CTL Model Checking based on Giraph

Tingyin Duan[1,a], Qinglei Zhou[1], Shan Pan[1], Weijun Zhu[1]

[1] School of Information Engineering, Zhengzhou University, Zhengzhou City 450001, China

[a]clickyeah@yeah.net

Keywords: CTL; Kripke; Model checking; Giraph; Hadoop

Abstract. Model checking is a popular model based formal automated verification technology. Computation tree logic is a prominent branching temporal logic for specifying system properties. In order to dispose of the state space explosion problem, we proposed a novel method based on BSP model employing Giraph which is an iterative graph processing system with high scalability. The result of the experiments shows that our method is much more efficient than current methods based on Map-Reduce model.

1. Introduction

Critical industrial system requires a high level of reliability where it is of great importance to ensure that a system meets some particular specifications. Model checking[1] is a key formal method to verify complex software and hardware system. In such case the system is modeled as a system model and the specifications are formalized as property specification. The main obstacle of model checking is the state space explosion problem[2]. Researchers have sought kinds of methods to cope with this problem such as abstraction[3], symbolic model checking with ordered binary decision diagrams[4], partial order reduction[5], bounded model checking[6] and molecular methods[7].

With the advent of the era of big data, more and more platforms to deal with big data have been developed rapidly such as Hadoop, Spark and Storm. Especially the Apache™ Hadoop® project provides an open-source software for reliable, scalable and distributed computing. And Hadoop has become the de facto standard in the research and industry uses of big data. In recent years, some researchers have tried to apply such platforms to cope with the state space explosion of LTL model checking[8, 9, 10] and CTL model checking[11,12,13]. However, Hadoop is designed for batch process of big data and restricted to Map-Reduce pattern. It is hard to design Map-Reduce algorithms based on Map-Reduce to implement CTL model checking algorithm and is inefficient when lots of iterations occur during model checking.

In this paper, we will introduce a novel efficient method based on BSP model employing Giraph to cope with the state space explosion problem of CTL model checking. We will show the algorithms for verifying whether a system modeled as Kripke structure satisfy the CTL formula EXp, $EpUq$ and EGp. In section 2, we will introduce CTL model checking briefly. In section 3, we will elaborate Giraph with an example to find the shortest distance from a predetermined source vertex to any vertex in a graph. Section 4 describes our algorithms and some experiments in detail. The last section consists of our conclusion and future work.

2. CTL Model Checking

Computation Tree Logic(CTL)[14] is a famous branching temporal logic for specifying system properties. CTL is a branching-time logic which models time as a tree-like structure where each moment can be followed by several different possible futures.

A basic CTL formula must consist of path quantifiers and temporal operators. CTL formulae are inductively defined as:

$$\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid A\psi \mid E\psi \quad (state\ formulas)$$

$$\psi ::= X\phi \mid F\phi \mid G\phi \mid \phi U\phi \quad (path\ formulas)$$

where $p \in AP$, the set of atomic proposition; A is the universal path operator and E is the existential path operator; X means "next", F means Finally and U means "until". It can be shown that any CTL formula can be written in terms of $\neg, \vee, EX, EG$, and EU [11].

The interpretation of a CTL formula is defined over a Kripke Structure. A Kripke structure T is a quadruple $< S, S0, R, L >$, where S is a finite set of states, $S0$ is the set of initial states, $R \subseteq S \times S$ is a total of transition relations, that is $: \forall s \in S, \exists s' \in S$, such that $(s,s') \in R$ and $L: S \to 2^{AP}$ labels each state with the set of atomic propositions that hold in that state.

