EPISODE 468

[INTRODUCTION]

[0:00:00.1] JM: The labor market is moving online. Taxi drivers are joining Uber and Lyft. Digital freelancers are selling their services through Fiverr. Experienced software contractors are leaving contract agencies to join Gigster. Online labor market places create market efficiency by improving the communications between buyers and sellers. Workers make their own hours and their performance is judge by customers and algorithms rather than the skewed perspective of a human manager.

These marketplaces for human labor are in different verticals but they share a common problem, how do you most efficiently match supply and demand? Perfect marketplace matching is an unsolved problems. Hundreds of computer science papers have been written about the problems of stable matching and this is a problem that often turns out to be NP-complete. The stock market has been attempting to automate marketplace matching for decades and inefficiencies are discovered every year.

Today's show is about matching buyers and sellers on Thumbtack, a market place for local services. For the first seven years, Thumbtack was building liquidity on its two sided marketplace. During those years, the model for a job requests was as follows: Let's say I was on Thumbtack looking for someone to paint my house. I would post a job that would say, "I am looking for house painters." The workers on Thumbtack that paint houses could see my job and place a bid on it. Then I would choose from the bids and get my house painted.

This was the asynchronous model. The actions of the buyer and seller were not synchronized. There was a significant delay between the time when a buyer posted a job and the time when a seller places a bid and then another delay before the buyer selects from the sellers. Thumbtack recently moved to an instant matching model. After gathering data from the people selling services on the platform, Thumbtack is now able to avoid the asynchronous bidding process that cost people a lot of time.

In the new experience, a buyer goes onto the platform, requests a house painter and is instantly matched to someone who has a history of accepting house painting tasks that fit the parameters of the buyer. So from the user's perspective, this is a very simple, very desirable improvement. I can now just make a request and I instantly get offers from professionals who can do the request that I have been looking for but from Thumbtack's perspective, there was actually significant architectural change that was required.

In the asynchronous model, the user requests lined up in a queue and then they were matched with pros who placed the bids on the items on that queue but in the instant matching model, a user request becomes more of a search query. The parameters of the request hit this index of pros and get a response back immediately. Xing Chen is an engineer from Thumbtack and he joins the show to describe the re-architecture process; how Thumbtack went from an asynchronous matching system to synchronous instant matching. We also explore some of the architectural themes of Thumbtack, which we dive into in further detail in tomorrow's episode about scaling Thumbtack's infrastructure, which uses both AWS and Google Cloud. So I recommend checking out that episode as well and it's a great compliment to today's episode.

On Software Engineering Daily, we've explored the software architecture and business models of these different marketplaces. From Uber to Fiverr to Instacart and now Thumbtack and you can find all of these old episodes by downloading the Software Engineering Daily app for iOS and for Android and the great thing about these apps is that they give you all of our old episodes. In other podcast players, you are only going to find the most recent one hundred episodes but in this apps you will find all 600 episodes with related episodes. If you listen to this episode, you might get recommended to other episodes about marketplaces or about cloud services or about different architectural patterns.

We are really trying to create a new way to consume content about software engineering. These apps are open sourced at GitHub.com/softwareengineeringdaily. So if you're looking for an open source project to get involved with, you can definitely check it out. We would love to get your help and with that, let's get on with this episode with Xing Chen from Thumbtack.

[SPONSOR MESSAGE]

[0:04:48.2] JM: Amazon Redshift powers the analytics of your business and Intermix.io powers the analytics of your Redshift. Your dashboards are loading slowly, your queries are getting stuck, your business intelligence tools are chocking on data. The problem could be with how you are managing your Redshift cluster. Intermix.io gives you the tools that you need to analyze your Amazon Redshift performance and improve the tool chain of everyone downstream from your data warehouse.

The team at Intermix has seen so many Redshift clusters they are confident that they can solve whatever performance issues you are having. Go to intermix.io/sedaily to get a 30 day free trial of Intermix. Intermix.io gives you performance analytics for Amazon Redshift. Intermix collects all your Redshift logs and makes it easy to figure out what's wrong so that you can take action all in a nice intuitive dashboard.

