EPISODE 787

[INTRODUCTION]

[00:00:00] JM: Google's Borg system is a cluster manager that powers the applications running across Google's massive infrastructure. Borg provided inspiration for open source tools like Apache Mesos and Kubernetes. Over the last decade, some of the largest new technology companies have built their own systems that fulfill the roles of cluster management and resource scheduling. Netflix, Twitter and Facebook have all spoken about their internal projects to make distributed systems resource allocation more economical. These companies find themselves continually reinventing scheduling and orchestration. With inspiration from Google Borg and their own internal experiences running large numbers of containers and virtual machines.

Uber's engineering team has built a cluster scheduler called Peloton. Peloton is based on Apache Mesos and is architected to handle a wide range of workloads. Data science jobs like Hadoop MapReduce, long running services such as a ridesharing marketplace service, monitoring daemons, like Uber's M3 collector and database services such as MySQL.

Min Cai and Mayank Bansal are engineers at Uber who work on Peloton. When they set out to create Peloton, they looked at the existing schedulers in the ecosystem, including Kubernetes, Mesos, Hadoop's Yarn System, and Borg itself. Bot and Min and Mayank join the show today to give a brief history of distributed systems schedulers and to discuss their work on Peloton. They've been working in the world of distributed system schedulers for many years including experiences building core Hadoop infrastructure and virtual machine schedulers at VMWare.

Before we get started, I want to mention that the company that I'm working on called FindCollabs is holding a hackathon. FindCollabs is a place to find collaborators for your projects or to find projects that other people have posted that you want to work on them with. You can go to findcollabs.com/hackathon to find out more, or you can listen back to the episode I did a couple of weeks ago called Find Collabs.

[SPONSOR MESSAGE]

[00:02:27] JM: Today's show is sponsored by Datadog, a monitoring and analytics platform that integrates with more than 250 technologies, including AWS, Kubernetes and Lambda. Datadog unites metrics, traces and logs in one platform so that you can get full visibility into your infrastructure in your applications. Checkout new features like trace search and analytics for rapid insights into high-cardinality data; and Watchdog, an auto detection engine that alerts you to perform its anomalies your applications.

Datadog makes it easy for teams to monitor every layer of their stack in one place, but don't take our word for it, you can start a free trial today and Datadog will send you a t-shirt for free at softwareengineeringdaily.com/datadog. To get that t-shirt and your free Datadog trial, go to softwareengineeringdaily.com/datadog.

[INTERVIEW]

[00:03:33] JM: Min Cai, you are a senior staff engineer at Uber; and Mayank Bansal, you're a staff engineer at Uber. Guys, welcome to Software Engineering Daily. Thanks for coming on the show.

[00:03:41] MB: Thank you. Thank you for inviting us.

[00:03:43] JM: So you're the developers of the Peloton resource scheduler from Uber. Let's start off with some basics. What is a resource scheduler?

[00:03:54] MB: So resource scheduler is something which we can use to orchestrate all the workloads to different machines and different set of clusters, and we do it because we wanted to run all the workloads in the isolations, in the containers, and we do resource our scheduling that which machines have the loads to run those jobs or those containers. That's what the resource scheduler is, is pretty much orchestrate all kind of workloads, and Peloton is pretty much orchestrating batch workloads, stateless workloads, stateful workloads. So this is how resource scheduler works.

[00:04:33] JM: Every operating system has a scheduler. My phone has a scheduler. My laptop has a scheduler, but we're talking here about a larger scale type of scheduler, a scheduler that can schedule resources across multiple machines. How do the paradigms between a single node scheduler, like the one on my laptop, how does that compare to a multi-node scheduler, like what goes on in a data center?

[00:04:59] MB: So single node scheduler is pretty much responsible for that single machine, right? So let's say if I have a laptop and if I have a scheduler, so I can use a single node scheduler to run some of the workloads on that machine only, correct? That's a single point of failure. If that machine goes down, that scheduler is down, and then we need to have – Again, that scheduler will bootstrap, and then again pick up the workloads.

In a multi-node scheduler, let's say if I have a hundred of thousand machines and one scheduler which is managing all those thousand machines. First of all, it needs to be very much scalable. It needs to be able to scale, manage, compute resources of those thousands or hundreds of thousands of machines, correct? Then try to find out what is the need of each workloads and then place those workloads on an appropriate machine out of those thousand machine, because those thousand machines need only not to be similar. So they may be are different attributes. They may have different resource requirements and they may have a different profile.

To match those profile with these different kind of workloads and what is the optimal sweet spot for those workloads to run on those machine, that's how this multiple nodes scheduler works. That's very important, because in all the big companies, we have thousands of machine which runs on a scheduler and then how you optimally run all the workloads as well as how you do all kind of high-availability on those schedulers. Because these workloads cannot – Let's say the scheduler goes down, the machines goes down, you need to be very much fault tolerant. You need to be very much highly available. So this is the difference between the single node scheduler and multi-node scheduler.

Do you want to add anything, Min?