A path σ in T from a state s_0 is an infinite sequence of states $s_0 s_1 s_2 \ldots$ where $\forall i \geq 0, (s_i, s_{i+1}) \in R$. For a CTL formula ϕ, a Kripke structure T and $s \in S$, T satisfying ϕ in the state s can be written as $T \models_s \phi$ and there are satisfaction relationships as follows:

1) $T \models_s p$ *iff* $p \in L(s)$;
2) $T \models_s \neg\phi$ *iff* $T \not\models_s \phi$;
3) $T \models_s \phi \vee \varphi$ *iff* $(T \models_s \phi) \vee (T \models_s \varphi)$;
4) $T \models_s EX\phi$ *iff* $\exists t$ *such that* $R(s,t) \wedge (T \models_t \phi)$;
5) $T \models_s EG\phi$ *iff* $\exists a$ *path from* s, *in which* every state *satisfies* ϕ;
6) $T \models_s E\phi U\varphi$ *iff* $\exists a$ *path* $s_0 s_1 s_2 \ldots$ *such that* $: \exists i \geq 0, (T \models_s \varphi) \wedge (T \models_s \phi, \forall j < i)$;

If T is a Kripke structure and ϕ is a CTL formula, the model checking problem is to find every state $s \in S$ such that $T \models_s \phi$.

3. Giraph

Apache Giraph[15] is a prominent iterative graph processing system built for high scalability, which is currently used at Facebook to analyze the social graph formed by users and their connections. Giraph is inspired by the Bulk Synchronous Parallel model(BSP)[16] of distributed computation just like Pregel[17], a graph processing architecture developed at Google. However, Pregel is not open source up to now. What's more, Giraph adds several features beyond the basic Pregel model, including master computation, shared aggregators, edge-oriented input, out-of-core computation and so forth. Giraph has become a natural choice for distributed graph computation at a massive scale for it has a steady development cycle and a growing community of users worldwide.

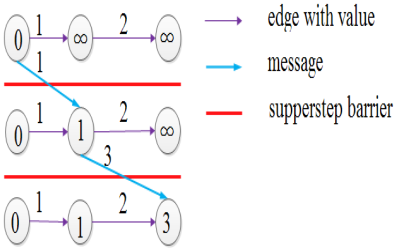

Fig.1. The Simple Source Shortest Path Algorithm

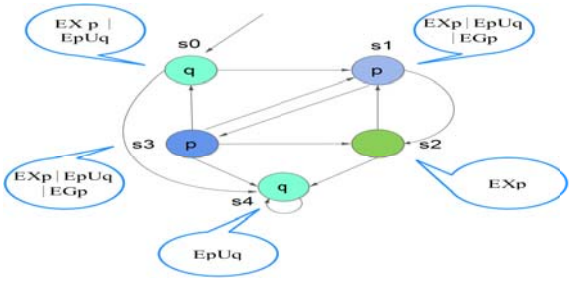

Fig.2. A CTL system model

3.1 Simple Source Shortest Paths Algorithm

In this part, a computation that finds the shortest distance from a predetermined source vertex to any vertex in a graph(SSP) [18] will be introduced to show how Giraph performs a graph algorithm before our CTL model checking algorithms are described.

As is shown in Fig. 1, the input is a chain graph with three vertices and two edges with values 1 and 2 respectively. The algorithm starts to compute the distance from the leftmost vertex. The initial value of each vertex described in the vertex(denoted by a circle) , representing the current shortest distance from the predetermined source vertex to this vertex, of the three vertices are 0, ∞ and ∞ respectively, which plus edge value may be sent as a message along the corresponding outgoing edge, resulting in an update to the target vertex's value. The computation proceeds as a sequence of

iterations, called supersteps in BSP, which will be introduced in next section. At the start, all the vertices are active. In each superstep the active vertex will invoke the compute method provided by the designer which implements the graph algorithm on the input graph.

For each active vertex, the compute method receives messages from the previous superstep and computes with the messages, the vertex value and outgoing edge values, which may lead to modifications to the value and send messages to other vertices.

Algorithm 1 shows the compute method in the single source shortest paths. It firstly finds the minimum value of all the arriving messages. If that value is less than the current value of the vertex, the vertex value will be replaced by the minimum and the new one plus each edge value will be sent as messages along every outgoing edge respectively. And after all the vertices halt and no messages are on the fly, the value of each vertex is the final shortest distance from the predetermined source vertex to this vertex.

