EPISODE 1189

[00:00:00] JM: Rust and Golang are two of the newest lower level languages for doing systems programming. They're often used for applications such as file systems, operating systems and latency-sensitive applications. So how do they compare in terms of safety, speed and programming ergonomics to each other? Linhai Song is an assistant professor and researcher at Penn State University and joins the show to talk about his work researching Go and Rust.

If you'd like to support Software Daily, go to softwaredaily.com and become a paid subscriber. You can get access to ad-free episodes and it costs ten dollars a month or a hundred dollars per year.

[INTERVIEW]

[00:00:46] JM: Linhai, welcome to the show.

[00:00:47] LS: Oh, thank you.

[00:00:49] JM: You work on programming language research. Can you give an overview of the kinds of research topics you're focused on?

[00:00:56] LS: Okay. So basically let me start on my Ph.D. During my Ph.D., my advisor educated me to do research, different techniques to combat different types of bugs. When I started to become a faculty, dependent researcher, I hope to continue this research direction. But I need to find some new angle or new aspects. I noticed that in recent years, program language designer, they propose a lot of new language. So the new language, they are designed under the goal to eliminate bugs from the language level. Basically when developers do the programming, the language of the compiler can help them to eliminate the common mistakes. So it kind of draw my attention and draw my interest. I want to see whether or not those language really completely solve the problem.

So basically this idea motivated two projects or two lines of my research. One is I take a look at a concurrency block in Go language, because Go is designed for concurrent programming. The other direction is I take a look at concurrency bugs and memory bugs in Rust, because Rust is designed for memory safety and thread safety. So basically all my current ongoing projects is focused on concurrency bugs in Go and combating memory and the concurrency bugs in Rust.

[00:02:42] JM: Can you give more detail on what a language level bug is?

[00:02:47] LS: Okay. It's kind of like you know that in order to support current programming, Go provide mainly unique primitive such as a channel. So basically one thread can send a message to another thread. This is different from a traditional concurrent programming. For C and C++, this support channel memory. So basically two threads can visit the same memory location. But in Go, Go also supported to transmit message from one threat to another thread. It is different from channel memory synchronization.

So basically when a programmer, they use a channel, they make a lot of mistake, and then they introduce concurrency bugs. Those bugs have different features from the traditional channel memory bugs. This is one thing. The other thing, I mean on the Rust side, Rust has a lot of unique language feature. For example, Rust, a lot of variable immutable and immutable. And the Rust also use left hand to automatically manage some resources. We observed that developer sometimes they misunderstood those language feature. And then they misuse those language features so that they cause kind of bugs in Rust program, in a Rust program.

[00:04:17] JM: Why does this topic interest you so much?

[00:04:22] LS: I think there are several reasons. The first reason is these two program language, Go and Rust, are really popular these days. Go was used to implement Docker and Kubernetes. So basically there could be hundreds or thousands of millions of Docker instance are running in data center right now. So that if I can identify bugs, build techniques to avoid the bugs or to detect bugs, my research will have a very large impact for Rust. Rust has become

the most beloved language in the last four years based on stack overload survey. And more and more developer, they changed from C and C++ to Rust. I believe that Rust is going to be used by a lot of – By many and many programmer. Thus, I think if I can understand the common mistake and propose design techniques to combat those common mistakes, my research will have a very large impact, a real world impact.

[00:05:34] JM: Let's focus on Go first. Can you say more about the kinds of concurrency bugs in Go that you're focused on?

[00:05:42] LS: Okay. We basically published a study paper. We sampled more than 100 real world Go concurrency bugs. And then we divided those concurrency bugs into two tabs. One is blocking bug. Basically some thread, they unintentionally block themselves. Cannot make any progress. We call them blocking bug. And the other type is non-blocking bug. Basically all threads can finish their execution. But somehow the result is undesirable. For example, a panic or exception is triggered.

So after this categorization, we also did some detailed study. We inspect their root cause and we also understand how those bugs can be fixed. This is the first project. Then this summer, our team, my team build a detection tool. Basically we used static analysis. We focused on Go bugs that are caused by misuse of channel.