[00:06:46] MC: Yeah. I think that's pretty much well said. So typically, like a single node scheduler for [inaudible 00:06:51], we have CPU schedulers. So those would decide at any

given time how much CPU shares we'll give to a given a process, which is like – But there's not really a placement initially you have to do. Even if there is some placement initially if you want to ping a CPU to a sudden cause, those will be very small scale. But as Mayank said, at Uber we're targeting for a cluster who is close to 10,000 nodes. Those are nontrivial amount of resources to manage as well as a nontrivial amount of workloads. We are dealing with close to like a million containers most of the time to be placed across those 10,000 to 25,000 nodes.

[00:07:34] JM: Right. Is this what makes the Uber case unique, because there've been so many distributed system scheduling tools in the past. There have been – What? Netflix built with Titus. You've just Mesos itself. You've got Kubernetes. All these different things that have been built in the past, and you guys set out to build your own, Peloton. So what is so unique about Uber that made you need to build a new scheduler?

[00:08:02] MC: I think it's not about unique of Uber. Uber has some kind of use case compared to many other companies. I think it's pretty much same requirement for many companies. By the way, this is not like never been done before. Actually, Google has this already which is called a borg, which is Google [inaudible 00:08:19] and Facebook has something called [inaudible 00:08:21], but they actually have the same vision as us as well, like collocating different workloads.

Any company like Uber when you reach a certain scale, you would like to automize the resource utilization of your computer infrastructure, because that's a nontrivial amount of investment. The unique thing about Peloton is we are probably the first one in the open source world which is trying to do collocation of mixed workload with the target objective to improve the utilization of the clusters in terms of collocating batch and [inaudible 00:08:57] workload. So as I said, like Kubernetes is doing the same thing, but Kubernetes is right now [inaudible 00:09:03] scale they are more for the cloud, not for big on-premise data centers. Also there are some other placement strategies we have in Peloton, which has some unique advantage compared to features compared to Kubernetes. We actually had a tech talk with Mayank with the team at KubeCon last year talk about the comparison between Peloton and Kubernetes.

[00:09:27] MB: So let me a little bit add to what Min also said, right? So when we started Peloton, we looked at what is the current paradigm of all the schedulers we have. We looked at

YARN. We looked at Kubernetes at that time and we looked at Mesos Aurora and all those workloads, all those schedulers, and we found out – Me and Min actually did a lot of investigation before we actually proposed this project. We found out that there is a gap in the industry. Every company right now or most of the bigger companies right now have these thousands of machine clusters of Hadoop which runs batch jobs and thousands of machines which are stateless clusters, which is mostly on meso at that time.

We found out they cannot share resources between each other, and there was no scheduler at that time and it's true for now in open source that which can merge all the workloads together. Then we thought we should be able to write and then we kind of tried one scheduler on top of other scheduler, which didn't work out because we wanted to solve the efficiency problem in the system. However by running one scheduler on top of another scheduler still creates the partitioning into the clusters, which is not getting the results which we want.

So we thought we should have one compute engine for all kind of workloads, whether it's batch workloads, whether it's Hadoop workloads, whether it's stateless workloads, or stateful workloads. If we can run together everything and if we can optimally place all these workloads together on the single machine and the cluster of machines, then we can get the efficiency out of the system, and that's where the Peloton born.

Currently as Min said, we have no other scheduler other than YARN is good at scheduling batch workloads. Kubernetes is doing some of the batch scheduling, but is not that good. We have the talk into the KubeCon and then we presented the numbers to the community as well, and Mesos Aurora is good for stateless, but not Batch. So there was not something there to do it. So that's the reason me and Min started Peloton.

[00:11:31] JM: Yeah. This difference in workload characteristics across different types of jobs, I want to give more color to what this means. So when you say a batch job, that might be like I want to run a job that's going to ETL or it's going to take data from one place and put it into another. It's a ton of data. So it's going to take some time. We need to spin up some machine to do this work, and then they need to spin down once they're done. That's just a batch workload.

You've also got services that need to be up for a long period of time and serve requests. Maybe they are stateful services like a database service, like a Cassandra or a PostgreS. Maybe they are stateless services, like you just make a request and you get a response and it's not doing any database workloads. You could also have monitoring workloads, where you've got some container agent that's just sitting there and monitoring some other service. Then you've got this pool of resources that any time a user, an engineer, needs to spin up a new service, that service is going to pull from this pool of resources. So this comes out to being a very complicated problem. It sounds like a knapsack problem or a stable matching kind of problem. Can you just give us a little bit more flavor on what is the variety of workload types? Why do these workload types differ? Why can't we just throw any kind of compute resource at any kind of workload?

[00:13:03] MC: That's a good question. So as you said, like for any big companies like Uber, you naturally have – We kind of categorize them into four type of workload as you have just said. They are the batch workload, which is a [inaudible 00:13:17]. So you can think of like the Spark jobs, Tensorflow jobs, any like big data jobs. The interesting part of that is those jobs normally are not that latency sensitive as you just said.

Then another category jobs, which is microservices, which are latency sensitive. What we found out is it's very – By the way, there's one big interesting part of how to improve the utilization of a cluster that only take an analogy to get there is called over commit, over subscription.

