EPISODE 1004

[INTRODUCTION]

[00:00:00] JM: Machine learning applications are widely deployed across the software industry. Most of these applications use supervised learning, a process in which labeled datasets are used to find correlations between the labels and the trends in the underlying data. But supervised learning is only one application of machine learning.

Another broad set of machine learning methods is described by the term reinforcement learning. Reinforcement learning involves an agent interacting with its environment. As the model interacts with the environment, it learns to make better decisions overtime based on a reward function. Newer AI applications will need to operate in increasingly dynamic environments and react to changes in those environments, which makes reinforcement learning a useful technique.

Reinforcement learning has several attributes that make it distinctly different than supervised learning from an engineering standpoint. Reinforcement learning relies on simulation and distributed training to rapidly examine how different model parameters could affect the performance of a model in different scenarios.

Ray is an open source project for distributed applications. Although Ray was designed with reinforcement learning in mind, the potential use cases go far beyond machine learning, and this project could be as applicable and broadly useful as distributed systems projects like Apache Spark and Kubernetes. Ray is a Project out of the Berkeley RISELab, which is the same place that birthed Spark, Mesos and Luxio.

The RISELab is led by Ion Stoica, a professor of computer science at Berkeley. He's also the cofounder of Anyscale, a company that was started to commercialize Ray by offering tools and services for enterprises looking to adapt Ray. Ion Stoica returns to the show to discuss reinforcement learning, distributed computing and the Ray Project.

You can find all of our previous episodes, including that previous episode with Ion Stoica, which was about serverless in the Software Daily website or the Software Daily mobile apps for iOS or Android. You can search for all of our past episodes, all 1,500+ of our episodes. You can look for technologies and companies that you're curious about. If there's a subject that you want to hear covered, you can feel free to leave a comment on this episode or send us a tweet @Software_Daily. We'd love to hear from you.

[SPONSOR MESSAGE]

[00:02:39] JM: Apache Cassandra is an open source distributed database that was first created to meet the scalability and availability needs of Facebook, Amazon and Google. In previous episodes of Software Engineering Daily we have covered Cassandra's architecture and its benefits, and we're happy to have Datastax, the largest contributed to the Cassandra project since day one as a sponsor of Software Engineering Daily.

Datastax provides Datastax enterprise, a powerful distribution of Cassandra created by the team that has contributed the most to Cassandra. Datastax enterprise enables teams to develop faster, scale further, achieve operational simplicity, ensure enterprise security and run mixed workloads that work with the latest graph, search and analytics technology all running across hybrid and multi-cloud infrastructure.

More than 400 companies including Cisco, Capital One, and eBay run Datastax to modernize their database infrastructure, improve scalability and security, and deliver on projects such as customer analytics, IoT and e-commerce. To learn more about Apache Cassandra and Datastax's enterprise, go to datastax.com/sedaily. That's Datastax with an X, D-A-T-A-S-T-A-X, @datastax.com/sedaily.

Thank you to Datastax for being a sponsor of Software Engineering Daily. It's a great honor to have Datastax as a sponsor, and you can go to datastax.com/sedaily to learn more.

[INTERVIEW]

[00:04:20] JM: Ion Stoica, welcome back to Software Engineering Daily.

[00:04:22] IS: Thank you. Happy to be here.

[00:04:24] JM: When I interact with an application that has machine learning, there's often a long delay between when I give that application input and when it takes that input into account, and I don't get the feeling that the systems that I interact with are continuously learning. Why is it hard to design machine learning applications that are continuously learning?

[00:04:45] IS: That's an excellent question, and there are several reasons for that. Let me start. The first one is that say you are going to get new data and you are going to learn a new model and you want to deploy that model because it's based on what you learn. But how do you know that model is good, right?

The operation process to deploy a model into production is not that easy. Typically what you want to do, you want to test to do a rollout, deployments, so you test your new model on a small number of users. If that is good, then you are going to increase a deployment. So all of this process takes time. That's one issue. The other issue is about how long actually is taking to train a new model. As you know, the neural network models are growing in size exponentially. The more parameters you have, the more weights you have, the more it takes to train.

Actually, the training time is super linear in the size of the model. That's another limitation. The other thing, the last thing is fundamentally building online, more real-time systems is hard. It's hard because, first of all, you want to provide the responsiveness. You want to provide the small latency, but then you want to provide also fault tolerance, because what happens when you have faults? You want to take into account strugglers, machine which kind of are slow for one reason or another. All of these reasons makes continuous learning really difficult.