Under our static analysis, basically we do not need to run the program. We do code inspection and we extend a traditional technique called constraint solving. The traditional technique, they cannot model how a channel works. They can only model how new text, how conditional variable works. We extend the existing techniques by modeling the channel behavior, channel operation. We extend this technique. We apply it to Go language. And we got another paper. We just received the notice. The paper was accepted by next year's ARS +. And our technique found the more than 300 previous unknown bug in very popular Go application such as Docker, Kubernetes, etcd, and the Go compiler.

[00:07:53] JM: What is involved in modeling how a channel works through static analysis?

[00:07:58] LS: So basically you know that channel, a channel has a sender and a receiver. A sender thread and a receiver thread. One biggest difference between channel and the channel variable, I think in that using channel, one thread can be for message from multiple channel, from multiple threads. If you want to use channel memory, you'll need to use new text. From new text and anytime there is only one thread can held. Is held in that lock of that mutex. So basically there is only one guy who is going to release the lock. There is another guy who is to acquire the lock. But in new message parsing, one thread can wait for message from multiple threads. Sometimes when those threads come at the same time, the Go scheduler or Go runtime, it will non-deterministically choose one to do the executing, to do the execution. So that the selection is really random.

So we think due to this feature, the concurrency, all the scheduling is much more complex if we use message to do the thread synchronization. We observed that if we only consider channel memory or lock, the dialogue, when it happens there are multiple strands. They need to circular with each other. But if we use channel to do the communication, we observe a very particular type of bug. There is only one or two channel debate for message. Or available to send out message on a particular channel. But no other stress is waiting for that message. It's kind of like one guy wait for others to unblock itself, but actually no other stress waiting for the blocking stress. So it's like the non-circular – The circular weight assumption is removed if we consider to use channel, if we use channel. I think this is a very big difference compared with traditional synchronization if we use Mutex and lock.

[00:10:26] JM: So this is a research domain with real practical applications, you would say.

[00:10:31] LS: Yes. We have identified bugs in Docker and Kubernetes. Basically those two applications are the most popular Go application. I think based on GitHub Stars, these two applications, both of them are among the top three.

[00:10:51] JM: Can you say more about the experience of discovering bugs in those applications?

[00:10:57] LS: Okay. So basically all those applications, they are open sourced publication. Basically their source code is available on GitHub. What we did is we downloaded the latest version of those application. We apply our technique and we identify definitely our tool that we report bugs. And sometimes there are false alarming. Both me and my students almost the whole group we inspect each reported resource to decide whether it is a bug or it is a false alarm. If we think it is a bug, we submit the bug to GitHub. Basically if we identify about in Docker, we will file issue report to the Docker application. Basically Docker developer, they will reveal our reports. They will decide whether or not what we told them is true. If they also think it is a real bug, they will create a patch or sometimes they will ask first to create a patch to fix those repository bug. And I think Go right now, there are around 200 bugs we identified are already fixed in the more recent version of those Go application. This is why we think our research generates a real world or real impact.

[00:12:31] JM: Can you give some perspective on how Rust and Go compares languages?

[00:12:38] LS: Okay. So Go is more designed for concurrent programming. We know that almost every computer, they have multi-core CPU. So concurrent programming is became more and more important. And the goal is designed for that purpose. Another feature for Go is that Go is really friendly to new language, to new guys, I mean in terms of learning Go. So it is very friendly to those new programmers. I think this is another reason why Go became very popular. And many people, they write a lot of blogs saying that Go has a potential to dominate the server-side application.

And I think probably Go has the potential to replace JavaScript. This is my personal opinion. For Rust, Rust is designed to eliminate also the memory and the concurrency bug in C program. So basically C, we know that C is very good to manipulate the low level hardware. And the C program has a very good performance compared with those backed up by a virtual machine.