But it's very difficult to over subscribe latency sensitive services, because you have to provision – You have to always over provision your microservices to the peak so you can have guaranteed response time. For example, at Uber we have a service called marketplace services, and whenever you request an Uber ride on your phone, you're going to have [inaudible 00:14:18] service. The service had to be response very fast and you have to always over provision your computer resource. So make sure they always have enough resources to run.

But that means in the off-peak time, then your utilization will be low. Batch workload is actually a very good category of workload to mix in in the off-peak time when the time the online service is not busy. So you can push your computer utilization high.

[00:14:53] JM: So what you're saying there with the peak – And I remember this when I was looking at a Netflix stuff. Netflix users are always watching at like 6 or 8 PM. They're watching tons of movies. Then at 8 in the morning everybody is going to work, and so you can run batch jobs at 8 in the morning to do data science, because you don't have to allocate those same services to be running services to serve movies to people.

[00:15:20] MC: Yeah. That's exactly the motivation. So that's why collocation of mixed workload is very important for improving cluster utilization. Then another aspect is it's going to reduce operation overhead as well, because otherwise you don't have to have like – Typically within a big company like Uber, you will have one team for one type of workload, multiple team for one type of workload. For example, you have Spark team, you have Tensorflow team. If every team need to manage their own cluster, then there are lots of operation overhead to provision in managing those physical machines as well.

[00:15:56] MB: So I will more add color to the different kind of workloads. One is the batch workloads, which is as Min said, which completes – Multiple batch workloads are like the Hadoop workloads, Spark workloads, all the machine learning workloads and all the deep learning workloads. All these workloads has different categories, which is training and then serving. So all these trainings and offline training can be done at the non-peak hours as we said, and then that's how you increase the utilization of this combined cluster, and that's where this power of Peloton or the clusters which can schedule all kind of workloads together.

Similarly, stateless services, as Min said, we have [inaudible 00:16:36] sensitive services, which needs to be up all time and can take peak workloads at any time. But most of the time those are not at the peak. So we can use those resources to run all these offline training jobs or the batch jobs. Then there is another kind of workload, which is stateful, which is very much sensitive to the data and which has a different categorization. It's very sensitive to the data. So all the data needs to be present. So we cannot move machines in the stateful workloads. So that's another category. It's provision for the peaks as well as it's sensitive to the data. It has to be stickiness on the machines. So this is another kind of workload.

Then there is a monitoring workloads or daemons workload, which needs to run all the machines, which is not as much sensitive, but it's important to run on each machine, which is

ensuring the health of the cluster. So all these kind of workloads, if we can run together, we can get the efficiency from the system by placing them at the time when there is no peak or they're placing them at the time when there is a peak. So doing these decisions intelligently will help improving the cluster utilization as well as many teams, as Min said, there is no operational overhead. They don't need to maintain their own clusters. They don't need to maintain provision for the peaks for each cluster, because we don't want the static partitioning in each organization and each cluster. We want one big compute stack where everybody can come and run.

So we want actually people not think about the machines. We want people to think about the containers. We want people to think about, "I want 5 CPU." That's how it. How and where I will run it? That doesn't matter. So that is where we are going from Peloton perspective.

[SPONSOR MESSAGE]

[00:18:33] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It's open source and free to use, and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plug-ins. Use the value stream map to visualize your end-to-end workflow, and if you use Kubernetes, GoCD is a natural fit to add continuous delivery to your project.

With GoCD running on Kubernetes, you define your build workflow and let GoCD provision and scale your infrastructure on-the-fly. GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn about how you can get started. GoCD was built with the learnings of the ThoughtWorks engineering team who have talked about building the product in previous episodes of Software Engineering Daily, and it's great to see the continued progress on GoCD with the new Kubernetes integrations. You can check it out for yourself at gocd.org/sedaily.

Thank you so much to ThoughtWorks for being a longtime sponsor of Software Engineering Daily. We are proud to have ThoughtWorks and GoCD as sponsors of the show.

[INTERVIEW CONTINUED]

[00:20:03] JM: So now I want to talk about other schedulers, other orchestration tools. I want to go through some of the research that you guys reviewed when you were scoping out Peloton, because I really want to drive home the fact that this was not like a walk in the park you decided, "Let's just build an Uber scheduler," because why not? You guys are very experienced. Min, you worked at VMWare for 8 years I think, and Mayank you worked at Yahoo for many years. You worked on the Uzi Project. You've worked on core Hadoop infrastructure. So this was not like just a side project. Uber had a very significant need here, perhaps a need that is represented more broadly in the industry by other large companies that have large data center, large scheduling issues.

So when you were sitting down and reviewing the literature that was out there, you obviously have the Borg project from Google. Google has published a couple of Borg papers. You've got Kubernetes, which was based on Borg, but is not as mature as Borg, but Kubernetes actually is open source. Borg is not open source. You've got the Mesos project. You've got Hadoop's YARN system.

So when you were doing that initial research and you're looking at these different systems, and you already said that you basically figured out, "Okay. Well, none of these were really built for workloads that can go on to a data center really large scale infrastructure," and Uber does run its own data centers, right?

So what you find in that initial research, like you said you liked the approach of Borg, but Borg is not open source. What did you like about Borg?

