Test Chord fault tolerance with Molly

Abstract

Since Chord is proposed, it has derived many visions. The model of Chord is simple to build, but hard to analyze. And currently there are no published version of Chord is correct [1]. It lacks an efficient tool to prove the correctness of Chord and find tricky bugs. This paper implemented Molly [2], which is a lineage-driven fault injector reasons backwards from the correct state of system, and answer the question that how does the system reach the state with no failure. In this paper, Molly is implemented to test the fault tolerance of Chord protocol. In order to find tricky bugs, some particular input is defined.

Keyword: Chord; Dedalus; Fault tolerance

Introduction

In large-scale P2P network, how to quickly and accurately locate resource remains the main problem. Based on Distributed Hash Table, Chord addresses the problem extraordinarily. The correctness and convergence can be strictly proved in mathematical way. And as a protocol, it detailed defined every node’ s message type.

Although, Chord is special for its simplicity, but system is complex. One simple event can cause unimaginable result. Moreover, the Chord ring is dynamic, there might be node join or fail or any event any time. Under this circumstances, failure is unavoidable. Compared with programmer staying nervous and being the firefighters in the middle of the night when system fail, we prefer doing more to improve the system availability. People have implemented fault tolerance algorithm to prevent this. However, when system becomes distributed, implement level becomes extraordinary complex. The possibility of one failure of one ring is exponential level. Therefore, there is a need for an automated fault testing approach to avoid the manual processes. In Peter Alvaro’ s paper, “Lineage-driven fault injection” , he presented a set of algorithms called “Molly” which achieved this safely automated fault injection test. The ultimate goal of fault testing is to achieve the goal that when real failure occurs, the system will not stop service. And the whole system will get rid of the part which failure occur by demotion very gracefully, and without any human intervention. However, Molly has some limitation because in real network, there are many other factors such like environment configuration, network performance and hardware performance. Without considering these conditions, Molly cannot guarantee the correctness after failure part is excluded. A very simple example is that the system can easily down if the hardware is full speed running processing a data. But Molly is undoubtedly a very valuable tool for engineers. Molly will test system and give engineers information without delay.

In this paper, Molly is implemented to Chord protocol, in order to check the fault tolerance of Chord starting with events in chord rings, such as node join, node leave, node fail etc. The protocol which includes the operation is written in Dedalus. Dedalus is a logic language which has strong semantics. Dedalus is a variate version of Datalog, in which Datalog has many advantages like parallelization efficiency, supported for recursion [6]. However, in Chord protocol, node operation is mainly concerned, update node’ s state and communication between those nodes seems essential. In other words, notion of time can incredibly simplify the logic constructing of Chord model.

System model

First, there are some general introduce of Chord protocol based on SIGCOMM paper. This version is proved incorrect by Zave by using alloy analyzer. Alloy analyzer can find bugs in distributed system by testing a program within limited scopes. The input data should be small [5]. Therefore it is not useful for large-scale distributed system what Molly can do.

Chord ring is similar to a memory data structure. Node on Chord ring can either store data or as server. It will depends on what you do with Chord. This paper is making approach to construct Chord ring with Dedalus language. Moreover, in the process of adding node, how to make the network become ideal and remain ideal.

In lightweight modeling Chord analyze, it uses Alloy language and Alloy analyzer. In Alloy assumption, the meaning of a model is a collection of instances. An instance is a binding to a variable [3] [18]. Therefore, the basic idea of Alloy is that all values are related, it can express relationships of all the data types. Alloy analyzer is developed by MIT software design group. It is a tool for model analyzing based on model detection theory. The process of model testing usually works on model validation algorithms, in order to show what property is necessary to satisfy the goals we set.

Chord chooses SHA-1 as the hash function to guarantee the non-repetition of the hash by mapping the Node and the Key into the same space. There are some definition:

1. We call each node on the chord ring an identifier
2. Clockwise, the node in front of one node is called predecessor, the node behind one is called successor.

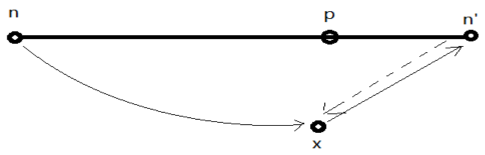


Figure 1

Assume that node n is an exist node. Node x is joins the chord.