[00:06:34] JM: As we build applications with tighter and tighter machine learning feedback loops that are responding to dynamic environments, reinforcement learning becomes more useful or more important. Can you explain why that is? What's the connection between reinforcement learning and continuously learning applications?

[00:06:54] IS: Yeah. Again, great question. A lot of progress which happened in the last decade in AI, machine learning, has been due to these two big paradigms, basically supervised and unsupervised learning. Supervised learning, when you hear, usually you hear about image classification and things like that use supervised learning. What supervised learning, you basically train a model by showing it inputs and outputs, correctly labeled inputs, right? You have a training stage. Then once you train the model, then you take the model, you deploy, say, in production, and then you see new inputs, like images and you are going to classify them, right?

Now, reinforcement learning, its approach, which is radically different. In reinforcement learning, you have an agent which continuously interact with the environment and through its interaction, it can change the environment, right? Again, supervised learning doesn't really interact with the environment. It doesn't really change the state of the environment. It's only observing it.

One example of reinforcement learning application will be an application which plays a game, like a chess game or Alpha Go, playing Go from Google. In that particular case we can think about the environment, it's a board with the position of the pieces, okay? This is what the agent you are training sees and you are training the agent to, based on the observation, in this case, of the board and the position of the pieces, to make a decision, to take an action. The action is to make a move on the board. By making the move, you change the state of the environment.

The goal of this move, it's to get the ultimate reward. In this case, to win the game, right? This is what it is. You can think about reinforcement learning like playing with the environment, observing the environment. Based on the observation, you take an action, you change the state of the environment, you observe again and you do it again. The main goal is to come up with an agent which maximizes some objective, the reward, winning the game.

When you do that, when you – For instance, you play a game. If you win, then what you do, you look back at all the action you made and you reinforce these actions. Meaning that next time when you are going to play, you are going to take this action with a higher probability, right? If you lose, you do the opposite. All these actions which you took, they're going to be a little bit less likely you are going to take next time when you are playing the game.

Now, some of the action, like some of the moves can be both part of the winning game or a losing game, but the hope is that the good actions you are going to see much more in a winning game than in a losing game. This is what is called reinforcement learning, because based on the outcome, you reinforce the good actions you made.

[00:10:07] JM: When we consider applications that are using reinforcement learning, we can also think about the importance of simulation, because if you're thinking about a chess game, chess, it's very easy to enumerate the different possible moves in any given scenario. So you can imagine the different scenarios and simulate how the game would look as those different decision trees might progress.

If we have all these simulations, it becomes easier to think about we could parallelize all of these different simulations occurring in order to come to a conclusion more rapidly. Can you describe the connection between reinforcement learning and simulation and parallelism?

[00:10:55] IS: Absolutely. Actually, reinforcement learning is a workload. It's a very complex workload, because you – First of all, this agent, it's learning what we call a policy, a model. You need to do training on that policy, right? The training itself can be done in parallel. It's a big model. In order to train fast, you are going to do that in parallel. Then once you have the model, you have to observe the environment and make a decision. You need to serve that model.

Finally, like you said, in many, many of these cases, you are going to resort to use simulations in order to simulate the environment to learn faster, right? If you play games, yeah, it's very easy. You are going to simulate the games. But the one other thing to consider is like in many other more practical applications, you do have simulators. In many industries, when people make decisions, very expensive decisions, they typically are simulators. If you want to figure out you are in, say, it's oil industry, you want to drill a hole in the ground, then that is very expensive. You're having to do some simulation before doing that.

Obviously, you have manufacturing when you build a car. Before you build a car, you kind of simulate it, right? Simulates how it's going this font to behave in a friction, with the air and so forth, things like that. Almost in all these fields you kind of have these simulators. You can actually use reinforcement learning for these practical applications just doing a much more

guided and much faster, much faster decisions based on using these simulations. Then the decision actually you can apply in the real-world.

Now, like you said, what is the connection between simulation and parallelism? Well, the more simulation you can do, the faster you're going to learn, so better decision you're going to make, right? Even simple things like a chess game or like Go, when you start to think about a few moves ahead, the number of possibilities increases total exponential, right? That's why the parallelism, if you can increase the level of parallelism, you can do more simulations.

You can go deeper in the, so to speak, serve space. What this means, that you are going to make better decisions. If I am playing chess and I can look four moves ahead, I can think much better decision if I can look only three moves ahead.