[00:21:38] MC: One thing I think Borg has been done very nicely is kind of mixing the collocating or co-scheduling a mixed workloads. That's kind of – We have lots of people from Google today working at Uber. So the experience was Borg as being very nice, because they been set up that's very upfront. Meaning that Borg has been invented at Google [inaudible 00:22:01] only and then all the workload [inaudible 00:22:03] to only running on Borg. Very few people in Google can get physical hardware, and they have very large scale too.

Currently, from the Borg paper, they have very high utilization, which much higher than we have today at Uber, which is [inaudible 00:22:20] try to push for towards but without collocating or co-scheduling mixed workload. We basically figured out we cannot really do that.

[00:22:28] JM: How do you measure utilization?

[00:22:30] MC: So cluster wide utilization is actually very tricky. Basically, you can measure like individual hosts then do average across all the machines in the cluster and then do P99s, P95 of individual. Could have be like over like three days, seven days or monthly. There are lots of like dimensions in the metrics.

In short, I started the Mesos project at Uber as well. Basically, back then like three years ago, we adapt Mesos and plus the scheduler from Twitter, which called Aurora. We're running that for all the Uber microservices. But that's actually much better than have aesthetically designing a set of machines for one step for each service. But even with Mesos and Aurora, we cannot push the utilization to be too high. So that's still a much lower number compared to what Borg has showed in the paper. That's why we have to – Then we also invest locally into can we just use Aurora to run batch workload? We did some benchmark [inaudible 00:23:31] like faraway from what YARN can do today. So that's why we have to invent Peloton.

So back to your question about Borg. I think the nice thing about Borg is it's basically – It's designed for a mixed workload to begin with I guess from the paper. What did we saw? [inaudible 00:23:49] is there, and we didn't really have any available open source solution, which kind of equivalent to Borg.

[00:23:56] MB: So I will add a little bit here. So when we were starting Peloton, we looked at YARN and we thought. We looked at what is the YARN capabilities at that time, and that time in fact right now also it's not good for scheduling all the stateless workload. It's not good at scheduling doing upgrade workflows and all that stuff, which is a need of stateless services or are latency sensitive services. Those are missing from the YARN.

We looked at Aurora, Mesos, those were good at [inaudible 00:24:27] all these primitives of the stateless services. However, are not good for the batch workloads. They were not able to

handle the scale which YARN could handle. We looked at Kubernetes, it was starting at that time and it was not as much popular as it is right now. However, still right now there is a very big gap of running batch workloads. It's good for running stateless jobs and the deployments but it's not good for the batch.

So we could get whatever information is available from Google –

[00:24:55] JM: By the way, where does Kubernetes fall over with the batch workloads?

[00:24:59] MB: So we did some experiments, and then there are internal choke points at Kubernetes right now which is I think we talked in our Peloton KubeCon talk also where we identified some of the choke points. Those choke points are like scheduling, throttling and etcd, how Kubernetes use etcd's. So all these combinations actually have – We cannot scale Kubernetes.

So let's say we tested like for five 50,000 task in one job, or 100,000 task in one job, which Kubernetes could not scale. However, we have those numbers in the slides that Peloton can scale. So these are the different schedulers we looked at it. We had all the literature from Borg, which is published outside. We don't have any other information. We might not be just – So we thought maybe out of the literature which we had, it's interesting enough for us to go in that direction to get those efficiency into the system. That's when we decided, "Okay. I had a lot of experience in YARN, because I've been working on YARN for a long time. I've been –"

[00:26:02] JM: YARN. This is the scheduler from Hadoop.

[00:26:04] MB: From Hadoop. Yes, exactly. So I'm one of the Hadoop committer too. So I looked at the – I was working on YARN earlier, then I took those learnings and I took – And Min got all the learnings from Mesos and Aurora, which he did. Then we kind of start, "Okay. This is something which we need to build for Uber specifically," and that was a gap in the industry too. So we probably wanted to see if we can fill that gap as well.

[00:26:33] JM: Yeah. Now, one thing I was looking at with Peloton is that there's something called a hierarchical max/min fairness model in the scheduling system. So I know that when

you're choosing how to allocate jobs to resources, or how to pair jobs together with resources, you can do that scheduling. You can do that pairing based on a priority. Like a user can say, "This is a high-priority job. These are like payments that are going to go out to partner drivers," or something like that. Something would be really high-priority, but there's other models for how you allocate resources to jobs that won this hierarchical max/min fairness.

Can you, I guess, contrast priority based scheduling and hierarchical max/min fairness?

[00:27:23] MB: Yeah, sure. This is where Peloton is different than Borg, at least based on that literature which we have. So Borg has this priority bands where they identify the priority of each workloads across companies and then they say, "This priority band will run higher priority than the other workloads." They need to go and find out the priority at the global level, at the company level, where they identify a job and put it into a band and then run them, correct?

So that's a good way to support priorities. However, we found out there are some gaps in this approach, and one of the gap is let's say if I'm an organization within a company and every organization have some quota to run in the compute resources. So if at any certain point of time, if my high priority jobs are not running, then as an organization within a company, I want my low priority jobs to run within that quota.