The alternative is doing that yourself. Running a bunch of scripts to get your diagnostic data and then figuring out how to visualize and manage it. What a nightmare and a waste of time. Intermix is used by Postmates, Type Form, Udemy and other data teams who need insights into their Redshift cluster. Go to intermix.io/sedaily to try out your free 30 day trial of Intermix and get your Redshift cluster under better analytics. Thanks to Intermix for being a new sponsor of Software Engineering Daily.

[INTERVIEW]

[0:06:33.4] JM: Xing Chen is an engineer at Thumbtack. Xing welcome to Software Engineering Daily.

[0:06:37.4] XC: Yeah, thanks Jeff. Great to be here.

[0:06:39.7] JM: Thumbtack is a marketplace for household labor or related tasks, event planning. Explain what Thumbtack is, describe the marketplace and the different things that people could buy on Thumbtack.

[0:06:56.7] XC: Yeah, so Thumbtack is basically a local services market place where you can come and find professionals to do all sorts of work for you and that ranges from my last few

Thumbtack hires where actually an interior designer right now to help with the house that I am trying to close. An iPhone screen repair when I dropped my phone a couple of months ago. We do things like personal training, wedding catering, tennis lessons, which I have tried before and so it is actually quite a wide range and sort of the home space is just one of our big areas.

[0:07:38.3] JM: Right, definitely. Why did Thumbtack gain market share against the other competitors in the space? Because for a while there was a wide spectrum of different people, different companies that were offering platforms to connect people who needed a wide range of services and those different service providers.

[0:07:58.0] XC: Yeah, good question. I think if I had to pin it one thing I think it would probably be the sort of intent-fullness of the results that we return. If I think about the platforms out there, it's not that hard. I mean it is still hard to return a good list of pros. So certain sites could give you a bunch of profiles or a list or a directory of people to do the job. I think what made early Thumbtack really successful was we didn't just give you a list, we gave you a professional who would actually be interested in your job and actually go to your house and paint your wall, and sometimes like that end to end task completion is the really hard part about all of this.

You know, a little bit of my background, before this I worked at Google for a little over five years and my last project there was working on Google Now, which was kind of the first version of the Google assistant and with Google Now, the problem was we could never actually get the person into someone's house to do personal training or catering your kid's birthday. The problem with that was sort of the product always stopped at information retrieval and giving users a bunch of information basically on a website. But I think where Thumbtack really sells is going beyond that to this last mile physical world connection, which I think is really actually the hard part.

[0:09:29.2] JM: What was Google Now? I don't remember that.

[0:09:31.2] XC: It was basically this app where we tried to predict what users would want to know before the users even knew that they needed to know that and so for example we'd show you things like traffic to home or flight delays or when your package arrived or news that you might be interested in or sports forums or things like that.

[0:09:50.8] JM: Oh I actually used that when I had a Nexus. I thought it was pretty useful. So did it get deprecated or did it just get worked into some other products?

[0:09:59.8] XC: No, it's sort of just evolving into part of the assistant. There was always both the predictive cards and the voice search aspect of it and the voice search and I don't really know anymore because I left a couple of years ago but that's essentially becoming the system and what you see in Google Home and so on.

[0:10:20.0] JM: At this point you've been at Thumbtack for almost three years. We're going to talk about the market place refactoring discussion, which you wrote a blog post about it. It's very interesting, basically you migrated one form of a market place to a different form of a market place. We'll get into that, but to give people a little more background on Thumbtack, what were some of the earlier engineering challenges that you worked on in your first two and a half years at Thumbtack?

[0:10:51.8] XC: Sure, yeah. To give a little bit of context, the way that Thumbtack traditionally works is I send a request for, let's say interior design. I just did this last week. Thumbtack has a network of designers, let's say in San Francisco, and we send the request to all of the designers who we think might be interested in your job and then the designers can then bid on the job and the way the business model works is that they would pay a bid on the job and I would get up to five results back.

So I would get a handful of interior designers who might do my job and that was sort of the traditional Thumbtack marketplace and it actually worked really well because for early Thumbtack because it was fundamentally high intent, when you got pros who were willing to pay the bid on your jobs. So the designers are giving some commitment if they bid. The challenge in that marketplace is in some ways similar to the instant marketplace but in some ways a little different in some sense.