And Rust, but the problem for C is that C has a lot to memory bug, because they allow user to use the pointer to access the memory directly. So it has a lot of memory bug. Those memory bug can potentially be exploited and became security vulnerability. Rust design Go is Rust want to use static compiler check to eliminate or to identify potential memory bug. If developers need to prove themselves, their code to the Rust compiler, it does not have any bug. As long as the compiler feel the code is suspicious, the compiler is going to block the compilation. Does not allow developer to compile their code. This is a feature of Rust. Rust has a very simple runtime so that the performance of Rust program is also very fast. Basically Rust takes the advantage you see, I mean the performance advantage. And the Rust design compiler checks to eliminate those memory bug you see those very important or very severe memory bugs you see. So Rust try to target the most severe problem in C program language. And many big companies such as Microsoft and Amazon, they began to use Rust to re-implement some key component originally in C program language. They tried to change those implementation, dual re-implementation using Rust with the goal that they can have the same performance, but they will have almost zero security vulnerability.

[00:15:48] JM: So tell me a little bit more about the applications that Go is a good fit for versus the applications that Rust is a good fit for?

[00:15:56] LS: Okay. Go is good for those server side application. We know that for server application, they need a tool. They need to process different requests sent to them. Usually when a request is allowed, the server application is going to create a new thread to process that request, to create a thread to process that request. This is why Go is very good. Go is designed for concurrent programming. And server-side applications need a lot of concurrent programming. This is why Go is suitable, very good and to build server-side application such as proxy. This is one potential application. The other one is database. This is another application.

And Go is also go down to implement resource RPC. For example there is a Go version of RPC library. Of course the most two famous application right now is Docker and Kubernetes. Basically those two are very popular in data center in modern data center. For Rust, Rust

basically, it is designed to replace C and C++. Rust is good to implement. So the low level and security critical application such as driver operating system or broader. And I think there is a container management tool designed by Amazon. It is implemented using Rust.

[00:17:40] JM: Let's talk a little bit about Rust. You've mentioned your work on static analysis in Go. What was your work in Rust?

[00:17:48] LS: We have one tool right now. Basically we build a GUI. You can consider it is a programming tool. So when Rust programmer, once they are writing Rust code, our tool can do some visualization. There is a very important concept in Rust called lifetime. Basically Rust do a strict compile time checking. The checking centers around two concept. One is ownership. The other one is laptop. So basically a value can only have one owner variable. The owner's lifetime earns the value is going to be dropped off read by the compiler.

We observe that sometimes developers do not have a good understanding of a variable's laptop. It can cause memory bug such as user for free. It can also cause concurrency bug. For example, in Rust, there is no unlock. It is very different from C and C++. The unlock is implicitly called. So basically when you call a log, the log function call is going to return a reference. And when the variable holding the reference, when that variable laptops ends, unlock is implicitly called by the compiler. We observe that because developer cannot understand where – Cannot understand the laptop. Sometimes they are going to held a log longer than their expectation.

Inside the critical section, the same log, it is possible to be acquired again causing double log data log. This is a very common bug in Rust. And we think those bugs are due to the reason developer cannot understand the laptop correctly. We build a GUI tool or IDE tool during when developers do the programming when they are writing Rust code. Our tool can help them visualize the lifetime scope for user selected variable so that developer can check whether or not our visualization matches their expectation. If it is not matched, probably developers need to consider whether or not they have made a mistake and whether or not a bug is introduced.

So basically we build a GUI tool. We can visualize the variables lifetime and help developer avoid bug during programming. This is what we have already done. Another ongoing project is that there is a technique called fuzzing. Fuzzing technique basically fuzzer technique generates a lot of random inputs to explore different code region our program to identify memory. Fuzzing technique is believed to be the most effective technique to detect or expose potential memory problem for C and C++ program. Right now we are trying to improve the fuzzing on Rust program.

The intuition is that Rust has already conducted a lot to compile time checking. A lot of memory bug that can happen in C and C++. Right now it cannot happen on Rust program. We are considering to use Rust language feature or language unique feature to improve the fuzzing technique. We believe there are a lot of code in Rust program, which has already been verified all completely inspected by the Rust compiler. We think the fuzzing technique do not need to spend a time on those code. If we force the fuzzing technique to ignore those codes, fuzzing performance or fuzzing effectiveness can be improved. This is an ongoing project we are doing right now.