[00:13:34] JM: The motivation for this discussion about distributed reinforcement learning is that you've worked on Ray and Anyscale, which is a productization of Ray, and the underlying idea there is that the software systems we've built for supervised learning may not be a great fit for some of the reinforcement learning tasks. Part of that reason is because in supervised learning, the training and the serving process can be handled separately. But in reinforcement learning, the workloads of training and serving and simulation are all tightly coupled. There are probably some other architectural reasons why supervised learning is considerably different than reinforcement learning.

Describe the ways in which the tools that we have available, pre-Ray perhaps, are insufficient for reinforcement learning.

[00:14:31] IS: Again, great question, Jeff. If you look back, you ask the first – Your first question about was continuous learning, which is related with reinforcement learning and obviously online learning. In many of these systems that you need to do, you need to do a few things, right? You need to ingest the data and to process the data, featurize the data. Then you need to train a model. You need to serve a model and then go back.

Traditionally – And if you have reinforcement learning, you also need to do simulations. You need to do some data processing. You need to do some training. You need to do serving of the models in production and you need to do simulations, okay?

Traditionally, for different workloads, we had different systems. Especially things we wanted to scale up all these workloads. You want to scale up. Serving, you want to scale up, training and so forth. When you have different systems and you want to have one application, to build one application, to integrate, to stitch together all these systems, that's very difficult. First, development. You have to handle different systems with different APIs, possible different languages. That's one.

Then say you build your applications, you need to deploy it. Again, when you deploy it, you have to deploy different systems and to make sure that they work well together. The service really is a performance, because now if you have different systems, to move the data between the systems, you may need to go through different storage systems, different formats and so forth. This absolutely takes time.

All of these together – We know that. When you have multiple systems, it's going to be much harder not only to be build, but to operate and to make that system fast. Long time ago, we had this – Some favorite analogy, like it used to be long time ago before we have smartphones, we had a bunch of mobile devices for different functions. Obviously, you have a cellphone to make calls. Then you have a GPS for maps, for directions. Then you have some video playing game, mobile video playing game, Nintendo, or whatever. Then you have some camera, digital cameras. We have all of these things. But if you want to put together some application, take a picture, then you analyze the pictures, and then you important it in your game and things like that, it's very difficult.

Then when the smartphone arrives, iPhone and so forth, put all of these together and now it's much easier. You have only one device. It's supports all the workloads and it not only makes possible to build the application you already build better, but it enables new applications. That's kind of the analogy I like to think when we are talking about where we are today. We're building these end-to-end continuous learning, reinforcement learning application requires stitching together different systems, like for training, you are going to use TensorFlow distributed of

Horovod. For serving, you are going to use maybe – There are a variety of systems out there. Many of them are homegrown.

Then you train, it's not only training a model, but then you need to tune that model. High-parameter search, high-parameter turning, right? You need a system for that as well because you want to see, "Okay. I have these network architecture. It's kind of working well, but what if I change the number of layers? What if I change the numbers of neurons in each layer? What if I change the activation function formula? Things like that. So then it's a huge space you want to search to find the best model. For that alone, you have other systems, like SQL. Yeah, you have to put all of these together and it's incredibly difficult.

[SPONSOR MESSAGE]

[00:18:48] JM: Over the last few months, I've started hearing about Retool. Every business needs internal tools, but if we're being honest, I don't know of many engineers who really enjoy building internal tools. It can be hard to get engineering resources to build back-office applications and it's definitely hard to get engineers excited about maintaining those back-office applications. Companies like a Doordash, and Brex, and Amazon use Retool to build custom internal tools faster.

The idea is that internal tools mostly look the same. They're made out of tables, and dropdowns, and buttons, and text inputs. Retool gives you a drag-and-drop interface so engineers can build these internal UIs in hours, not days, and they can spend more time building features that customers will see. Retool connects to any database and API. For example, if you are pulling data from Postgres, you just write a SQL query. You drag a table on to the canvas.

If you want to try out Retool, you can go to retool.com/sedaily. That's R-E-T-O-O-L.com/sedaily, and you can even host Retool on-premise if you want to keep it ultra-secure. I've heard a lot of good things about Retool from engineers who I respect. So check it out at retool.com/sedaily.

