PCTL model checking based on Giraph

Yuting Lu\textsuperscript{1}, Yueming Zhao\textsuperscript{1}, Yuchun Wang\textsuperscript{1} and Xiaoyan Wang\textsuperscript{*}

\textsuperscript{1}Jilin University, Changchun, Jilin, 130000, China
\textsuperscript{*}Corresponding author’s e-mail: wangxy@jlu.edu.cn

Abstract. At present, ensuring system security and verifying whether the system meets the given requirements have been the focus of computer engineers. Model checking is an automatic and exhaustive technique to prove whether there has absence of such errors in a system which has been widely applied. It has a regular requirement for a system model under consideration and a desired property. Then it systematically checks whether or not the given model satisfies this property. However, the traditional algorithm on Hadoop for checking Computation Tree Logic (CTL) formula does not scale well because of the need of conducting a huge number of states with each passing day. To deal with the state space explosion problem, we propose a scalable algorithm on the platform of Giraph, which is a new rising iterative graph processing platform with high scalability and employs the Bulk Synchronous Parallel (BSP) model. We also address this problem by selecting Probabilistic Computation Tree Logic (PCTL) as the property specification, which is strictly more expressive and makes a great advancement on using probability to describe the probability of an event occurring. This paper comes up with a novel method based on Giraph to verify one PCTL formula, which is used to represent a maximum probability of reaching a certain state, being regarded as a quantitative description of a property in system detection. Owing much to the Giraph platform, our experiment is of great interest as it displays much more efficiently than current methods which are based on Hadoop and so on. Ultimately, we discover that the processing time of Giraph platform applying message model is much shorter than that of Hadoop platform applying iterative model through experimental comparison.

1. Introduction
Society is becoming increasingly impartible with dedicated computer and software systems in nearly every aspect of daily life\textsuperscript{[1]}. Verifying the function of the system by using effective means and whether it meets the performance indicators has become an increasingly significant issue\textsuperscript{[2]}. For fear of using a system that has not been determined to be secure, model checking is an automatic verification technique, which verifies the properties of models or propositions of a finite concurrent system by means of explicit state search or calculation of implicit fixed points. At present, it has been put into use in the analysis and verification of communication, control system, computer hardware and so on\textsuperscript{[3]}. Served as a foundation for inspecting system, it explores all the system’s possible states in a violent and exhaustive manner, showing clearly whether there are potential errors in the system. When the system does not completely satisfy the propose properties, a counter-example path will eventually be put forward\textsuperscript{[4]}. However, it is faced with the problem of state space explosion\textsuperscript{[5]}. In past studies, some solutions have been raised to address this issue, such as ordered binary tree, partial order reduction and distributed algorithm\textsuperscript{[5]}. Among all of the above solutions, PCTL can effectively alleviate the state explosion problem by allowing probabilistic quantization of the described attributes.
As the information technology which is marked by the Internet, amount of information data is growing explosive[6], indicating that human have entered the era of big data. The technology produced by the wave of big data has been widely used such as Hadoop. Whereas, the data is always be shown and stored in the form of graph, since a graph can vividly represent the real world, such as references between technical literature and relationships between social networks[7]. The size of it turns out to be a new challenge. Even though Hadoop can execute high-speed computation and storage, it is useless at iterative computations such as iterative computation in graph problems. Nevertheless, combining the idea of BSP[8], Giraph is good at dealing with it.

The present study was designed to determine how to use the PCTL model checking technique on the iterated graph processing system Giraph, and finally calculate the maximum probability matching with the model and propose algorithm. This paper is further organized as follows:

In the first part, we are going to look at what the PCTL model checking is and give an operational account of the denotational semantics of PCTL. In the second part, we mainly illustrate the advantages and basic iteration and message conversation concepts applying to run the program on Giraph and compare it with BSP model. In the third part, we will put forward the selected formula and propose an innovative method based on BSP model employing Giraph to verify the formula for a system modeled as a Discrete Time Markov Chain (DTMC) structure. The model will be transformed into a recognizable Giraph expression and applied to the algorithm. As for the specific algorithm we hold, we will concentrate on its operation results and efficiency contrasted with Hadoop. The last section includes our future work.

2. PCTL model checking

PCTL is a temporal logic which describes the properties of Markov chain, especially DTMC, it consists of propositional logic connectives, atomic propositions, operators representing time and probability[9]. It is the extension of CTL and can be applied for a stochastic system model in model checking[10]. On the basis of the CTL, PCTL adds the probabilistic operator \( P \) to substitute universal \( \forall \) and existential \( \exists \) in CTL. Next, we will focus on discussing the PCTL specifications given as formulas and offering explanation.

\[
\text{DTMC } M = (S, P, S_0, Ap, L)
\]

is a five-tuple, in the midst of this:

- \( S \) is a finite state set;
- \( P : S \times S \rightarrow [0,1] \) is a conversion probability function, and satisfies any state \( s \in S \)
- \( \sum_{s' \in S} P(s, s') = 1 \).
- \( S_0 \in S \) is initial state;
- \( Ap \) is a finite set of atomic propositions;
- \( L : S \rightarrow 2^{Ap} \) is a marking function.