So when we get a job from a customer at Thumbtack, we need to actually try to figure out which professionals are interested. So you can think of this as basically a two-sided search problem, right? So we have a customer coming in, they are searching for pros. At the same time, we have a bunch of professionals, they are searching for customers and so what we are trying to do actually is we're trying to understand what each side wants and then do the matching between them.

So when I first joined Thumbtack actually it was pretty interesting, the matching happening in this marketplace was really simple and I won't say how simple it was. Although it was amazing it was working despite how simple it was and the business was growing quite fast and one of the first things I did — so I joined the marketplace team back then and back then, the market place team was very small and not really doing marketplace work. We were just fixing infrastructure.

Once we got past – yeah, you probably know how that goes. Once we got past getting our databases to scale and kind of building out a basic data platform then we could start using the data to try to do some of these optimization and actually Thumbtack has a fair amount of data on what local services jobs happen. So I remember one of the early things we did was we tried to predict. We tried to use historical data to predict what jobs a professional would pay to bid on. What jobs they would send their quote on.

We called this model "request affinity" because we were trying to measure a pro's affinity for a customer request and we basically built this fairly simple logistic regression based on historical signals like what pro historically bid on many types of jobs with many different preferences, which types did they bid on the most often? If they worked in different areas, which areas would they prefer over others based on their past data?

So we had signals like job preferences, location and things like that in there and I just remember when we first launched this model, we were able to decrease, for a given pro, we were able to decrease the number of projects — the number of customer request that we sent them by almost like a third while still increasing the number of bids that they sent. So that was pretty interesting. The funny backstory of the architecture of that was also at the time it was a super early product. No one really knew whether a model like this would matter.

So instead of building out a real data driven system and a real serving system to serve a model, we just hacked it up. So we loaded this massive file into memory and our matching service. We found the biggest machine that we could find on AWS to just store as much feature data that we could and then I think we also had a memory leak. So we had to restart this machines weekly to make sure that the server didn't blow up.

[0:14:56.8] JM: That's one way to solve a memory leak.

[0:14:58.3] XC: Yeah that's one way to do it and you know it's when you are trying to move fast, right? And just prove if this thing works or not and it turned out that it worked. So the first experiment was really successful and we're kind of like, "Oh okay, I guess we should invest in building a system to the served models.

[SPONSOR MESSAGE]

[0:15:21.3] JM: Dice helps you accelerate your tech career. Whether you are actively looking for a job or you need insights to grow in your current role, Dice has the resources that you need. Dice's mobile app is the fastest and easiest way to get ahead. Search thousands of tech jobs from software engineering to UI to UX to product management. Discover your worth with Dice's salary predictor based on your unique skill set.

Uncover new opportunities with Dice's new career pathing tool, which can give you insights about the best types of roles to transition to and the skills that you'll need to get there. Manage your tech career and download the Dice careers app on android or iOS today. You could check out dice.com/sedaily and support Software Engineering Daily. That way you can find out more about Dice and their products and their services by going to dice.com/sedaily.

Thanks to Dice for being a continued sponsor and let's get back to this episode.

[INTERVIEW CONTINUED]

[0:16:37.1] JM: You know, just to explain to people a little bit further who are unfamiliar with the Thumbtack marketplace, I think of this as a second generation marketplace business whereas the first generation of general, I don't want to say home services anymore because I already said that once and it is clearly more general. But maybe we'll say real world services. You know, you could get tennis lessons and whatnot.

You know the first generation of these services is like you go on Craig's List. Or you go on some other platform that isn't basically an index of what you on Thumbtack call professional service providers and as a customer, I am going through these and I am sending messages to these different people and maybe they get back to me, maybe they don't and it's a really frustrating experience., there is no ratings. But really one big thing Thumbtack solved is the problem of the customer having to do a bunch of leg work when they're just trying to spend their money.

And you turned that on its head with Thumbtack where you say, "If you're a customer you just express your interest in getting your house painted or getting dancing lessons," and you have professionals who are actually placing bids on what they are willing to accept. So the bid placing process shows their intent to actually close the deal and it makes for a much more seamless process and it is simple but it's also nuanced. It is a different – I think of it as like Tinder for example.