[00:22:11] JM: So what has been your experience doing research in Rust?

[00:22:15] LS: It is kind of similar to my experience when doing research in Go. Basically we follow the almost the same methodology. We try to understand a real world above firstly. We collect a lot of bug from real world programs, Rust program. We do empirical study to do – we design a categorization and we inspect different code pattern and fix for each type of bug. This is our first step. We try to understand real world problem.

The second step is we build different techniques. And we think for a program, it has a left cycle developer. Need to implement some Rust program. The developer apply the Rust compiler to do inspection, to do the compilation. After compilation, developer can do testing. For example, they can use a static tool or dynamic tool to identify bug to verify the program and identify the bug. Then the program can be released to the user.

Rust data, a lot of work. They had a lot of language feature in the compiler stage. But there are multiple – There is a developing stage before the compilation and there is a testing stage after compilation. Basically our GUI tool is trying to optimize trying to help the development stage. It is before compilation. And our fuzzing technique, our normal fuzzing technique is trying to trying to help the testing stage. It is after compilation.

Our insight is that Rust did a very good work as a compeller. But developers still need to support before the compilation and after the compilation. So basically before the compilation, it is the development stage. After compilation, it is a testing stage. And our group, we have a finished project and our ongoing project targeting those two stage different from compilation.

[00:24:33] JM: Does the work and Rust have practical applications like the practical applications of Go?

[00:24:39] LS: Yes. Our work are working on your application. There are several OS implemented in Rust. One is it [inaudible 00:24:50]. You can consider it is a Rust version. Linux, this is one. The OS is called Redux. Another thing is there is a browser [inaudible 00:25:03] browser it is implemented by the Rust team. So basically it's vulnerable. Some component of this model is already replaced. The call of Firefox. Firefox is also a browser maintained and designed implemented by Mozilla. Rust is a language invented by Mozilla. And the Mozilla person, people think that Rust is much safer than C and C++. So several safety critical components in Firefox has already been used, has already been replaced at their Rust version.

Our bug, all the bug we studied come from those famous Rust project open source application, such as [inaudible 00:25:57] I just mentioned and Redux, the operating system. I think it is implemented by Stanford University, although it is not as popular as [inaudible 00:26:10]. But I can view that more and more people are trying to explore this new operating system.

[00:26:22] JM: Tell me more about how you structure your research projects when you're delving deep into how you can improve a programming language's systems.

[00:26:31] LS: Yeah. This is a very good question. We actually have a meeting with the Rust team this week and they asked us if we can change once thing or add one thing in Rust. What we are going to do? And I think it is very good. Our research draws the attention from the real language designer. I think it is very exciting.

Yeah, I think definitely, since we studied as a common mystic made by developer and we definitely think the program language designer should consider how to change their language or how to improve their language to help a programmer not make the same mistake again. One thing I'm thinking is that there are multiple or there are many, we call it the [inaudible 00:27:21]. So basically it is the non-circular bridge of broken bug we just discussed. There is one [inaudible 00:27:27] that guy with four messages from others, but actually no other [inaudible 00:27:35] going to send that message. So one single [inaudible 00:27:38] is blocked there forever. This is very common, a very common problem in Go language.

And the way we solved in Rust, there is a different design. For Rust, basically the sender and the receiver, both side has each laptop. When the sender ends its laptop, if there is a receiver, there is another thread trying to receive from the particular channel. The receiver is going to receive a panic. The rust we are going to trigger a panic for this particular case. But on goal side, a thread is going to wait there silently. There's no sender. We think that for Go, this problem is very difficult to debug or identify. But for Rust, there is explicit panic is going to be triggered. So developer can easily identify the problem.

And we think this experience in Rust can definitely help the language design in Go. And yeah, since we are comparing our study different program language, definitely some good feature, some advantage in one language can definitely help the other language.

[00:29:03] JM: What are the gaps in programming language quality when you look at Go and Rust? How could these languages be improved or the tooling around them?