1. Node n will find x’ s immediate successor n’ , without telling other nodes in the ring.
2. Each node runs stabilize algorithm periodically in a specific order (in this case, it does not matter because there is only one node joining). In figure , x runs stabilize algorithm: x will check x’ s successor’ s predecessor p, if p should be x’ s successor instead (compare the identifier of x and p). If x>p, n’ should be x’ s successor, else, p should be x’ s successor. And when n runs stabilize algorithm, because the predecessor of n’ is now x, then x becomes the successor of n. Now consider an example: There are nodes a, b, c and d. The order of running stabilize algorithm is c →d →a. When a runs stabilize, it will check b’ s predecessor, which is now d. Because d<b, therefore d becomes the successor of a. Notice node c is at incorrect position now.

Chord

The hash table is the key in locating node. And the original version of Chord protocol is proved incorrect. However, many people try to write the correct version of real protocol that can actually work in p2p lookup service [4,5,7,9,10], but it still lacks of further study. If we use don’ t have a tool helping us checking protocols, it will be difficult to find bugs. Because the structure of Chord is simple, so the logic in Chord is easy to write. This paper tries to verify a modified version of Chord by Zave [16]. The Chord structure uses ring topology (chord ring). The main interface of the ring is as followed:

• Insert (K, V): Store <K, V> in Successor(K)
• Lookup(K): find location of V related to K
• Update(K, new_V): update V based on K
• Join(node): node join
• Leave(): node leave actively

In this paper, I just consider the Join operation, which includes 3 subfunctions: stabilize, notify and join.

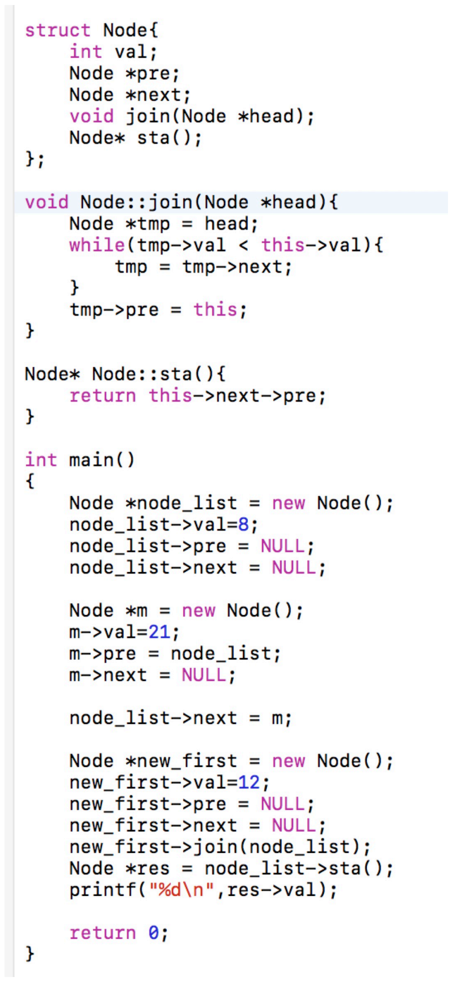
Join(): new node joins at an exsiting node, the location of node is known.

Stabilize():A queries the successor node B of its predecessor node C to determine whether C should be the successor node of A, that is to say when C is not B itself, it means that C is new join, then the successor node of A is set to C.
Notify():New node informs its existence. If C has no predecessor, or C is closer to B than B existing predecessor, then C is set as a predecessor.

Mapping and hash function is not considered in this paper. The process of join is pretty simple. The new node knows one or some of the nodes in the ring in advance, and initializes its own pointer table through these nodes. That is, the new node N will require a node in the known system to look for each entry in the pointer table. When other nodes run the probe protocol, the new node n will be reflected in the pointer table of the relevant node and the successor node pointer. The first successor node of the new node n will hand over all K whose id is less than the id of the n-node to the new node.