I did a couple of shows with Tinder recently and what Tinder really did that was innovative, aside from the swiping user interface, is the double opt in. Is the fact that they only connected people and forced them to do work to talk to each other when there was a double opt in, when both people have committed to expressing interest to one another and it's a simple mechanism but it lowers the friction of the overall marketplace and when you have an insight like that.

Like, "Oh this is actually a completely better way to do a marketplace," that's going to create a lot of volume to come in and then that's why you are talking like, "Oh the first two and a half years there were like, "Okay we figured out this breakthrough and how to do the market place right. Now let's just build the infrastructure to be stable and sort of have the minimum amount of work of infrastructure that will actually service this breakthrough in marketplace dynamics." Am I describing things correctly?

[0:19:10.3] XC: Yeah, definitely. That's exactly how we think about it.

[0:19:13.9] JM: Yeah. So beyond that, once you had the infrastructure stabilized, you were able to say, "Okay now that we've got this infrastructure stabilized, let's figure out how can we improve this marketplace?" So in the original model, again, the pros are bidding for jobs. So if I am a professional painter I'm going to need to put in the time and the effort and the money just to get hired and if I win the bid I have to go and complete that paint job. What are the pros and cons of that marketplace format?

[0:19:46.8] XC: Yeah, great question. So, you know, probably the strongest pro to that format is the intent of the bid that you get and so to your point about first versus the second generation local service marketplaces, in the first generation marketplace you're really just getting an index of sort of I think of it as this flat index of contact information and you got a little metadata and a bunch of contact information.

Thumbtack in some sense is looking to in the second generation of local services, we're trying to go beyond that flat contact info to actually send the person to your house to actually do a job and so that sort of – that's the biggest benefit of the bidding model. The biggest downside of the bidding model, and if you think about what a professional is doing, we [inaudible] our products here and so you sign up as a professional, let's say a professional resume writer.

And we have written some — helped write some resumes here and what happens is you find that you are actually doing the same job selection over and over and over and so let's say I can help write tech resumes in the Bay area. Every day if I'm really doing this as my living, everyday I'm thinking like, "All right, here is a tech job, three years of experience in Oakland. Here is another tech job, three years of experience in San Francisco bid," and you are just doing this ten times a day over and over. And it's like, "Well that's a lot of effort and it does not really lend itself to scale," and that's probably the biggest downside.

[0:21:31.8] JM: Do you have any numbers that illustrate how much volume is going through the system, just to give people a perspective on how much volume is going through the market place?

[0:21:41.0] XC: Yeah, so roughly speaking we do millions of jobs, millions of projects on the platform every year. We have a quarter million active paying pros and to give you some sense here, when we say "millions of projects", each project is a pretty significant thing, right? Like a wedding catering job, an interior design job, personal training client. So these are big projects and we are doing millions of them.

[0:22:08.1] JM: How much internal tooling do you have to build in order to support the users in the market place? Because I have dealt with all kinds of these marketplaces; I've dealt with Uber and Fiverr and Lyft and all of these different things. They all have internal support tools because we're still figuring out how online marketplaces will collide with the real world and there is a lot of frictions and so you have to give the customer success team the tools to deal with those edge cases. What kinds of internal tooling do you have to build to support those users in the marketplace?

[0:22:44.2] XC: Yeah, good question. So we are actually somewhat different from a business like Uber or Lyft where for them to really scale the marketplace, they kind of go more city by city and there's an ops team on the ground to build up the inventory in that city. For us, it's actually much easier. So there is not a strong sort of ops-driven component to our scale and the basic advantage of Thumbtack is, we can just turn it on anywhere and anyone who is an existing small business in that marketplace and in that market can just go ahead and use the platform. And so that's in some sense like the inherent advantage of the way that the platform works is we don't really require like a strong ops component to scale.

Now with that said, we do definitely have a great ops team here that helps with very specific things. So one very specific area is marketplace trust and safety and, you know, this is where we have a great sort of I would call it human-computer interface in the team sense where on the human side, we have a team reviewing cases. But then we also try to automatically detect issues on the computer side. So to give one example of this, when I got here most of our requests were actually past through an ops team to scan for scams. Because customers will actually try to scam pros and a lot of these are classic like, "I bought a house and can you wire an extra amount of money to my landlord who is to the UK," or something. So it's sort of these classic wire fraud scams and people would try to scam the pros with this.

