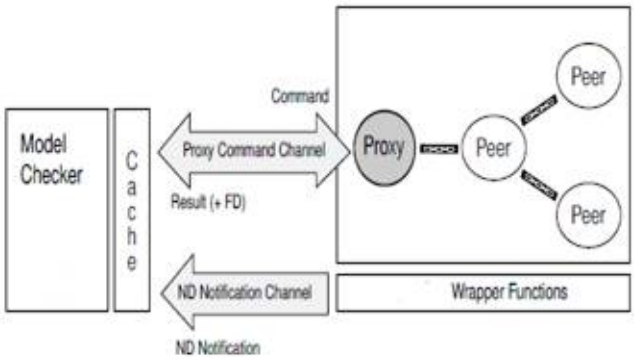

Fig. 4: The proxy process represents the FIFO inside the peer environment

Proposed implementationwork incorporates amplifying the abilities of existing approach and applying extended cache-based model checking to other application architectures, for example, P2P. An alternate plus point is to utilize JPF model checker for running peers and additionally FIFO with the goal that low-level non-determinism, for example, thread scheduling is in control. By doing this, we could investigate the associate practices in more detail and specifically perform the ones that possibly uncover faults in the FIFO. Versatility restrictions may oblige heuristics to lead the checking to where a bug is prone to happen.

VII. SIMULATION RESULTS

Simulation results have demonstrated the efficiency of our approach to find out the accuracy of proposed model checking. The below fig.5 and fig.6 presented that using the implementation of FIFO in cache based model checkingwe can schedule and verify the multiple processes at single time like in Figure 5 five main peer processes R1, R2, R3, R4 & R5 are scheduled at same time. Figure 6 present average times calculated to reconfigure the whole processes is 1.0.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 5, Issue 11, November 2017

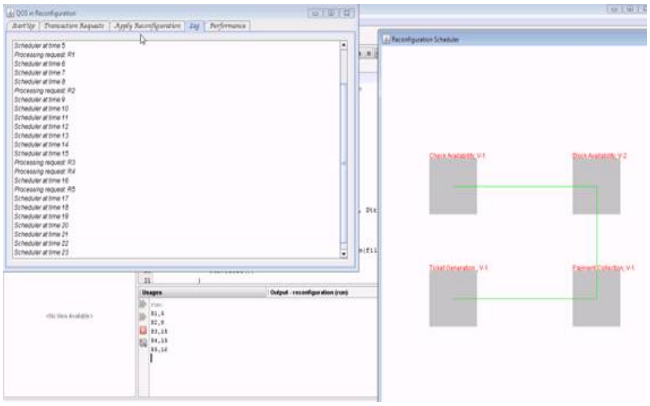

Fig. 5: Model checking of five processes at one time

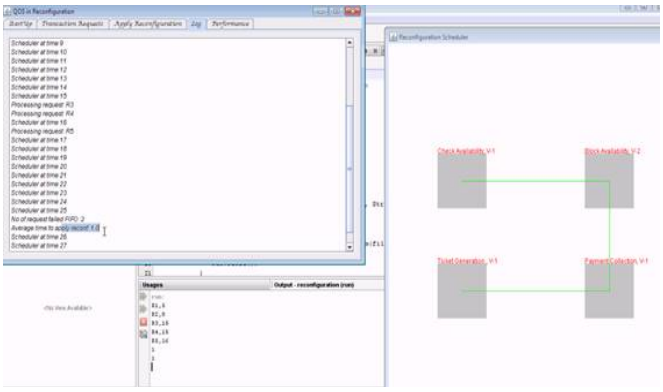

Fig.6: Present average times calculated to reconfigure the whole processes is 1.0

Fig.7: Performance of FIFO violation log files by number of requests

The simulation results shows in figure7illustrates that the implementation of FIFO violations reduces the number of violation as per number of requests.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 5, Issue 11, November 2017

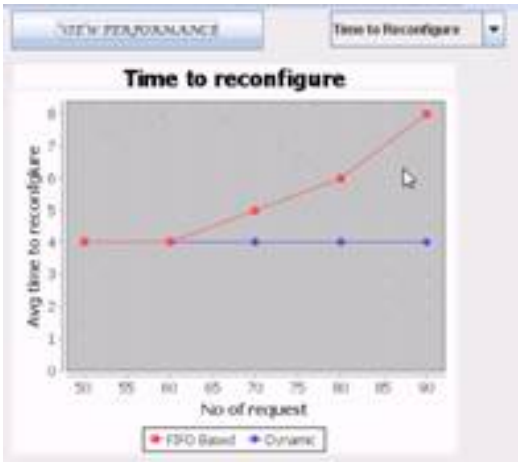

Fig.8: Average time to reconfigure the processes using FIFO violations

Figure 8 shows thataverage time to reconfigure the processes using FIFO violations shows that the average time to reconfiguration is higher and in a steady state than the existing system. As the previous work is grouped by QoS attributes and after that acknowledged by unique reconfiguration methods. But the proposed FIFO violation method process the demanding of request is assured while reconfiguration.