But because of this priority banding, you cannot do that. Even if your high priority bands are free, you cannot go it, because your priority bands cannot go and run on the high priority band resources, right? So that was the gap we found out. I've been working on YARN and Min was in DRS scheduler and VMWare. So in YARN and DRS, we were using hierarchical min/max fairness model, where each organization will be assigned a quota and the priority will be relative to only for that organization.

So we call them resource pool. So resource pool is a virtual entity that identifies an organization, which says, "I have, let's say, 100 CPU of quota." So we let you run any workloads based on the priority you defined, and that priority is within that resource pool and you don't need to go and identify the whole priority across company. So you are – You know in your team which jobs are which priority, and you can go and run those job. Then scheduler can go and identify which priority is which and then schedule it accordingly.

However, these max/min fairness also allow you to go and get the additional quota from other organizations if they are not using it. So if let's say there are two organization, organization one and organization two, organization two is not using the quota right now because there's not much enough workloads. So organization one can go and borrow the resources from them. However, when organization two comes in and says, "I need more quota," then Peloton is able to preempt the jobs which is running over their quota for organization one and give resources back to organization two. So this is the difference between the max/min fairness as well as the priority bands.

[00:30:12] JM: Did you find out anything else when you were scoping out this project? You talked to some Borg people, you talked to some Googlers. Did you find anything else that they talked about in the construction of Borg that was either not in the paper or just people hadn't really picked up from the paper? Because the paper is very deep. The Borg paper is very deep. So was there anything else that you found that you ended up implementing in Peloton?

[00:30:40] MB: I mean I don't think we get any other information other than what is there in the paper. So we had to view whatever is published literature from Google. We didn't have anything else other than what is published.

[00:30:52] JM: Okay. So that was enough to – There was just more information in the Borg paper that hadn't been implemented in Kubernetes, for example? I guess put another way, like when you look at Kubernetes – So Kubernetes was kind of based on Borg. I'm just wondering Kubernetes has a scheduler in it. So how does the Kubernetes scheduler compare to what you wanted to build with Peloton.

[00:31:15] MC: Yeah. Let me clarify that. I think, at least from our understanding of Borg by talking to other Googleists, many of them are really senior people. The architecture of Kubernetes is quite different than Borg. So Borg is actually a kind of – Whether if we look at Borg, right? They have Borg master and Borg [inaudible 00:31:35]. Borg master is basically active [inaudible 00:31:38] standby mode. You have like one Borg master leader and then four followers. For example, like one feature in Borg, which is not in Kubernetes today is in Kubernetes, Borg does, they call it link-shard. So let's say you have 10,000 node agents or

1,000 agents, they were doing sharding, like 1/5th of those agents were reporting to one Borg master. Then each Borg master will do some aggregation then reporting to the leader.

So that's how Borg can scale to a much larger bigger cluster than Kubernetes, because Kubernetes right now is basically using API server [inaudible 00:32:19] watch and post notice status to Kubernetes API servers and it's going to hit etcd, and etcd is not shardable today. So basically, all those [inaudible 00:32:31] one etcd master. There's some scalability limitations over Kubernetes.

Also in some sense, some other design philosophies are different in Kubernetes. For example, in Kubernetes, all the information about a pod is like they have spec, which is a user provide an input as well as the status. They are all stored in one etcd node. Whenever there are any state change, they have to make copy of the whole thing. So that's also kind of limiting some of the Kubernetes scalability as well.

Then from a scheduling perspective, for example, yes, Kubernetes does allow us to plug in different schedulers. But the scheduler in Kubernetes is more like – We call it a placement engine. We call it placement engine in Peloton. So it's basically decide where to run this – Which container should be running on which machine. They don't really decide how much resource should this particular use at any given time.

So back to what Mayank said, the nice feature in Peloton is resourceful, basically give you elastic resource sharing between different organizations. So in Kubernetes, they have something called a quota manager, which is [inaudible 00:33:44] today. Basically, if you allocate 10,000 calls for this organization, if that organization is not using it, other people still can only use that quota.

[00:33:54] MB: So that is a differentiator for Peloton also. One of the differentiator is that we allow elastically grow and shrink resources between different quota boundaries, which is not present at least in Kubernetes right now, which is present in YARN, but that is also not good for stateless workloads. So these are different things, because we don't want to have any statically partitioning into our cluster to increase a utilization. For increasing utilization, we need to remove those static partition.

[00:34:27] JM: As we get into a discussion of how Peloton works, when did you go from the process of scoping out Peloton to starting to build it? How did you get to the point where you're confident in your architecture? Was there as a long architectural process before you started implementation?

[00:34:47] MB: Yes. Actually, we came up with the current [inaudible 00:34:51] architecture through multiple [inaudible 00:34:54] –

[00:34:55] JM: So you didn't even start building it. You just architected the whole thing

[00:34:58] MC: Well actually, we did [inaudible 00:34:59] is we did some – Basically, we did some prototyping and then – I came from VMware. One mistake we had 15 years ago is we have a monolithic class manager. You probably heard about it as vCenter. So that's actually a pretty – Like it's widely used. If anybody use vSphere, they know like vSphere, vCenter. So that's basically the class manager for virtual machines for all the VMware platform. But that's actually a very monolithic one. So that's why when we did Peloton, we definitely don't want a monolithic architecture at all. But how to –

[00:35:39] JM: What is – When you say monolithic, what is the monolith that you're referring to?