So initially when Thumbtack was small we just said, "Okay, we'll just send all of these to a human to read," right? It's kind of the simple bootstrapped solution. As we got bigger and bigger, obviously that created scale problems and bigger and bigger delays and so actually just this past year, we started deploying machine learning systems to basically automatically pick out which requests are likely to be spam and we use all sorts of signals around the customer requests and where it is coming from and the content of the request and third parties to add signals and so that gives you a prediction of whether this thing is a piece of spam or not and then given that prediction, we can decide if we want to automatically deal with it or send it for manual review and sort of just building out these interfaces between our automatic systems and the manual review is really important.

[0:25:23.8] JM: Now that we have given people a table stakes view of what Thumbtack is and how that unique market place component works, let's describe the reformatting of the marketplace because that will get us towards a discussion around software engineering because you had to actually reformat the backend infrastructure in order to accommodate this marketplace change.

So we talked about baseline transaction that Thumbtack started out as. Where I as a user post, "Hey I am looking for a painter to come paint my house," and you as a professional painter are going to see that request and you can choose, "Okay I want to bid on this. I want to state a price and when I'm going to do it, or the details of what I'm going to offer to you as the customer," and the customer can accept that. Describe the new marketplace structure that you decided to move to.

[0:26:25.7] XC: Yeah, so to give some context on this, what we realized – so we were talking about first versus second generation marketplaces and what we realized is in this bidding world, it's great that we get high intent pros but you only get so many of them because you are forcing them to sit there and click these buttons in our app ten times a day and they are clicking the same buttons and it's like, "Well that's what computers were invented for."

So we realize it's more like, "Oh well actually you don't need them to click the same buttons ten times a day. You can just have them click the buttons once and tell you that they are an interior designer with contemporary modern taste and they like to do design of rustic condos in the mission in San Francisco or something," and they don't have to pick that every day. They can just tell us once and then you can also say, "Okay well what if they also tell us their schedule? What if they also tell us when they do these jobs? What if they tell us their capacity? How many jobs they're willing to do?" and so on.

And so, if you collect all of that information from a pro, what are you really creating is sort of I would call it this tier terminology, this second generation index. This time, we are not indexing pros. We are not indexing a bunch of contact information and profiles. What we're indexing is actually in some sense future jobs that could be done, right? We are indexing the fact that in San Francisco on Saturday nights, there are four DJ's that play EDM and these are the types of venues where they are willing to play and these are the types of music that they like and by the way, here are some past jobs."

And if we actually know that, in the future there are a bunch of DJ jobs that are going to happen, if we have a customer who's like, "Well I have a birthday that I want to get a DJ for." I don't know who does that but you know let's say someone's like, "I want a DJ for my birthday," we can say like, "Oh yeah, here are three people who do the DJ job, the kind of music that you want on the day that you want it," and if we do that right then those three people will actually be willing to show up at the venue and DJ the job, right? So sometimes we're trying to create this local services jobs index of future jobs that can happen. Does that make sense?

[0:29:04.8] JM: It does and the part of this that becomes problematic is indexing these different pros is not as easy as it sounds. It's not a highly normalized set of data. So you have this thing that you describe as the laundry problem, which is that you can't just categorize all of the housecleaners. So if I want a housecleaner, there's a question of, "Is the housecleaner that I am going to order on Thumbtack going to do my laundry or not?" Some of them are willing to do the laundry. Some of them are only willing to just do the housecleaning because you could argue, "Okay, doing laundry that's not cleaning the house. That is doing the laundry." So certain pros that you would assume are a good fit to do a task are actually maybe not going to do that task well.

So for example, another example, you have IKEA builders who don't like to pick up the furniture, they just like to come by your house and build the furniture. But some of them are going to be willing to pick up their furniture. Some math tutors are going to dislike tutoring a student in calculus. You know, they're a math tutor, you'd assume they're going to do calculus but that's actually not necessarily the case. Why is this so problematic? Why is the laundry problem so problematic when you're trying to build this instant matching marketplace?