[00:29:12] LS: I think for both of these two languages, they have very good language feature such as the concurrent programming support and the compile time check based on ownership

and the laptop. Both zones are definitely great language. But of course since both of them are very new and they are they can still being improved. And the one thing we definitely can do is our team can do is we can build more tool to have programmer in this two language and we can also help summarize the common mistake and discuss with the language sender to see whether or not new language feature can be introduced or some existing feature can be modified to help developer not make those mistakes anymore.

Yeah, I think of course there are many other languages appear or are invented these years and we can definitely look at the language feature in those new language and consider whether or not the language can help the problem in Go or in Rust.

[00:30:29] JM: Tell me more about what kinds of research projects you have planned for the near future.

[00:30:33] LS: Okay. Okay. For Go, basically we have built a static technique. Right now one of my students is trying to explain how to do a dynamic concurrency bug detection. Project for Go. Basically our idea is we observe, we try to apply the lifetime management in Rust and we borrow the idea and we try to apply this idea to Go. So basically there are auto channel and the each channel it could have a sender threat and a receiver threat and we try to dynamically monitor the reference of a channel and we apply the laptop monitoring and we can better detect a non-circular wave dialogue or we just discussed this is the dynamic bug detection. And after a bug is identified, it can still cause problem until it is fixed.

So we are also trying to build automated fixing or patching. So basically for identify the bug, we will take a try whether or not we can synthesize patch to eliminate those bugs. And we also need to do some testing to guarantee that the generated patch is cracked. It will not introduce new bugs. And these are two projects we are thinking on the goal side. For Rust, for us, basically what we want to do is another featuring rust is Rust has Rust diver code into safe code and unsaved code. Basically saved code, the compiler has do a lot of checking. On the safe code, a developer can do something dangerous. Developer need to take the responsibility to guarantee our save code is correct. And right now we are considering to do some

verification projects. We try to verify whether or not housing code is maintained or implemented correctly do not contain bug. This is one project. Another project is the fuzzing project we just discussed. And basically for the verification, we are trying to build a normal static technique. But for the fuzzing, we are trying to build a dynamic technique.

[00:33:07] JM: Can you give me more detail on the low-level primitives in Go and the low-level primitives in Rust and how those have led to their unique ecosystems?

[00:33:17] LS: I see. Basically Go supported channel, basically channel is a particular tab. It is the same as integer float a primitive tab in Go. So you can create a channel in Go like you create a variable, you declare variable. So channel has become a language feature. And as we just discussed, Go allow one threat to wait for a message from multiple threats. To achieve this purpose, to achieve this functionality, Go provided another keyword called a select. So basically if you – If a thread call select, it is going to block there. And the select has multiple keys. Each case is used to read for a message from another threat or another channel. So basically this is a primitive supported by Go to enable one slide to wait for multiple channel messages.

And we did a study and we found that channel is indeed widely used by a Go programmer. We count how many channels are using Go program. I think almost all Go program developer is going to use channel. And of course if they make some mistake, it will cause unique bugs in their program.

On the Rust side, the language of the grammar is kind of complex. And I read a lot of blog, and those blogs are saying that Rust is very difficult to learn, because the grammar is very complex. And for Rust, basically as we just discussed, Rust in falls each value only have one owner. And the Rust allow to borrow the ownership using reference. And the reference plus diver reference into a mutable reference and immutable reference. You can change the border value through mutable reference. But you cannot change the border value using immutable reference.

You must also do a lot of checking to guarantee that there is at most one mutable reference at any time. Rust will use compeller to guarantee this. If you violate this rule, the competitor is going to throughout an error. And Rust, in order to share a variable, you must provide a lot to – Actually it is a library, but those libraries are used by a lot commonly used. I think it is already part of the language. In order to immutably share a variable, Rust force the programmer to declare the variable with a library called ARC in order to both mutable share and immutable share an immutable share, you are the first programmer to declare a variable with both ARC and a Mutex. So basically Mutex need to – Mutex mutants can provide the mutual exclusion in order to change or modify a variable protected by Mutex. You need to call the log methods firstly and to get reference of the protected variable. Rust compiler is going to check to guarantee all the change modifications are conducted through the retained reference.