Algorithm 1 Simple Source Shortest Paths

```
1  function compute(vertex, messages) do
2    minDist=MAX_VALUE
3    for each message in    messages do
4      minDist=min(minDist, message)
5    end for
6    if   minDist < vertex.getValue() do
7      vertex.setValue(minDist);
8      for each edge in vetex.getEdges()   do
9        distance=minDist + edge.getValue();
10       sendMessage(
         edge.getTargetVertexID(), distance);
11     end for
12   end if
13   vertex.voteToHalt();
14 end function
```

Algorithm 2 EX

```
1  function compute(vertex, messages) do
2    labels_previous=vertex.getValue()
3    if getSuperstep() == 0 do
4      if any edge value contains   "p" do
5        modify labels_previous with "EXp"
6        vertex.setValue(labels_previous)
7        labels=getLabels(labels_previous)
8        messages=vertext.getID() + labels
9        send messages to previous states
10     end if
11     vertex.voteToHalt()
12   end if
13   set edge value according to messages
14   vertex.voteToHalt();
15 end function
```

3.2 Barrier and Superstep

A barrier exists between consecutive supersteps which makes sure that the messages sent in any current superstep will arrive at the destination vertices only in the next superstep and vertices start to compute the next superstep after all the vertices have finished computing the current superstep.

What's more, values are retained across the barriers. The value of any vertex or edge at the beginning of a superstep equals to the corresponding value at the end of the previous superstep, if the graph topology is not mutated. Of course the value of the vertex and the outgoing edges can be modified during any superstep.

Any vertex may stop computing during any superstep which means such a vertex wouldn't like to be active anymore but it will be made active once any incoming message arrives. Only when all the vertices have voted to halt and there are no messages in flight will the computing stop. At last, each vertex may output some local information, which usually is the final vertex value.

4. CTL model checking on Giraph

4.1 Kripke structure in Giraph

Fig. 2 shows a CTL model which can be described by a Kripke structure. For example, state s_1 satisfies EXp for its next state s_3 satisfies p, $EpUq$ for there exists a path $s_1s_3s_4\ldots$ starting from s_1 which satisfies $p \mid p \mid q\ldots$ and EGp for there is a path $s_1s_3s_1s_3\ldots$ which satisfies $p \mid p \mid p \mid p\ldots$.

In order to perform our CTL model checking algorithms based on BSP, such a model will be stored in a Json format shown in table 1 which is a common format used in Giraph.

Table 1. A Json Format for a Kripke structure

Vertex
[s0,q-s3,[[s1,p],[s4,q]]]
[s1,p-s0+s2+s3,[[s2,null],[s3,p]]]
[s2,null-s1+s3,[[s1,p],[s4,q]]]
[s3,p-s1, [[s0, q], [s1,p], [s2,null], [s4,q]]]
[s4,q-s0+s2+s3+s4,[[s4,q]]]

Table 2. Information of System Model

Numbers of States	Total of transition relations
100	1393
200	5533
300	12974

In the first row in table 1 as an example, this first field "s0" is the name of a state regarded as a vertex ID, the second field "q-s3" regarded as value of the vertex includes labels "q" of current state and its previous state "s3", and the last field regarded as the outgoing edges with edge value from this vertex consists of the tuples of the target vertex ID and the value of each edge. We regard creatively the labels of next state as the the edge value between current state and next state, which provides a great convenience to describe and implement algorithms of CTL model checking in Giraph.

4.2 CTL algorithm based on BSP

Our algorithms based on BSP for EXp, $EpUq$ and EGp are described in Algorithm 2, Algorithm 3 and Algorithm 4. In order to compare our algorithms with those based on Map-Reduce[11, 12, 13], we first generated randomly some system models the numbers of states and transition relations of which are shown in Table 2 , and then perform some experiments on them in a single-node, pseudo-distributed cluster.