[INTERVIEW CONTINUED]

[00:20:24] JM: The observation that there's a variety of different systems that are being put together to solve machine learning problems, that seems like it's generally true for machine learning problems. That doesn't seem like something that's relegated to reinforcement learning. Is there something that reinforcement learning that places such a constraint on these systems that we already have that makes it such that we cannot do the stitching together for reinforcement learning?

[00:20:49] IS: You already said that earlier on and we discussed it. With supervised learning, you can – Supervised learning application is more amenable to build and deploy it in different stages. You do training offline. You can use systems. Then you take the model, you put it in production, monitoring production, right? It's kind of coarse grain. You have in this stage, do training in this stage, you do serving and monitoring and things like that. You update your model sometimes every few days or even every few weeks.

But with reinforcement learning like we discussed, you need to train the model as new data arrives and as you are – You need to do simulations and you need to serving, because once you have to train a model, a policy, you need to serve it. It's like you see the environment changes, you take a new action based on the policy. You have all these workloads, it has to work together. That's what it makes much more difficult.

[00:21:55] JM: As we get into the programming paradigm of Ray, there are two kinds of computational tasks that Ray is made to address. Those are task parallel and actor-based computations. The task parallel ones are things that are – They're stateless. They can be parallelized and the actors are stateful computations. Can you give examples of each of these kinds of computations and how they might be useful in a machine learning application?

[00:22:27] IS: Absolutely. Let me take a step back with about task and actors. Fundamentally, in many programming languages, especially imperative programming languages, you have to construct when it comes to program, two main abstractions. One is functions. We have functions. Then you have classes or objects.

Tasks are basically in Ray and in other systems are basically executing that function remotely. An actor is instantiating a class or an object remotely. An actor like a small microservice. That's

why Ray is so general because it takes these two fundamental abstractions and gives the ability under the hood to this developer to execute them distributededly and in parallel. Now you ask about, “Okay, you have a task and actors and when we are going to use. Give examples about application for each of them.”

Whenever the task are best, for instance, when you do some data processing. You have some set of data. You want to say to process it, like you have an image. You want to do some image transformation. Then you run a function and maybe you can execute it where the data is stored and ingest some images and process them and then write the output and the function is gone. That we say is stateless, because every function after it operates, all its output is outside, is stored outside as a function.

Now, with actors, actor have the internal state. It’s like a class, like an object class. You are going to encapsulate the state there and then you are going to have the methods you are going to invoke on that object to change the state. Where do you use these actors? You use actors when you want performance on operating on a certain state. For instance, let’s say on one particular application, you want to train a model, and obviously today, most of the training will happen on the GPU, okay? If you want to use functions, this means that you are going – You do some training episode. You are going to have to read the data to the GPU. Train on it and get the results out once the training episode is done.

If I use actors, I can actually have the actor basically on the GPU and each method on invocation will be a training episode. Every time I’m going to basically – Maybe the first time I need to move the data to the GPU, but the next training episode, I’m not going to move any data. Instead, I can reinitialize, say, instance the weight. Reinitializing the weights on the GPU is much easier than ingesting, getting the weights from outside the GPU, moving on the GPU and then moving them out, right? That’s one.

Another thing is that which is a classic one is streaming. You do streaming, you get new data and if’s changing some internal state, then you don’t want, say, at every message to write the state outside on an external storage in order to have a stateless computation. Another example will be is very – State is games. The games, they have been taught on state.

If you think like a game, it's like a big actor. It's like you act, you click on the keyboard or whatever. These are the actions. They go. They change the state of the game. But the state is internal to that game. That's another example.

In summary, you are going to use actors when you want to really collocate the state with a computation for performance reasons. You are going to use stateless operators, in particular one, you want to process lots of data and you are going to execute that function. Actually, you can also send the function where the data is. Process that data and write the output and you are done.

[00:27:12] JM: Two components of the Ray architecture are a task scheduler and a metadata store. Can you describe these two components?

[00:27:21] IS: Yes. There are a few questions you need to answer when you build a system like Ray. Again, like with Ray, you want to support both actors and you support both tasks. One question when you are going to create, say, a new task or an actor, because you can execute them remotely, where are you going to execute them, right?

You want to execute them on a node, which has a required resources. For instance, if you have an actor which is doing some training, you really want to execute on a node which has as GPU. If you, like earlier on, like we discussed, if you want to execute a function, which is doing some data processing, you may want to schedule it on the node who has that data.