As the main purpose is to find out what will happen if failure occurs. First illustrate the node change and failure algorithm. When some node X leave or crashes, all the pointers in finger table which include X should update their pointers pointing at the first node whose key is bigger than X (X’ s successor). Each node periodically runs a stabilize protocol to detect new, existing or crashed node. Thereby updating its own pointer table and pointers to subsequent nodes. This insure that the system will automatically detect the node changes in rings. And this is the initial model. However, I am going to prove it is incorrect. We can see that the correctness of Chord depends on the correctness of the successor pointer, which ensure the entire network is correct. However, if some nodes fail at the same time, to ensure anyone in the ring can still know its successor and predecessor, every node should include a table which maintain the list of its successors. It is provable that locating successor time is O(log N).

To simulate the node join situation, I wrote a stabilizes operation function in C++.



```
struct Node{
    int val;
    Node *pre;
    Node *next;
    void join(Node *head);
    Node* sta();
};

void Node::join(Node *head){
    Node *tmp = head;
    while(tmp->val < this->val){
        tmp = tmp->next;
    }
    tmp->pre = this;
}

Node* Node::sta(){
    return this->next->pre;
}

int main()
{
    Node *node_list = new Node();
    node_list->val=8;
    node_list->pre = NULL;
    node_list->next = NULL;

    Node *m = new Node();
    m->val=21;
    m->pre = node_list;
    m->next = NULL;

    node_list->next = m;

    Node *new_first = new Node();
    new_first->val=12;
    new_first->pre = NULL;
    new_first->next = NULL;
    new_first->join(node_list);
    Node *res = node_list->sta();
    printf("%d\n",res->val);

    return 0;
}
```

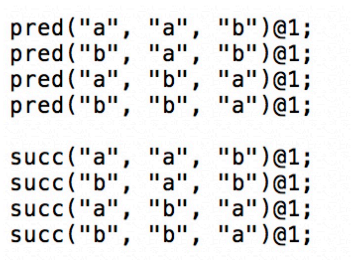
I found that it is inconvenient to create logic between different nodes. Dedalus is absolutely an easier way to implement protocol.

Protocol building

After got familiar with Dedalus, I found it very useful and efficient to build Chord with Dedalus [10]. And the key of dedalus is table. Everything is a table. When there is something on the left ":- ", it means it create an entry in the table if things on the right of ":- " are all true. For example, notify(Old, New)@async :- join(New, Old). This describe the node join. When there is a move join("c" , "b")@5 which means node c is joining b at logic time 5, it will create a table called notify(). And we can use notify() to activate the next move.

I will use four nodes to build the model. Assume that “a” and “b” is a part of a ring of any size. Node “c” and “d” are new nodes. To simplify the process of code writing, I assume that all the nodes already knows each other, and the point where “c” and “d” join is known. By “join("d", "b")@3; join("c", "b")@2;”, assume that node “c” is joining before “d”.

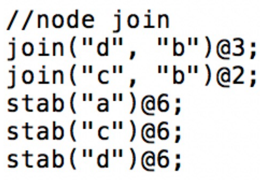
Initialize the predecessor and successor table:



```
pred("a", "a", "b")@1;
pred("b", "a", "b")@1;
pred("a", "b", "a")@1;
pred("b", "b", "a")@1;

succ("a", "a", "b")@1;
succ("b", "a", "b")@1;
succ("a", "b", "a")@1;
succ("b", "b", "a")@1;
```

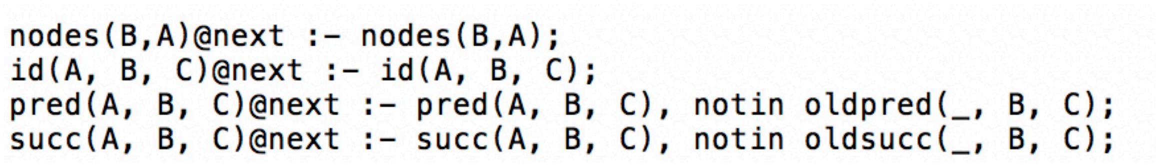
All nodes are aware of their predecessor and successor.



```
//node join
join("d", "b")@3;
join("c", "b")@2;
stab("a")@6;
stab("c")@6;
stab("d")@6;
```

The entries are defined separately. In order to make sure node “b” can correctly find its predecessor, node “d” is joined after “c”, since the correct answer is “c”. In case of “d” replaces “c”.