The formula of PCTL is composed of state formula and path formula.

State formula: \( \phi ::= a \mid \neg \phi \mid \phi \lor \phi \mid \phi \land \phi \mid P_\psi(\phi) \). \( a \in Ap \) is an atomic proposition, We can use the Boolean operators \( \neg \phi \), \( \phi \lor \phi \) and \( \phi \land \phi \) to combine several state formulas. \( p \) is the probability, \( \Delta \) is the comparison operator, \( P_\psi(\phi) \) indicates that the probability that paths fulfill \( \phi \) is \( \Delta p \), where \( \Delta \) can be replaced by \( >, <, \geq \) and \( \leq \).

Path formula: \( \psi ::= X\phi \mid \phi U^k \psi \). \( X \) is denoted as next operator, \( U \) is called the until operator. Those path-operators \( \psi \) always need to be wrapped inside a PCTL state formula \( \phi \). \( X\phi \) means that the next state on the path will fulfill \( \phi \). \( \phi U^k \psi \) means that \( \phi \) holds on all the states along the path until \( \psi \) holds in \( k \) steps or less. Note that when \( k \) is not provided, it means that \( k = \infty \).

Semantic interpretation of PCTL formula on DTMC:

The satisfying relation \( \models \) of the state formula is defined as[3]:

\[
\text{DTMC } M = (S, P, S_0, Ap, L)
\]

is a DTMC, make \( a \in Ap \) for atomic proposition, \( s \in S \).

\[
s \models a \iff a \in L(s)
\]
\( s \models \neg \phi \) iff \( s \not\models \phi \)  
(2)

\( s \models \phi \land \phi_2 \) iff \( s \models \phi \) and \( s \models \phi_2 \)  
(3)

\( s \models P_{\phi}(\phi) \) iff \( P_{\phi}(s) = \phi \in \Delta_p \)  
(4)

\( P_{\phi}(s) = P_{\phi} (= \{ \pi \in \text{Paths}(s) \mid s \models \phi \}) \)  
(5)

3. Giraph

Giraph is a brilliant platform to process the large-scale graph with little time and it is propitious to implement iterative algorithms executed on Hadoop. It encapsulates the Mapper in MapReduce and could be regarded as a variant of the BSP model. Giraph is made up of a series of supersteps, in which the local computation is performed internally\(^{[12]}\). Giraph is a highly scalable interactive graph processing system composed of ZooKeeper, JobTracker and TaskTracker as shown following.

3.1 Basic idea of Giraph

The basic idea of Giraph expresses formally like every superstep in Giraph can be broken down into three steps. Firstly the user-defined function is invoked for each vertex in parallel\(^{[13]}\). Then the active vertexes send messages to others or vote to halt and finally, every vertex comes up with some computations depending on the incoming partial results which potentially alter the vertex’s internal state\(^{[14]}\) and change to active. The state transition of vertex is shown in figure 2.

3.2 Bulk Synchronous Parallel and Giraph

Some of the terminologies in Giraph are borrowed from the BSP model, but with a slight difference in calculation. In the above article, BSP is a parallel computing model and it was firstly devised as a "transition model" between software and hardware in the field of parallel computing. Its design goal is to provide a good theoretical model of software development, which is independent of the specific architecture and has scalable parallel performance, for the various parallel architectures that may appear now and in the future.

The conceptual organization of BSP model and Giraph is shown in figure 3\(^{[8]}\). Unit 1 to Unit 5 refer
to the parallel computing process, each node can have more than one parallel computing process. Computation is the calculation of a single Unit, each Unit will split some nodes for calculation. Communication is the correspondence between Units, all data used need to be synchronized through messaging between Units. Each synchronization is the completion of a superstep and the beginning of the next superstep. When a processing unit completes the calculation of its sub-problems and sends its intermediate results, it also waits for other processing units to be finished.

Figure 3. BSP model to Giraph mapping.

According to figure 3, we know that a superstep can be divided into three steps: local computing, communication, and synchronization barriers. These steps are calculated one after another. Based on the ideology of the BSP model, Giraph computes supersteps by overlapping local computing and communication. The processing unit no longer waits for vertex calculations to complete before sending messages. Instead, it begins interchanging messages straight after they are generated. This special computing method can dilute network utilization in the longer term. This aspect is particularly helpful in Hadoop clusters running multiple jobs, of which Giraph is one of them, so Giraph does not saturate the network[8].