[00:35:43] MC: Monolithic is basically you have a single process, which does all the things, scheduling, monitoring, [inaudible 00:35:51], everything. So in some sense, even Borg is, in my opinion, is monolithic as well. It has a leader follower, but still only the leader is doing most of the things. That's why we don't want a monolithic architecture. But on the other hand, it's very difficult to break the functionalities into different modules and still keep the same performers, because one example is – So a scheduler has a couple of pieces. You need something to monitor all the node status. So know where is the fluidity resource on which machine.

So that's going to grow to all of the [inaudible 00:36:29] machines. Then you have the user-facing job and task lifecycle management, which is grow at an order of number of containers or the jobs from the user. They have this mix matching to decide where to place what. That's kind

of a combination of both. Basically, you have [inaudible 00:36:48] containers, you have tens of thousands of nodes. How do you breakdown this is a little bit tricky.

So yeah, we came up – The current [inaudible 00:36:56] architecture of the four components after some much more iterations of prototyping and design discussions, all those things. So I'm still pretty happy with what we have today. Basically, what we can [inaudible 00:37:11] architecture, we have a host manager, which is node-facing. You can shard them. It's similar to the Borg master does. You have linked-shard. Then user-facing, you have we call a job manager. You can shard them as well based on the number of jobs.

The only thing which is not shardable today is we call the resource manager, which is only managing [inaudible 00:37:32] of the resource pools. But luckily, the number of organizations or the tree is supposedly to be relatively small within your organization, like in the order of thousands. So that can be totally fitted into one master. Then as a component we call a placement engine, which can be powerized as well. So you can have much more – Even today, Peloton has much [inaudible 00:37:57] engines, like for example, [inaudible 00:37:58] engines for batch workload. One placement engine for stateful, one placement engine for stateless. So those can be powerized as well.

[00:38:06] MB: I mean, yes. We had like a marathon of architecture decisions and discussions and POCs and we were like multiple months we were discussing about, "Okay, which approach to take?" We read through maybe whatever – Maybe all of the paper which were available in the literature at that time. So yes, we did a lot of discussions, a lot of POCs. After that, we came up with this design maybe by end of 2016. Since then, this design we were implementing, and I think that we are holding up to their design. So we are pretty happy about what we came up with. I think that's pretty scalable.

[SPONSOR MESSAGE]

[00:38:52] JM: Triplebyte fast-tracks your path to a great new career. Take the Triplebyte quiz and interview and then skip straight to final interview opportunities with over 450 top tech companies, such as Dropbox, Asana and Reddit. After you're in the Triplebyte system, you stay there, saving you tons of time and energy.

We ran an experiment earlier this year and Software Engineering Daily listeners who have taken the test are three times more likely to be in their top bracket of quiz scores. So take the quiz yourself anytime even just for fun at triplebyte.com/sedaily. It's free for engineers, and as you make it through the process, Triplebyte will even cover the cost of your flights and hotels for final interviews at the hiring companies. That's pretty sweet.

Triplebyte helps engineers identify high-growth opportunities, get a foot in the door and negotiate multiple offers. I recommend checking out triplebyte.com/sedaily, because going through the hiring process is really painful and really time-consuming. So Triplebyte saves you a lot of time. I'm a big fan of what they're doing over there and they're also doing a lot of research. You can check out the Triplebyte blog. You can check out some of the episodes we've done with Triplebyte founders. It's just a fascinating company and I think they're doing something that's really useful to engineers. So check out Triplebyte. That's T-R-I-P-L-E-B-Y-T-E.com/sedaily. Triplebyte. Byte as in 8 bits.

Thanks to Triplebyte, and check it out.

[INTERVIEW CONTINUED]

[00:40:42] JM: One thing that stands out to me about the Peloton project that also stood out to me in a show I did a while ago the Uber data platform is that you've really architected something for the long-term. This is a really long time horizon project. It's not like let's spin up the MVP. Let's move as fast as we can. It's like we're going to build this thing that's going to be a core piece of infrastructure for the lifetime of the company.

How is that different? Could you contrast the idea of building core infrastructure that's going to last the lifetime of the company and the development process, the architecture process, the testing process, the release process. How does that different from something like some production microservice that's going to get rotated out in a couple of years?

[00:41:36] MB: So the basic difference between building a core infrastructure piece than a service is we need to think about not the current workloads. We need to think about the

workloads which will be there in the company down the line five years, and we can we support that many load into our system?

So the difference between the infrastructure and services is – Code infrastructure and services is that code services can go down and multiple instances will come up again. We can restart them or we can bring them somewhere else. Code infrastructure, you don't have that liberty. You need to be highly scalable. You need to be highly available. You need to – Your latencies cannot grow after a certain amount of time. You have to do all kind of fault tolerance. You need to think through each and every condition. You cannot do it in one shard. So that's iterative process.

You put it in production. You get all the feedback. You again iterate on it. So these things gets time to build. Microservices can be built faster than a code infrastructure. You have to be thoroughly tested. Your e-check in to the code has to be good through a performance test in code infrastructure. So these are something best practices you need to build into your development process. Of course, you get bugs. You get production outages and you learn from them and then you iterate over them. So that's what we did.