This is how the Rust compiler guarantee there is at the most one writer at any time. So basically leverage the mutual exclusion provided by Mutex. Those are the unique set I can think of inside Rust. And of course those features, those features because they are enforced by the compiler, they influence almost all Rust programmer.

[00:37:26] JM: Do you think go and rust have displaced C and C++ and other low-level programming languages successfully? Are there still applications of C and C++ that are necessary?

[00:37:40] LS: I think we definitely need a tool, we definitely need some type of tool to transfer to a new program language. We did an empirical study on Rust and we observed that there are a lot to last program. This still needed to depends on existing C and C++ code. People has working on C and C++ for so many years. There are also a legacy called very good existing code in C and C++. Those new languages definitely need a new IO to exist in the ecosystem. Although I think the new two languages are really great. But I think people still need a long time to fully rely on those two language.

Right now I think the ecosystem is kind of like for something that is good to do it in Go or in Rust, developer will do that. And for some legacy part, for some legacy code, if developers can

find some trusted C and C++ version, they kind of combine Go with C, C++ or Rust with C and C++. Combine different code together.

[00:38:59] JM: How does the compiler tool chain of Go and Rust respectively compare to other compiler tool chains like C++?

[00:39:08] LS: Go. Basically Google designed – google completely implemented our compiler for Go. But that compiler is very simple. It almost has no optimization, very simple compiler optimization. Go is featured with its runtime. Go's runtime dynamic part is very complex. But the compiler is very simple. For Rust, Rust to be used RVM as its backend. So basically Rust share the compiler backend as C and C++. What Rust did is Rust has a very good frontend. Rust did a lot of compiler time check as we just discussed based on ownership and the laptop. All those checking are conducted on the compiler frontend or the backend almost for the optimization. Those parts are used the C and C++ version. So basically Rust share something with C and C++ and it also adds some unique checking before those existing optimization.

[00:40:23] JM: If you had to build an operating system tomorrow, would you use Go or Rust?

[00:40:27] LS: Oh, I will use Rust. Because I think the performance is really important for an operating system. And I think Rust is good at this.

[00:40:40] JM: Did you say there have been a lot of actual production operating systems that have been written in Rust?

[00:40:46] LS: There are some. There are several operating system. But I'm not sure how many people are using them. It's kind of they are famous. A lot of people know that. What I can see is those operating system is not as popular as Linux or Ubuntu, which I see is maybe they will become popular in the future. But right now it is not something like Docker and Kubernetes. So it is not popular at that level. But definitely a lot of operating system researcher, they know there are some OS already implemented in Rust.

[00:41:34] JM: Can you say more about the kinds of deadlocks that your concurrency and memory bug detection tool can help detect?

[00:41:44] LS: Okay. So basically we build a technique to do static analysis to identify bugs caused by misused channel. So it's kind of more like more related to message person in Go. It is different from just traditional shared memory data log. So basically traditional one is more about channel memory, dialogue. So they misuse Mutex. But our technique try to identify dialogues that are caused by misused channel, misused message passing. This is different. And what we did is we leverage a technique called constraint solving. So previous consonant solving currently work on channel memory dialogue. That are caused by Mutex. Those are traditional technique. What we did is we extend or enhance this traditional technique to enable it unbox caused by misused channel, misuse message passing. And we remodel how channel works, and that channel has different behavior. When there is no buffer or when there is some buffer, the channel behavior is going to change. And the one buffer is for channel behavior is also going to change. So basically channel's behavior is much more complex than Mutex, which is modern in previous work.

We model the behavioral channel. We extend the current constituent solving algorithm or technique. This is the core of our static detection technique. Another thing we did, we contributed that. You know that Go applications like Docker, Kubernetes, they contain meanings lines of code. You need to solve one problem, which is how to scale. How to apply, how to enable our static analyzer to analyze a code base in such sites. So we also designed some algorithm to enable the scalability of our proposed technique.

[00:44:04] JM: Well, thanks for coming to the show. It's been a real pleasure.

[00:44:06] LS: Okay. Thank you so much.

[END]