By the way, the problem is that you don't have only one entity or node, trying to schedule the task or actor it has created, but any node in the system can do that. Now, you have this kind of really complex scheduling problem in each in each distributed scheduling problem, each multiple nodes wants to issue [inaudible 00:28:36] and wants to schedule multiple tasks and actors. There can be many of those. The task can be very short. It can few tens of milliseconds or something like that. In a lot of system, you can have millions at the same time. You can have millions of tasks and actors being scheduled.

For that one, again, in many systems before, what you have the scheduler, it's kind of centralized. You have only one entity. Imagine that if I want to schedule something, I'm going to

send to this guy, say, it's called master or something like that, and ask that guy, master, to schedule on my behalf. That's an easier problem because the master will have more of a global state. There is only one guy to schedule the task or actors.

However, like we discussed, the problem with that is that doesn't scale. Now, you have to distribute the scheduling and this is what makes it difficult and is one of the innovation in Ray having the distributed scheduler where you allow every now to independently schedule and then there is a way in which you are going to allow mistakes to happen, say, for instance too many tasks are scheduled on a node and then to correct these mistakes by taking some of these tasks and moving to another node till everything is more or less load balanced. That was about scheduling.

About this metadata store, we call it global control store. It's, again, distributed system – Building a distributed system is hard especially when you want to be high-performance and fault tolerant. Let's take, there are two reasons for this control. First, when I schedule a task, that task create an object and that object can be used by other task or actor, okay? If I schedule a task who uses an object created by another task, I need to know where that object is. In Ray, things are very dynamically. Actually, it's possible to schedule a task would require an object, which has not been created yet.

[00:31:00] JM: Like a future.

[00:31:01] IS: Like a future. Exactly. Like a future. Absolutely. Now the question is that how do I know? How does this function know where to get the object from? Our answer to that to use this global control store whenever objects are created, you write in the global control store the metadata of the object about saying where the object has been created. This allows you to discover all the objects you need to perform the computation. You, in this case, being a function or an actor. That's one reason.

The other reason for a global control store is fault tolerance. Especially for stateless operators like functions, one way – And some systems, they do implement fault tolerance is basically by replaying the computation. If I lose a data – Take a step back. There are two general ways to handle fault tolerance. One, you are going to replicate the data. If you have a failure, you have a

replica still around. But that's again for – When you have large amounts of data and so forth, it's quite expensive.

Another way is basically using this lineage. In many system like Spark and so forth, they use lineage. With lineage, basically, instead of replicating the data, in some sense, you replicate the computation. If I lost some data, I am going to re-execute all these operation, all the tasks who created the data in the first place. But now, how do I know what functions were used to create a particular data item? How do I know this lineage for the data?

The answer here is that, well, you are going to store that in this metadata store, the global control store. If I ask for an object, and object is no longer around, then I can look at the metadata in this global control store. Then from that metadata, from that which maintains a lineage of the object I'm looking for, I'm going to now which function, which task I have to run in order to recreate the object. I run those. I recreate the object. I use the object.

[00:33:21] JM: Does the developer do the checkpointing for that lineage or does the system take care of that?

[00:33:27] IS: Excellent question. There are two abstraction again. It's task, stateless tasks. Then there are actors. For stateless tasks, the developer doesn't need to do anything. Because, again, they don't have internal state, and I have lineage, I can just rerun them.

For actors, things are a little bit more complicated, because the actor have internal state, right? For that, we have different ways to handle failures, but one, in which is a developer, will checkpoint periodically state in the actor. If the actor fails, then when actors is recreated, restarted, then it's going to restart from the previous checkpoint.

[00:34:15] JM: Implementation question. When you're building that storage system, can you take something off-the-shelf like etcd and build something over it, or do you have to build it entirely from scratch?

[00:34:29] IS: Actually, we always try to use whatever best systems are available there. You can take something like etcd. The one thing though is that you need to take something which is very

high-throughput and low latency, because if you want to write to this global control store, it cannot slow down too much the execution of the task or method of the actors.

Actually, in Ray right now, we are using Redis, the in-memory store. We have multiple shards. We can do many of the updates and reaching parallel. Obviously, you can use etcd to provide fault tolerance for those shards. However, I want to say even now, like obviously, Ray is a continuously evolving system. Even with these in-memory store – Again, it's not [inaudible 00:35:29] S3 or whatever. Everything is in-memory. Even with everything in-memory, things can be slow. Actually, we try now with new re-architecture of Ray to move as much data as possible, so to speak, to the workers out of the critical paths. We want to reduce the traffic between these workers, which handle task execution and actor executions and this global control store.