4. PCTL model checking on Giraph

4.1 The storage format of a graph

It is well known that the reliability of a system needs to be judged in a professional way, such as integrity and availability. In order to analyze the quality of one system accurately, we provisionally want to know the following question: What is the max probability for a host to transform into the given state.

For purpose of modeling an automated system and describing its state transitions with probability, we must choose a format to perform it. Nowadays, DTMC model is widely used for model checking to represent the behavior of a state migration system. As introduced in section two earlier, DTMC model can be shown in a triad which includes a finite state set, a conversion probability function and a marking function. It is equivalent to a weighted graph in which each node represents the state that the system may reach and edges represent the transitions of states.

For convenience, the formalized description of a DTMC model as shown in figure 4 was selected as the initial graph for this model checking research.
In order to perform our PCTL model checking algorithms based on Giraph, the DTMC model will be stored in the JSON format[15] which is commonly used in Giraph as shown in the following table 1.

Table 1. Input JSON format.

|     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|
| 0   | 0.0 | [0,1.0],[[1,0.6],[2,0.4]],[[2,0.4],[6,0.6]],[[1,0.6],[5,0.4]] |
| 1   | 1.0 | [[1,1.0],[4,1.0]] |
| 2   | 2.0 | [[2,1.0],[4,1.0]] |
| 4   | 4.0 | [4,1.0] |
| 3   | 3.1 |
| 5   | 5.1 |
| 6   | 6.1 |

Each row represents the information of a vertex, which is expressed in the following format: [vertex ID, the value of the vertex,[[target vertex ID, edge weight],[target vertex ID, edge weight]],...]. It is worth noting that the vertex identification is treated as a probability value in the PCTL. The weight of the outgoing edge namely the transition probability between states. It would stand to reason that the data shown in Table 1 can be obtained by this rule of JSON format and the equivalent weighted graph shown in figure 4 of DTMC model.

4.2 PCTL algorithm based on Giraph

Our algorithms based on Giraph for the PCTL formula is down below.

\[ P_{\text{max}}(F(s = 3)) \] (6)

The model checking algorithm for the formula we selected means to verify the maximum probability of the initial vertex when the final state is 3. In the vertex-centric programming model provided by Giraph, it becomes very natural to implement it and this algorithm has much in common with the PageRank Computation algorithm in Giraph, in which all vertexes get messages and calculate through adjacent nodes and messages received. The time when to vote to halt rests with the iterative times. First all vertexes multiply the weight of all their outer edges by the value of the corresponding target point. Find the maximum value in the result and compare with it the original value at that vertex and update its value according to the greater one and send messages to its precursor vertexes subsequently. Vertexes’ halting condition depends both on the changing size of the value and the number of iterations. The vertex is woken up when it receives message and vote stops after each evaluation of a set of messages.

How to find the precursor vertex becomes the main problem of the algorithm. We first proposed to store the precursor vertexes identity of each vertex in an array, which need to be operated in the first two supersteps. But because it allocates a corresponding space for each vertex to store its array about all precursor nodes, this kind of practice occupies a large storage space. So storing precursor nodes in an array is not a good idea. Therefore, the task becomes finding an algorithm not only can find the precursor nodes but also don’t waste much memory. We then hold the edge-adding thinking and the integrated algorithm is as follows in algorithm 1.
Algorithm 1. MCWithGiraph.

**function** maxProb (vertex, message) **do**
  **if** getSuperstep() == 0 **do**
    sendMessageToAllEdges(vertex, vertex.getID())
  **end if**
  **if** getSuperstep() == 1 **do**
    **for** msg in messages **do**
      vertex.addEdge(Edge(msg, -1))
    **end for**
  **end if**
  **if** getSuperstep() > MaxStep **do**
    vertex.votetoHalt()
  **end if**
  **else do**
    minmax1 = vertex.getValue()
    **for** edges in vertex.getEdges() **do**
      endpoint = edge.getTargetVertexId()
      weight = edge.getValue()
      minmax2 = weight * endpoint.getValue()
      **if** (minmax2 > minmax1) **do**
        minmax1 = minmax2
      **end if**
    **end for**
    minmax2 = vertex.getValue()
    vertex.setValue(minmax1)
    **if** minmax1 - minmax2 > \( \varepsilon \) **do**
      **for** edges in vertex.getEdges() **do**
        **if** edge.getValue() == -1 **do**
          endpoint = edge.getTargetVertexId()
          sendMessage(endpoint, 0)
        **end if**
      **end for**
    **end if**
  **end else**
**end function**