To keep persistence of the table, define the following rules:

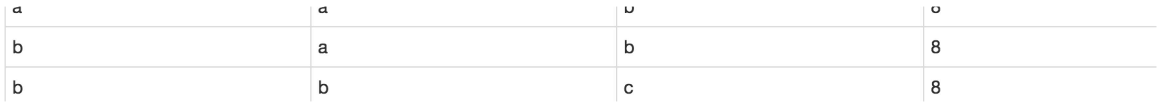


```
nodes(B,A)@next :- nodes(B,A);
id(A, B, C)@next :- id(A, B, C);
pred(A, B, C)@next :- pred(A, B, C), notin oldpred(_, B, C);
succ(A, B, C)@next :- succ(A, B, C), notin oldsucc(_, B, C);
```

This ensures that four tables persist over time. And if anything changes in the table, which cause an old predecessor or old successor, it will not appear in the table (replaced by new node).

replacepred(B, Bpred, A)@next :- notify(B, A), id(B, A, Aid), pred(B, B, Bpred), id(B, Bpred, Bpredid), Aid<Bpredid;

This is the condition when a new node A join at position B. It should satisfies the condition that the new node’ s id is more close than the old predecessor. If the condition is true, then B’ s predecessor becomes A.

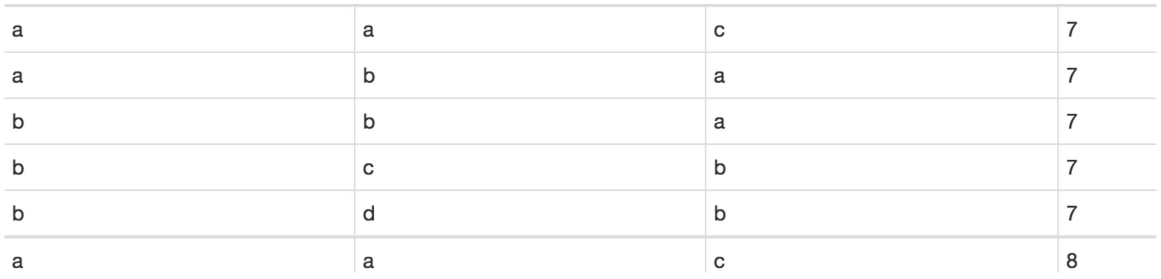


a	a	b	8
b	a	b	8
b	b	c	8

Figure2: predecessor table

From Figure 2 we can see after node c and d joins, c’ s predecessor becomes c since c’ s id is more closer than d’ s.

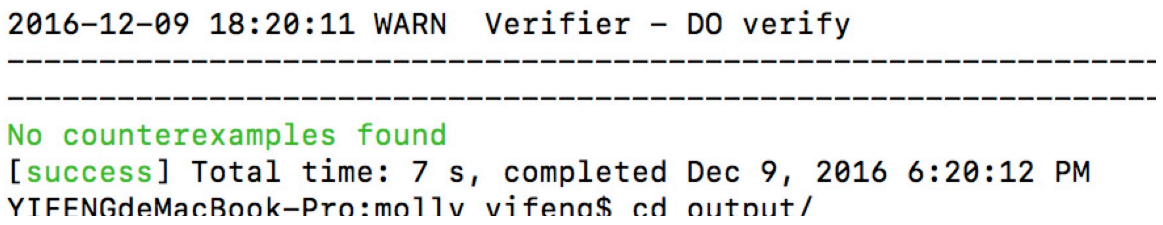
replacesucc(A, Asucc, Asuccpred) :- stab(A), succ(A, A, Asucc), pred(Asucc, Asucc, Asuccpred), id(A, Asucc, Asuccid), id(A, Asuccpred, Asuccpredid), Asuccpredid < Asuccid;



a	a	c	7
a	b	a	7
b	b	a	7
b	c	b	7
b	d	b	7
a	a	c	8

Figure3: successor table

Result shows that after stabilizes, “a” ’ s successor becomes “c” . “c” ’ s successor becomes “b” . Nodes “a b c” becomes part of the ring. As we can see from the figure, the successor of d is still b, which is wrong. Because the correct sequence is a>d>c>d instead of a>c>d. Even if d is not stabilized, its successor should be c then.



```
2016-12-09 18:20:11 WARN  Verifier - DO verify
---------------------------------------------------------------
---------------------------------------------------------------
No counterexamples found
[success] Total time: 7 s, completed Dec 9, 2016 6:20:12 PM
YIFENGdeMacBook-Pro:molly yifeng$ cd output/
```

The program runs very fast. And although there is no counterexample found, it doesn’ t mean there is no fault tolerance bugs. Because the configuration set does not include any crashes, so Molly won’ t display a counterexample. So we have to look up in the table manually. Obviously, node “d” ’ s successor is at wrong place (should be “c” instead of “b”). It is also very convenient to look up in the table.