Our results are shown by Fig. 3, Fig. 4 and Fig.5. The first bar pair(blue for Hadoop, yellow for Giraph) of each figure denotes the time cost of an empty system model which is the time to start Hadoop and Giraph. The second, third and fourth denote the time for model checking EXp (Fig. 3), $EpUq$ (Fig. 4) and EGp (Fig. 5) with numbers of vertices 100, 200 and 300 respectively. There are 2 iterations in the experiments shown in Fig. 3 and Fig. 5 and 3 iterations in the experiments of in Fig.4. All the 3 figures show our algorithms cost less time than those based on Map-Reduce and Fig. 4 proves that the more iterations, the better our algorithms will perform. Further, if there are more iterations, the algorithms based on Map-Reduce turn out impossible and our algorithms work healthily.

Algorithm 3 EU

```
1  function compute(vertex, messages) do
2    labels_previous=vertex.getValue()
3    if getSuperstep() == 0 do
4      if labels_previous[0].contains(q)) do
5        modify labels_previous with "EpUq"
6        vertex.setValue(labels_previous)
7        labels=getLabels(labels_previous)
8        messages=vertext.getID() + labels
9        send messages to previous states
10     end if
11     vertex.voteToHalt()
12   end if
13   set edge value according to messages
14   if not labels_previous.contains("EpUq") and
     labels_previous.contains(p)  do
15     if any edge value contains "EpUq" do
16       modify  labels_previous with "EpUq"
17       vertex.setValue(labels_previous)
18       labels=getLabels(labels_previous)
19       messages=vertext.getID() + labels
20       send messages to previous states
21       vertex.voteToHalt()
22     end if
23   end if
24   vertex.voteToHalt()
25 end function
```

Algorithm 4 EG

```
1  function compute(vertex, messages) do
2    labels_previous=vertex.getValue()
3    if getSuperstep() == 0 do
4      if  labels_previous[0].contains(p) do
5        modify labels_previous with "EGp"
6        vertex.setValue(labels_previous)
7        labels=getLabels(labels_previous)
8        messages=vertext.getID() + labels
9        send messages to previous states
10     end if
11     vertex.voteToHalt()
12   end if
13   set edge value according to messages
14   if  labels_previous.contains("EGp") and
     vertex.getNumEdges()>0 do
15     if no edge value contains "EGp" do
16       remove "EG" from labels_previous
17       vertex.setValue(labels_previous)
18       labels=getLabels(labels_previous)
19       messages=vertext.getID() + labels
20       send messages to previous states
21     end if
22   end if
23   vertex.voteToHalt()
24 end function
```

5. Conclusion and Future Work

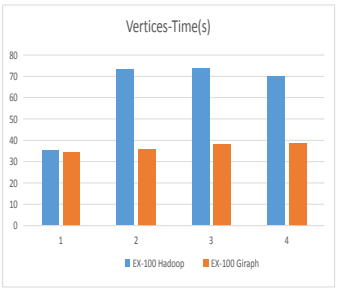

Fig.3. EX

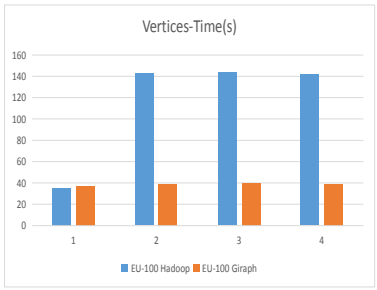

Fig.4. EU

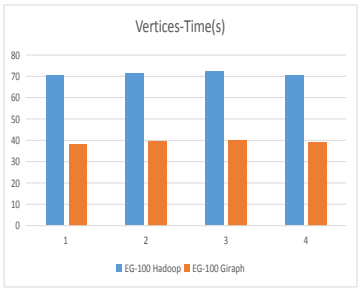

Fig.5. EG

From the result shown by Fig.3, Fig.4 and Fig. 5, it is obvious that our CTL model checking algorithms are much more efficient than those based on Map-Reduce especially when there are lots of iterations in the model checking. In our experiments, there are 2 iterations for EXp and EGp and 3 iterations for $EpUq$ because of which the difference of the time consumed is much larger. Therefore, Giraph a better choice for CTL model checking. What's more, our algorithms benefit from Giraph and is highly scalable with the system scale. In the future, we will try to apply Giraph to LTL model checking[19], CTL* model checking[20] etc..