[00:36:03] JM: To talk about the applications of Ray, I want to get a little bit into the company that you're building around Ray, Anyscale. You've had some early enterprise users of Ray. You've had some interactions with them. Can you tell me about the interactions with those enterprise users how they're using Ray?

[00:36:24] IS: Absolutely. Ray at its core, it's a very general system. It's a very general system. While initially we started building a system for reinforcement learning, because the reinforcement learning like you discussed is such a general workload. Then it drove us to build a very general system. We have quite a few use cases. I'm going to enumerate a few of them.

One, there is a very large company, financial service company. Probably I cannot give the name, but it's using Ray to do some of these continuous learning you mentioned about. It's about like online learning. Their application is that when you open or when you look their mobile application, then they want to promote the products of their affiliates or recommend new services. Basically, the recommendation system.

The key there is that they wanted to update the model very fast because they want to learn very fast. For instance, if there is a new product, who is buying it? Because if you know who is buying it, I can immediately recommend to other people right away, so I can increase my revenues. Initially, the solution is quite simple. First of all, you get the logs from the users and then you train a model and you serve it, like we discussed, the same thing. They're updating this

model every day, but they saw that this is too slow. They want to go much faster. They use a best of breadth systems out there for featurization, for training, for serving. Again, like we discussed different systems.

This is a very, very engineering-savvy company and they went from one day to one hour. In that process, they improve the click through rate by 5%, which is pretty huge. Obviously, now they wanted to go even lower, right? It's like do it even faster.

Then when they started to look into Ray, because again using the existing multiple systems to stitch them together to provide an end-to-end online learning solution was too slow, was too complex. They use Ray, because with Ray, you can do featurization, you can do training, you can do serving in one system, on one platform. The data is there. It's memory. You don't need to move it around and so forth. They reduce it from one hour to 5 minutes, and they got another 1% improving click through rate. That's one example.

There are companies who are doing in financial banks, financial forecast and so forth. They use reinforcement learning to do it, and these are large companies, large enterprise companies. Very large. There are companies also, they're using for say fraud detection. In particular, it's for money laundering. With money laundering, every transaction, money transactions, it's an edge in the transaction graph. Typically, the way people detect money laundering is when they have a cycle in their graph.

Ideally what you want, for each transaction, you want to modify the graph and immediately look at it and see whether this is a money laundering. You suspect this to be a money laundering activity before you make the decision to accept the charge. You want to do this pretty much in real-time. That's another application.

Then there are startups which are doing – Are building their infrastructure on top of Ray. In particular, there are two startups I could mention who builds their infrastructure on Ray and RLlib, like Bonsai. It's a startup from Berkley who was actually acquired last year by Microsoft, and Pathmind. Also, they're building their infrastructure on top of Ray and RLlib. Yeah, I think obviously there are many more, but this probably give you a taste of what people are doing with Ray.

[SPONSOR MESSAGE]

[00:40:46] JM: If you are a SaaS or software vendor looking to modernize your application distribution to gain more enterprise adoption, checkout replicated.com. Replicated provides tools to deliver your Kubernetes-based application to enterprise customers as a modern on-prem private instance. That means your customers will be able to install and update your application just about anywhere.

Bare metal servers in a cloud VPC, GovCloud and their own Kubernetes cluster, vSphere. This is a secure way the your customers can use your application without ever having to send data outside of their control. Instead of your customer sending their data to you, you send your application to your customer.

Now, this might sound difficult and maybe you're not used to it because you're a SaaS vendor. You're a software vendor, but Replicated promises that recent advancements from tools like Kubernetes make it far easier than before, and the Replicated tools can help vendors operationalize and scale this process.

The Replicated tools are already trusted by noteworthy customers like HashiCorp, CircleCI, Sneak and many others. As a result, over 45 of the Fortune 100 already have an application deployed via Replicated in their infrastructure. That's a strong sign of adaption.

Go to replicated.com for a 30-day trial of the full Replicated platform. You can also listen to an interview with Grant Miller, the CEO of Replicated, that we did a while ago.

Thank you to Replicated for being a sponsor of Software Engineering Daily, and you can check it out for yourself at replicated.com and get a free 30-day trial.