In this algorithm, all vertices propagate random messages along the outgoing edges in the beginning. In the next superstep, any vertex in the graph can receive a message from its precursor node, then the edge pointing to the precursor node can be added and the weight is initialized to -1. Above goals can be achieved by calling the addEdge function in Giraph. On account of the existence of these newly added sides, we can easily find the precursor vertexes and send messages to them without wasting space.
The input of our experiment is a graph of DTMC model, and the output after algorithm’s calculation is the same sized DTMC model. There is the value of vertex in each row which means to express the probability meeting the given formula, and the weight of edge shows the result after several iterations of the algorithm. Specifically, the main contribution of this work is to implement the model checking algorithm on the basis of reducing the explosion of the state space. It has obvious advantage in time to implement this algorithm when base on the Giraph platform. The efficiency comparison with the most prevalent platform for processing big data, Hadoop, is shown below in figure 5.

It is apparent to see that our algorithm benefited form Giraph has tremendous superiority compared with Hadoop. The larger the scale of the graph is, the more advantages it has.

5. Future work
Model checking can be reduced to the calculation of graphs. However, Hadoop was initially used to solve big data's batch processing problems, but its processing of graph data is not ideal. Therefore, this paper proposes a model checking method based on Giraph. The experimental results in distributed mode show that the performance of the method proposed in this paper is better than the method based on Hadoop.

However, how to optimize the system according to the scale of the system needs to be further studied. The model checking involved in this article is based on PCTL. Next, we will try model checking based on Linear Temporal Logic (LTL) and Interval Temporal Logic (ITL)[10]. At the same time, the algorithm will be further applied to the actual system, and the security indexes of the system, such as confidentiality, integrity and availability, will be formally described and further verified[17].

Acknowledgments
This research supported by the National Natural Science Foundation of China (Grant No 61502196). Xiaoyan Wang is with the College of Computer Science and Technology, Jilin University, Changchun, China(corresponding author to provide phone: 8613596194506; e-mail: wangxy@jlu.edu.cn).

References
[1] MarkoMäkelä.(2003)Efficient Computer-Aided Verification of Parallel and Distributed Software Systems.
[2] Ji, M, Wang, H, Chen, Z.(2013)A Logic Verification method of probabilistic Computation Tree based on Automata. Computer engineering, 39: 285-289.
[3] Zhou, C., Liu, Z., Wang, C.(2012)Checking the bounded Model of probability Computation Tree Logic. Journal of Software, 23: 1656-1668.
[4] Wang, F., Hu, Y., Dong, R.(2008)Research on State explosion and Optimization Strategy in Model checking. Proceedings of 2008 Annual meeting of Guangxi computer Society.
[5] Tingyin Duan, “Research and implementation of temporal Logic Model checking algorithm based on Cloud Computing platform,” Zhengzhou University, Zhengzhou, Henan, China, 2016.
[6] Peng Chen, "RESEARCH OF PRETREATMENT OF GRAPH COMPUTER BASED ON BSP
MODEL,” Ph.D. dissertation, University of Electronic Science and Technology, Sichuan, China, 2015.

[7] Jian, X., Luo, N. (2017) Research on Multi-Division Strategy in large Map processing system based on BSP. Journal of Lanzhou University of Arts and Sciences (Natural Science Edition), 31: 88-95.

[8] Martella, C., Shaposhnik, R., Logothetis, D. (2015) Practical Graph Analytics with Apache Giraph. Springer Science+Business Media New York, NewYork.

[9] Zhang, J. (2016) Real-time random model checking. Tsinghua University Press, Beijing.

[10] Yang Liu, “Game theory semantics for PCTL model checking label-extended probabilistic Petri net,” 2014.

[11] Coursera, 2019. https://www.coursera.org/learn/quantitative-model-checking.

[12] Jinfeng Xu, “A large-scale dynamic Adaptive Graph Partition algorithm,” Ningbo University, Ningbo, China, 2015.

[13] Hsiao-Fei Liu, Chung-Tsai Su, An-Chiang Chu, “Fast Quasi-biclique Mining with Giraph,” 2013.

[14] Jimmy Lin and Michael Schatz, “Design Patterns for Efficient Graph Algorithms in MapReduce,” MLG 10. Washington, DC USA, 2010.

[15] Tingyin Duan, “CTL Model Checking based on Giraph[A],” International Society for Informatization and Engineering. Proceedings of 2016 5th International Conference on Advanced Materials and Computer Science(ICAMCS 2016)[C]. Computer Science and Electronic Technology International Society, 2016:6.

[16] Cau, A., Moszkowski, B., Zedan, H. (2002) Interval Temporal Logic. Lecture Notes in Computer Science, 164: 371-382.

[17] Yan, Y., Zhou, Y., Hui, W. (2017) Research on the Application of Model checking in formal Verification of Integrity. Computer Engineering and Application, 53:59-63,134.