References

[1]. E.M.Clark, O.Grumberg, D.A. Peled. Model Checking.The MIT Press,Cambridge,Massachusetts (1999)

[2]. A. Valmari. The state explosion problem. In: Lectures on Petri Nets I. London, UK:Spring-Verlag, pp.429-528 (1998)

[3]. E.M.Clark, O.Grumberg, Long, D.E.Model checking and abstraction. ACMTrans.Program.Lang.Syst., 16(5),pp.1512-1542 (1994)

[4]. J.R. Burch, E. Clarke, K.L.McMillan, D.Dill, L.J. Hwang. Symbolic model checking:1020 states and beyond. In: Logic in Computer Science,LICS 90,Proceedings,Fifth Annual IEEE

Symposium on e,1990,pp.428-439 (1990)

[5]. R.Alur, R.Brayton, T.Henzinger, S.Qader, S.Rajamani. Partial-order reduction in symbolic state space exploration. In: Computer Aided Verification, ser. Lecture Notes in Computer Science,O.Grumberg,Ed. Springer Berlin heidelberg, vol.1254,pp.340-351 (1997)

[6]. T. Latvala, A.Biere, K.Heljanko, T. Juntila. Simple bounded LTL model checking. In: Formal Methods in Computer-Aided Design,ser. Lecture notes in Computer Science, A. Hu and A.Martin,Es.Springer Berlin Heidelberg,vol.3312,pp.186-200 (2004)

[7]. Emerson E A, Hager K D, Konieczka J H.MOLECULAR MODEL CHECKING. J. International Journal of Foundations of Computer Science, 17(4) (2011)

[8]. Aziz R A. Distributed Model Checking Using Hadoop. Technical Report, Department of Computer Science, Aalto University, Finland (2010)

[9]. M. Xie, Q. Yang , J.Zhai. A vertex centric parallel algorithm for linear temporal logic model checking in Pregel. J. Journal of Parallel and Distributed Computing, 74(11): 3161-3174 (2014)

[10]. Barre B, Klein M, Soucy-Boivin M. MapReduce for parallel trace validation of LTL properties. [C]//Runtime Verification. Springer Berlin Heidelberg,184-198 (2013)

[11]. Bellettini C, Camilli M, Capra L: Distributed CTL model checking in the cloud. J. arXiv preprint arXiv:1310.6670 (2013)

[12]. F. Guo, G. Wei, M. Deng. Ctl model checking algorithm using mapreduce. [M]//Emerging Technologies for Information Systems, Computing, and Management. Springer New York, 341-348 (2013)

[13]. Camilli M. Coping with the State Explosion Problem in Formal Methods: Advanced Abstraction Techniques and Big Data Approaches. J (2015)

[14]. E.M. Clarke, E.A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Logic of Programs, Workshop. London,UK,UK:Springer-Verlag,pp.52-71 (1982)

[15]. Giraph: http://giraph.apache.org/

[16]. Bisseling R H, Mccoll W F. Scientific Computing on Bulk Synchronous Parallel Architectures. J. Ifip Transactions A, 1:509--514 (2001)

[17]. Malewicz G, Austern M H, Bik A J C. Pregel: a system for large-scale graph processing.[J]. Sigmod, 135--146 (2009)

[18]. SSP: http://giraph.apache.org/intro.html

[19]. Pnueli A. The temporal logic of programs. J. Foundations of Computer Science Annual Symposium on, 46-57 (1977)

[20]. Bhat G, Cleaveland R, Grumberg O. Efficient On-the-Fly Model Checking for CTL. [C]// IEEE Symposium on Logic in Computer Science, Lics. IEEE Computer Society,388-397 (1995)