Future Work

In this project, I successfully tested one of the Chord protocols, node join and stabilize. Because there are no nodes crash, so Molly cannot automatically find a counter example. But since I have learnt how to use Molly, I will make my code complete. It should include situations that include node crash. Because the difficulty of this project. The mechanism of Molly is so abstract that I need more time to understand it. I will learn Scala [13]. Implementing fuzz testing is unfinished. But this project gives me great experience of studying Molly and Chord and I definitely have a better understanding of them. And Molly is a very potential and useful tool in the future. As I spent many time in learning dedalus. And since I have a better understanding of dedalus, I will try to write situations which includes node crashes. And Molly will detect the failure automatically. After checked the bugs in initial version of Chord, I will try to create the modified version [16] [14], and check its correctness, see if Molly can detect bugs.

As far as I go, I find the project is really hard. I wish I could expand my protocols and take more situations into account. And it definitely needs a lot of time. Through this project, I got familiar with Molly and Chord. I believe it can be very useful in the future and in work.

References

[1] Zave P. Using lightweight modeling to understand chord[J]. Acm Sigcomm Computer Communication Review, 2012, 42(2):49-57.

[2] P.Alvaro, J.Rosen, J.M.Hellerstein. Lineage-driven Fault Injection. SIGMOD 15. ACM.

[3] Alvaro P, Marczak W R, Conway N, et al. Dedalus: Datalog in time and space[C]// International Conference on Datalog Reloaded. Springer-Verlag, 2010:262-281.

[4] Rowstron A, Druschel P. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems[C]// Ifip/acm International Conference on Distributed Systems Platforms Heidelberg. Springer-Verlag, 2001:329-350.

[5] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In Proc. SIGMETRICS'2000, Santa Clara, CA, 2000.

[6] Seo J, Guo S, Lam M S. SociaLite: An Efficient Graph Query Language Based on Datalog[J]. IEEE Transactions on Knowledge & Data Engineering, 2015, 27(7):1-1.

[7] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In Workshop on Design Issues in Anonymity and Unobservability, pages 311–320, July 2000. ICSI, Berkeley, CA, USA.

[8] P. Zave. Lightweight modeling of network protocols: The case of Chord. Technical report, AT&T Laboratories—Research, January 2010.

[9] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H.Weatherspoon,W.Weimer, C.Wells, and B. Zhao. Oceanstore: An architecture for globalscale persistent store. In Proc. ASPLOS'2000, Cambridge, MA, November 2000.

[10] Tom J. Ameloot, Jan Van den Bussche. Positive Dedalus programs tolerate non-causality ☆[J]. Journal of Computer & System Sciences, 2014, 80(7):1191-1213.

[11] Seo J, Guo S, Lam M S. SociaLite: An Efficient Graph Query Language Based on Datalog[J]. IEEE Transactions on Knowledge & Data Engineering, 2015, 27(7):1-1.

[12] Manna Z, Waldinger R. DEDALUS - The DEDuctive ALgorithm Ur-Synthesizer[C]// afips. :683.

[13] Odersky M, Rompf T. Unifying functional and object-oriented programming with Scala[J]. Communications of the Acm, 2014, 57(4):76-86.

[14] Chen Y, Sun L Z, Liu H L, et al. The Improvement of Chord Protocol about Structured P2P System[J]. Telkomnika, 2013, 11(2):393-398.

[15] Daniel Jackson. Alloy 3.0 Reference Manual. May 10, 2004

[16] Zave P. How to Make Chord Correct[J]. Computer Science, 2015.

[17] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In Proceedings of SIGCOMM. ACM, August 2001.

[18] Andoni, Alexandr; Daniliuc, Dumitru; Khurshid, Sarfraz; Marinov, Darko (2002). "Evaluating the small scope hypothesis". CiteSeerX 10.1.1.8.7702.