[00:43:06] MC: Yeah, and I totally agree with you. Code infrastructure is a big investment. I don't know if it's a lifetime investment for a company, but it's definitely a good investment. For example, Borg has been Google for more than 10 years, I think.

[00:43:21] JM: But I guess even then they updated to Omega, which –

[00:43:23] MC: Yeah, that failed, right? Because you couldn't not easily replace something like Borg to some new system.

[00:43:31] JM: Oh, did Omega fail?

[00:43:32] MC: Yeah.

[00:43:32] JM: Oh, I thought it was an upgrade.

[00:43:34] MC: No. It is from what we learned. Omega didn't really replaced Borg. They borrowed some idea from Omega and improved Borg, but they didn't really able to replace it. I actually even asked some of my friends at Google and said, "Hey, when do you guys think you can replace Borg with Kubernetes?" That's probably pretty far along. So I think you're absolutely right. That's very tricky to – You needed like deliberate thinking for those kind of software.

That's why actually when we started Peloton, the first thing we had is we call the performance test. So we actually had like running Peloton on top of Peloton as a virtual cluster, because you need to test like large number of nodes and large job after – Basically, after every change then to master, we bring up a virtual cluster and then we run performance test.

Also, we invested in something called a failure test, which is we simulate different failure cases in the system and then make sure all the failures are the recovered and behaved as expected. Those kinds of things, people are doing that for critical microservices, like Uber's marketplace service, but not like an average low priority microservices.

[00:44:57] JM: There are multiple ways that a user might want to interact with Peloton. There's a UI. There's a CLI. There's an API. What are the different ways that a user might want to interface with the Peloton system?

[00:45:11] MB: We have multiple ways, right? As you said, we have UI. We have CLIs. We have Peloton clients. So Peloton is a good [inaudible 00:45:21]. Peloton exposes the GRPC interfaces. So by that, any external, any language you can use to GRPC [inaudible 00:45:28] and call Peloton. So we have the Java client, we have Go client, we have Python client within Uber. We can interact with those.

There is a well-defined API and we can build any external client. So people have their different orchestration engine on top of Peloton APIs, which uses Peloton client and interact with it. People go and use CLI to interact with it. People go to the UI and interact with. There are multiple ways we can do it.

[00:45:57] MC: Yeah, and addition to that. Basically as Mayank said, we already have multiple extensions or solutions build on top of Peloton. Today, there are more than 10 systems within

Uber are using Peloton APIs. For example, if you are a microservice developer, you'll go to your deploy, which is our Uber internal deployment system and that can submit jobs through Peloton, to Peloton [inaudible 00:46:23] API. Then if you are a batch user, like if you want to do some Spark job, then we have a job server for Spark which talks to called a jargon, which use the Peloton API talk to Peloton to submit jobs.

Then if you're a machine learning user want some of the Tensorflow job, you are most likely go to Michelangelo, which is a machine learning platform which user the Peloton API to launch jobs as well. If there are some infrastructure services, they can directly use Peloton CLI and Peloton job file to launch those jobs.

[00:46:57] JM: One component in Peloton is the storage gateway. What is the storage gateway?

[00:47:03] MC: The storage gateway is basically like a storage abstraction we have between Peloton and the actual storage system. Today, we're using Cassandra for Peloton as the storage engine. But as the storage gateway is abstracted out of that [inaudible 00:47:21]. So we can plug in other storage solutions like MySQL or potentially etcd or other storage system.

[00:47:29] JM: So what does that mean? Does that mean if I spin up a MySQL cluster every write to that MySQL database goes through Cassandra?

[00:47:37] MC: No. No. No. Basically, the storage gateway is a library within Peloton. Basically, all the application code in the Peloton don't need to know is a Cassandra or MySQL.

[00:47:47] MB: So Peloton need to restore its metadata to some storage. So for that metadata, we have this storage gateway where we can plug in any storage. So we had like Cassandra, we had like MySQL. Tomorrow we can write a driver for any other storage we want to. So that abstracts away all the Peloton demons to interact with the storage. So they don't need to worry about. They just talk to interface, and interface is implemented for a specific storage.

[00:48:14] JM: Could you map that to the Kubernetes world? Does Kubernetes have a storage gateway?

[00:48:19] MC: Yes. Actually, Kubernetes does have some server. If you look at the API server, they have a storage interface which abstract it out to the etcd backend. So you could potentially plug in something else if it opens up that interface.

[00:48:36] JM: Okay. Basically you have the storage gateway that is an interface into whatever you're using to maintain some consistency, like configuration logic, stuff that you need consistent.

[00:48:50] MC: Yes. Something you need persistence.

[00:48:53] JM: Oh, persistent.

[00:48:53] MC: Yeah. That's why it's a database. For example, people submit a job, you have to persist all those job specs. Also, for the job status, whatever job you have to complete, you have to persist all the completion information.

[00:49:10] JM: Okay. I think I understand, because I know that Mesos uses ZooKeeper, and so –

[00:49:15] MC: Not really. Actually –

[00:49:16] JM: Oh, it's not?

