CTL Model Checking based on Giraph

Tingyin Duan\textsuperscript{1, a}, Qinglei Zhou\textsuperscript{1}, Shan Pan\textsuperscript{1}, Weijun Zhu\textsuperscript{1}

\textsuperscript{1} School of Information Engineering, Zhengzhou University, Zhengzhou City 450001, China
\textsuperscript{a}clickyeah@yeah.net

\textbf{Keywords:} CTL; Kripke; Model checking; Giraph; Hadoop

\textbf{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.

\section{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\cite{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\cite{2}. Researchers have sought kinds of methods to cope with this problem such as abstraction\cite{3}, symbolic model checking with ordered binary decision diagrams\cite{4}, partial order reduction\cite{5}, bounded model checking\cite{6} and molecular methods\cite{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\cite{8, 9, 10} and CTL model checking\cite{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 $\text{EX}p$, $\text{EpU}q$ and $\text{EG}p$. 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.

\section{CTL Model Checking}

Computation Tree Logic(CTL)\cite{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:

\begin{align*}
\phi & ::= p | \neg \phi | \phi \vee \psi | \text{Ax} \phi \text{ (state formulas)} \\
\psi & ::= \text{X} \psi | \text{F} \psi | \text{G} \psi | \text{U} \psi \text{ (path formulas)}
\end{align*}
where \( p \in AP \), the set of atomic propositions; \( A \) 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 \( \neg, \lor, EX, EG \), and \( EU \) [11].

The interpretation of a CTL formula is defined over a Kripke Structure. A Kripke structure \( T \) is a quadruple \( < S, \emptyset, R, L > \), where \( S \) is a finite set of states, \( \emptyset \) 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 \( L : S \rightarrow 2^AP \) labels each state with the set of atomic propositions that hold in that state.

A path \( \sigma \) in \( T \) from a state \( s \) is an infinite sequence of states \( s_0, s_1, s_2, ... \) where \( \forall i \geq 0, (s_i, s_{i+1}) \in R \). For a CTL formula \( \phi \), a Kripke structure \( T \) and \( s, s' \in S \), \( T \) satisfying \( \phi \) in the state \( s \) can be written as \( T \models \phi \) and there are satisfaction relationships as follows:

1. \( T \models p \) iff \( p \in L(s) \);
2. \( T \models \neg \phi \) iff \( T \not\models \phi \);
3. \( T \models \phi \lor \psi \) iff \( T \models \phi \lor T \models \psi \);
4. \( T \models EX \phi \) iff \( \exists t \) such that \( R(s, t) \land (T \models \phi) \);
5. \( T \models EG \phi \) iff \( \exists \) a path from \( s \) in which every state satisfies \( \phi \);
6. \( T \models ED, \phi \) iff \( \exists \) a path \( s_0, s_1, s_2, ... \) such that \( \exists i \geq 0, (T \models \phi) \land (T \models \phi) \land (\forall j < i) \);

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

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](image1)

![Fig. 2. A CTL system model](image2)

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, \( \infty \) and \( \infty \) 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](image)

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 $s_i$
satisfies $Exp$ for its next state $s_j$ satisfies $p$, $EpUq$ for there exists a path $s_i,s_2,s_3,...$ starting from $s_i$
which satisfies $p \mid p \mid |...$ and $Egp$ for there is a path $s_i,s_2,s_3,...$ which satisfies $p \mid p \mid p \mid p \mid ...$

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

<table>
<thead>
<tr>
<th>Vertex</th>
<th>[s0,q-s3,[[s1,p],[s4,q]]]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[s1,p-s0+s2+s3,[[s2,null],[s3,s]]]</td>
<td></td>
</tr>
<tr>
<td>[s2,null-s1+s3,[[s1,p],[s4,q]]]</td>
<td></td>
</tr>
<tr>
<td>[s3,p-s1,[[s0,q],[s1,p],[s2,null],[s4,q]]]</td>
<td></td>
</tr>
<tr>
<td>[s4,q-s0+s2+s3+s4,[[s4,q]]]</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Information of System Model

<table>
<thead>
<tr>
<th>States</th>
<th>Numbers of States</th>
<th>Total of transition relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1393</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>5533</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>12974</td>
<td></td>
</tr>
</tbody>
</table>

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 $\text{EX}_p$, $\text{EpU}_q$ and $\text{EG}_p$ 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 $\text{EX}_p$ (Fig. 3), $\text{EpU}_q$ (Fig. 4) and $\text{EG}_p$ (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.
5. Conclusion and Future Work

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 $E_{xp}$ and $E_{Gp}$ and 3 iterations for $E_{pUq}$ 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


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


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