[INTERVIEW CONTINUED]

[00:42:54] JM: Here you talk about Ray as a general purpose distributed computing framework. It makes me think of it in terms of some of the distributed computing frameworks that have come

to market recently, and start contrast, is something like Kubernetes where Kubernetes is a distributed framework for actual traditional application services, fully-fledged running services that are just distributed on to containers. Then another distributed computing framework that comes to mind is the serverless idea, where the atomic unit of execution is much smaller, much more fine-grained.

If I think about those two kinds of systems, Ray seems a little bit closer to the serverless paradigm, because you are encouraged by the programming API to think of your entire computation as these tasks or these actors and the system takes care of scheduling those in an intelligent fashion.

[00:43:56] IS: Absolutely. You are totally correct. Kubernetes, it's orchestration service, or resource orchestration. If you think about Kubernetes, targeted DevOps. This is what they target. Ray, like serverless, they target developers, programmers. You want to write code, they provide you the API and everything to write code and forget hopefully about everything what happens under the hood, right? Yes, it's much closer to serverless. Actually, the tasks in Ray, they're very similar with functions. But in addition, Ray has also actors. Again, if you – Now, the analogy point, it's like actors. You can think of actors like some microservices. In some sense, it's a very high-level and it's not entirely accurate, but maybe helps. It's like think about Ray that marries serverless with microservices.

[00:44:53] JM: Right. Those use cases that you described still sound like machine learning use cases. I mean, I guess the fraud detection one is a little bit less of directly a machine learning use case, but do you have an idea of how the system might be used for more general tasks?

[00:45:16] IS: Absolutely. Actually, we see that as well. Actually we have said it. We see a lot of [inaudible 00:45:20] of users which have the following problem. As you have said, right now that Ray provides a Python bindings, Python API. We have a lot of people which are facing the following the problem. I have an application, which is doing whatever, maybe machine learning, maybe some computations, whatever. I have more data or I need to do more computation, so I want to scale it, right? It's Python application. How am I going to do it?

I mean, you can do it. You can build your own system. You can maybe use Kubernetes plus GRPC and so forth. Maybe you can use MPI or something like that, but it's hard. Ray has a very, very simple API. At the core, there are only – It depends how you count, 5, 6, functions. That's all. It's very easy to learn. We have a lot of people who use Ray to scale their existing application. The cool thing about it, it doesn't – In most of the cases, it doesn't require to rewrite your application. That's why we see more and more people using Ray to scale their existing Python applications.

[00:46:39] JM: Let's zoom out a little bit and we'll begin to wrap up. My sense is that – Well, I mean if you ask a lot of people, they'll tell you, "Whatever we develop in industry, or in the open, in academia, it's great, but ultimately it's 10 years behind whatever Google is developing inside DeepMind or whatever, deep inside the bowels of Google. Do you have a sense whether that's true? Inside Google or Facebook, are they just light years ahead of what we're doing? Is what we're doing with Ray or whatever, other cutting edge we're developing, is it child's play compared to what they're doing at Google or is it just in a different domain? Is it just slightly different?

[00:47:22] IS: Yeah, that's a great question, and I have other hats, as I got any hat. I think that there are a few answers to that question. As a high-level, TLDR, I'm an optimist. The answer – One answer to that is like factual. Actually, we at Berkley, where I'm from, we have these labs and we build actually successful systems and open source systems before, Apache Spark for instance. I've been involved also in Apache Spark, and Mesos. But I think that if you look forward, yes, I think that Google is doing great things. There is no question about that. Maybe Facebook is doing great things. No question about that, or Microsoft, or Amazon.

But I do think that when it comes to also machine learning platforms, eventually a lot of platforms, almost every company, these companies, they have their own machine learning platforms, like [inaudible 00:48:22] Michelangelo, obviously Google has this, so on and so forth. But they're very integrating their existing infrastructures, very integrated. It's very hard to take those out and abstract them away, especially because there are many workloads, there are many different systems. It's very hard.

I think that here, I think it's a great opportunity and it will be a huge value to have some systems which can be general supporting machine learning, scalable machine learning applications, reinforcement learning applications which can be used by everyone. Then if you have that, what you are going to get, you are going to get an almost hopefully network effect, right? A lot of innovation, right?

Let's not forget, the point that not all hope is lost for people outside Google and Facebook. Think about a lot of people from our audience, they are too young to remember that. I'm not. But think about Linux. When Linux was developed, the question was what is – It makes no sense to create a new operating system, because Microsoft has 90% of the market and everyone is choosing Microsoft. Then, yes. Also Apple then. Why would you even want to create a new operating system? What is the chance? You stand no chance. It's not only Microsoft has a huge army of engineering doing that, but they have also great applications. It's not only the operating system.