[00:49:17] MC: Yeah. Mesos is using ZooKeeper as a [inaudible 00:49:19], but its persistence state is in something called a replica log, which is a Paxos – Basically, Mesos is very similar to a blog model. So you basically have a built storage engine in the process which writes something called a replicated log into disk as file. But you have to be highly available. That's why they're on Paxos protocol. So every write goes to all the five nodes and they will dump in the data in each file – Oh, in different machines.

By the way, that's so how Aurora, the scheduler from Twitter is using. That's has been the performance bottleneck for Aurora as well. So actually, some Aurora developer has been

expressed before, like one thing they have been regretted is they should have a separate database solution instead of using the repercussion logs, because basically building a storage system, a database system, is hard. So you don't want to build this yourself. You should offload some mature storage system.

[00:50:28] MB: So let me a little bit abstract this out, right? So each scheduler or each resource manager needs some storage to store all the job metadata, all the status, all the history information, because you can go back and see what happened to my job. So you want that persistence. For recovery, you need – Let's say if the scheduler goes down, then when you come up, you need somewhere to recover from.

As Min said, Aurora has been using replicated logs. YARN uses HDFS. Kubernetes uses etcd, and that's how Peloton is also using Cassandra or MySQL which we can plug in.

[00:51:10] JM: That's helpful. People who are deeply curious about this can go read the Peloton blog post, which has a lot of detail on the overall system. Now, I know we're running out of time here, so I want to talk a little about the usage. So if I'm a user and I want to spin up – Maybe I want to spin up a service. It's got a Redis cluster alongside it or maybe I'm spinning up a data science job. Give me a little more detail on my usage. Do I need to know a lot about how Peloton works or do I just make some simple request to it to spin up resources? What's the usage model?

[00:51:44] MC: For that, actually it's pretty simple. So first thing you have to do is you have to get a resource pool in Peloton, which is similar to you need a name space in Kubernetes. Resource pool will going to basically tell you how much resource you have are reserved, or elastically you can borrow from other people. Then you can write, we call it Peloton job file, which is similar to the Kubernetes Java YAML file, where you're going to specify some metadata of your job. How many instance are you going to have and what other container image you're going to use, what other entry point script you're going to have. Then you just run a Peloton – You can use the Peloton CLI to submit the job to Peloton. Then you can use UI to watch the status of those containers and then look at the logs. It's pretty straightforward.

[00:52:35] JM: Yeah. To what degree is Peloton deployed to production today?

[00:52:40] MC: You mean within Uber?

[00:52:41] JM: Yeah, within Uber.

[00:52:42] MC: We are in all the data centers Uber has.

[00:52:45] JM: Okay. Is it managing all the infrastructure at this point?

[00:52:48] MC: Not yet. So basically, we are the only – Basically, I don't know –

[00:52:52] JM: Oh, okay. That's fine.

[00:52:53] MC: So basically we're like in –

[00:52:55] JM: You're rolling it out slowly but surely.

[00:52:57] MC: We have been in production for –

[00:52:59] MB: I think we've been in production for a long time and we are in all the data center as Min said. We are in thousands of machines and running a lot of containers. Still, we are moving all kind of workloads slowly and steadily to the Peloton platform.

[00:53:14] JM: What's the vision for the future of the project?

[00:53:16] MC: So we're still – The future is basically Peloton, I think we're still in the early time of the journey. So right now the first thing is we need to onboard all the workloads for Peloton. Then we can do lots of fancy stuff to improve the utilization, optimize a pretty similar algorithms, scheduling algorithm, all those things.

[00:53:37] MB: So the vision is pretty much to move all Uber workloads to Peloton. So that is the long-term vision. So we've been executing on that vision and then I think we are somewhere in middle or less than middle and then we have a long way to go.

[00:53:52] MC: Yeah, but it's basically run – We're pretty much ready for – Basically, we're pretty much in the process of migrating batch workload right now and we're going to be starting migrating all the microservice workload, which is a big chunk of the Uber computer workload today. So Peloton hopefully in the next quarter if everything works okay, we'll probably be able to wrap up by early next year. So at that time Peleton will have a significant footprint for Uber's data centers.

[00:54:24] JM: That's great. The end result will be you'll save a bunch of resources. You'll save a bunch of money.

[00:54:29] MC: Oh, yeah. That's actually going to be some essential.

[00:54:31] JM: Yeah, cool. Well, guys, thank you for coming on Software Engineering Daily. It's been really fun talking to you.

[00:54:35] MB: Thank you so much for inviting us. It was fun talking about the Peloton.

[END OF INTERVIEW]

If you're not a developer, it's not a problem. There's plenty that you can do without writing a lot of code, although of course if you are a developer, then you can do much more. You can explore all the resources on the Wix Code's site to learn more about web development wherever you are in your developer career. You can discover video tutorials, articles, code snippets, API references and a lively forum where you can get advanced tips from Wix Code experts.

Check it out for yourself at wicks.com/sed. That's wix.com/sed. You can get 10% off your premium plan while developing a website quickly for the web. To get that 10% off the premium plan and support Software Engineering Daily, go to wix.com/sed and see what you can do with Wix Code today.

[END]