VIII. CONCLUSION

In this paper, an approach for model checking in distributed systems is proposed. In this research work algorithms for model checking using checkpoint in software model verification has been proposed. The results demonstrate that the proposed approach is effective in enhancing the accuracy in detecting bugs. The most vital conclusion from our examination is that the characterized QoS qualities can be completely accomplished under some satisfactory stipulations.We plan to extend the model checking in multi-processing systems and compare the results of amongdynamic and FIFO violation implementation.
For future recommendation we also plan to take account of fault detection in other major application programming.

REFERENCES

[1] Classen, A., Cordy, M., Heymans, P., Legay, A., Schobbens, P.-Y.: Model checking software product lines with SNIP. International Journal on Software Tools for Technology Transfer 14(5), 589–612 (2012)
[2] Emerson, E.A.: Model Checking and the Mu-calculus. In: Immerman, N., Kolaitis, P.G. (eds.) Proc. of DIMACS Workshop on Descriptive Complexity and Finite Models. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 31, pp. 185–214. AMS (1996)
[3] J. M. E. Hyland and C.-H. L. Ong. On Full Abstraction for PCF: I. Models, observables and the full abstraction problem, II. Dialogue games and innocent strategies, III. A fully abstract and universal game model. *Information and Computation*, 163:285–408, 2000.
[4] M. Hague, A. Murawski, C.-H. L. Ong and O. Serre. Collapsible pushdown automata and recursion schemes. In *Proceedings of 23rd Annual IEEE Symposium on Logic in Computer Science*, pages 452–461. IEEE Computer Society, 2008.
[5] N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *Proc. of POPL*, 2009.
[6] K. Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Logical Methods in Computer Science*, 3(3), 2007.
[7] Fisler, K., Krishnamurthi, S.: Modular Verification of Collaboration-Based Software Designs. In: Proc. ESEC/FSE 2001, Vienna, pp. 152–163. ACM (2001)
[8] C. Artho, W. Leungwattanakit, M. Hagiya, and Y. Tanabe, “Efficient model checking of networked applications,” in Proc. TOOLS EUROPE 2008, ser. LNBIP, vol. 19. Zurich, Switzerland: Springer, 2008, pp. 22–40.
[9] W. Leungwattanakit, C. Artho, M. Hagiya, Y. Tanabe, and M. Yamamoto, “Model Checking Distributed Systems by Combining Caching and Process Checkpointing,” Proc. IEEE/ACM Int’l Conf. Automated Software Eng. (ASE ’11), 2011.
[10] C. Artho and P. Garoche, “Accurate Centralization for Applying Model Checking on Networked Applications,” Proc. IEEE/ACM Int’l Conf. Automated Software Eng. (ASE ’06), Tokyo, Japan, pp. 177-188, 2006.
[11] C. Artho, W. Leungwattanakit, M. Hagiya, Y. Tanabe, and M. Yamamoto, “Cache-Based Model Checking of Networked Applications: From Linear to Branching Time,” Proc. IEEE/ACM Int’l Conf. Automated Software Eng. (ASE ’09), pp. 447-458, Nov. 2009.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 5, Issue 11, November 2017

[12] W. Visser, K. Havelund, G. Brat, S. Park, and F. Lerda, "Model checking programs," Automated Software Engineering Journal, vol. 10, no. 2, pp. 203–232, 2003.
[13] W. Diffie and M. E. Hellman, "New directions in cryptography," IEEE Transactions on Information Theory, vol. 22, no. 6, pp. 644–654, November 1976.
[14] W. Leungwattanakit, C. Artho, M. Hagiya, Y. Tanabe, and M. Yamamoto, "Verifying networked programs using a model checker extension," in ICSE Companion proceedings, Vancouver, Canada, 2009, pp. 409–410.