There are all the application or the entire ecosystem. Look at where we are today. Certainly, things are quite different at the moment. Yeah, I'm really hopeful that we are living in a very dynamic world. Things are moving faster and faster. They are changing faster and faster. Then when such kind of work, do you have more and more opportunities?

[00:50:18] JM: Well said. As we wrap up, last time we talked, you mentioned Ray a little bit near the end of the conversation and I regret not taking a look at it sooner. I'm hoping to make up for my mistake. Now I want to ask you once again, what are the projects that brewing in the lab? What are the other paradigms that we haven't explored in this conversation that perhaps the listeners should take a look at and I should take a look at?

[00:50:46] IS: Clearly, one, and we touched a little bit over those, is like Ray and it's a core system. It's a very general system, but then there are a large number of libraries being built on top of it, like reinforcement learning, like [inaudible 00:50:58] parameter, high parameter tuning. We are working at a library to provide serving and many more. Just in that, so to speak, in the ecosystem, there are a lot of interesting things happened. We are also very happy to see more and more contributors from outside the lab and from industry actually. The vast majority of contributors are from outside Berkley.

The other thing in the lab I would mention, there are two, maybe – Again, it's like I don't want to pick, but something – It's hard to pick. There are a lot of great projects. But one maybe I want to mention, ones I want to mention is about learning on confidential data. We call this cooperative learning. Basically, the problem here is you have multiple organizations which want to cooperate for some mutually beneficial outcome. For instance, you have multiple banks. We want to cooperate to learn better fraud detection models, to detect fraud detection. Obviously, these banks are the same time competitors, so they cannot share the data with each other.

The question is how can you learn a model on all these disparate datasets without leaking the information from one bank to another? That's one example. I think the other area we are looking quite a bit is about program synthesis. How you are going to use machine learning in general not only end-to-end, but also to generate, to synthesize, say, a set of rules, some programs which will solve your problem? We have for instance some work in which you, for instance, for packet classification in networks. It's a very well-studied problem. You want to – One way to think about using machine learning is to do end-to-end. You get a packet. You look at the packet. Have that classified. What to do with it? Drop it? Forward to [inaudible 00:53:01] box or things like that.

But another way what we do is like instead of doing end-to-end, we use it to generate a decision tree. The cool thing about that because the decision tree or these like synthesized set of rules or programs, they're explainable. They're interpretable. You know that one of the biggest challenges with neural networks is hard to interpret it, it's hard to explain why it made a particular decision. That's one exciting avenue. I mean, generally, using machine learning to improve systems. To do better query optimizations, to optimize compilers better, the code of the compiler is better. Yeah. These are some of the things we are really excited about within the lab at Berkley.

[00:53:47] JM: Ion Stoica, thanks for coming back on.

[00:53:48] IS: Thank you. Thanks for having me.

[END OF INTERVIEW]

[00:53:59] JM: You probably do not enjoy searching for a job. Engineers don't like sacrificing their time to do phone screens, and we don't like doing whiteboard problems and working on tedious take home projects. Everyone knows the software hiring process is not perfect. But what's the alternative? Triplebyte is the alternative.

Triplebyte is a platform for finding a great software job faster. Triplebyte works with 400+ tech companies, including Dropbox, Adobe, Coursera and Cruise Automation. Triplebyte improves the hiring process by saving you time and fast-tracking you to final interviews. At triplebyte.com/sedaily, you can start your process by taking a quiz, and after the quiz you get interviewed by Triplebyte if you pass that quiz. If you pass that interview, you make it straight to multiple onsite interviews. If you take a job, you get an additional $1,000 signing bonus from Triplebyte because you use the link triplebyte.com/sedaily.

That $1,000 is nice, but you might be making much more since those multiple onsite interviews would put you in a great position to potentially get multiple offers, and then you could figure out what your salary actually should be. Triplebyte does not look at candidate's backgrounds, like resumes and where they've worked and where they went to school. Triplebyte only cares about whether someone can code. So I'm a huge fan of that aspect of their model. This means that they work with lots of people from nontraditional and unusual backgrounds.

To get started, just go to triplebyte.com/sedaily and take a quiz to get started. There's very little risk and you might find yourself in a great position getting multiple onsite interviews from just one quiz and a Triplebyte interview. Go to triplebyte.com/sedaily to try it out.

Thank you to Triplebyte.

[END]