[0:30:28.0] XC: Yeah, so it gets really interesting when you start to see a significant amount of both the supply and the demand side of the market. Sort of the example that you called out where you kind of realize like, "Oh, pros have very specific preferences about the kind of jobs they want to do," and so for us, what we're trying to do now is, you know, we want to create this index of pros of local services jobs. When we get a customer, we want to be able to tell them, "Okay, you are the professionals, the prices and the times that your job could happen."

Now, the interesting thing becomes these two things don't always line up, right? It's a two sided search. The people the professionals are searching — The customers of the professionals are searching for are not always the same customers that show up in a marketplace and so this thing that we called the laundry problem, we sort of jokingly branded it that because it was kind of funny when we found out that house cleaners don't like to do laundry at least in the San Francisco Bay Area.

There are actually a few different aspects of this problem. Kind of the first – So this kind of goes into the core, some of the core matching challenges here at Thumbtack. One of the first problems in this area is you can think of this as kind of machine learning speak, it's sort of this classic explorer versus exploit problem.

If you know that you have a great professional who can do house cleaning, laundry, whatever you want, you have a great math tutor that can just do everything up to like college level, whatever, linear algebra. You could give them as the top result for everyone, you could sort of try to send them to everyone's house to do all of the jobs. Now, the problem is going to be, they won't have the capacity to do all of the jobs and also, you know, another professional that might actually be really good at laundry or calculus will never realize that. Because we'll never send them to do these kinds of jobs.

So when you do the matching, it's actually really important not to sort of naively optimize a black box, right? Earlier I talked about this request affinity model where we optimize for the probability that a pro will quote on a job. If you just naively followed that optimization and you just kind of said, "Okay, I'm going to order the pros this way. I'll take the top 10% of pros and I'll just return that set to every job." You'll actually get a nice bump in metrics probably in the short term. But in the medium to long term, I could almost guarantee you the metrics would go south because what will happen is you'll effectively reduce your supply, right? That's kind of the challenge of balancing how much you kind of exploit the best pros who can do all of these sorts of jobs in the market, versus explore pros who can't do all of these jobs.

[0:33:34.9] JM: All right. To pull these different threads together, let me see if I understand this correctly. The original model is essentially asynchronous. I as a customer publish that I'm looking for a house painter and I have to wait while the different pros bid on that job. There's a little bit of waiting, there's some synchronicity. The place that you wanted to move to is, I as a customer state that I'm looking for a painter and I am instantly matched with people who have in the past, shown to have a propensity to accept jobs that are similar to mine and they have a certain bid that they have made in the past on these types of jobs and so I automatically get matched with the person with that bid and that is a synchronous process. I make a request and I instantly get a response.

Am I architecting things correctly so we can get into the engineering?

[0:34:38.3] XC: Yeah, exactly. So from a customer perspective, we can now give results instantly because we've already asked the pros upfront what they are, what types of jobs they are willing to do and you know, the other thing I'll say is on the pro side, this also becomes a fundamentally higher scale supply. Because now they're no longer gated by the fact that they have to quote on every single request.

[SPONSOR MESSAGE]

[0:36:15.4] JM: Do you have a product that is sold to software engineers? Are you looking to hire software engineers? Become a sponsor of software engineering daily and support the show while getting your company into the ears of 24,000 developers around the world. Developers listen to software engineering daily to find out about the latest strategies and tools, for building software. Send me an email to find out more, jeff@softwareengineeringdaily.com.

The sponsors of software engineering daily make this show possible and I have enjoyed advertising for some of the brands that I personally love using in my software projects. If you're curious about becoming a sponsor, send me an email or email your marketing director and tell them that they should send me an email. jeff@softwareengineeringdaily.com. Thanks as always for listening and supporting the show, and let's get on with the show.

[INTERVIEW CONTINUED]

[0:36:15.4] JM: The original infrastructure that reflects this is customers are making requests, they're posting jobs for painting or house cleaning or whatever else. Those jobs are getting queued up in SQS, an Amazon simple queue service and then those requests get pulled off of the queue and you have these workers that are matching those requests to pros and you're notifying the pros, "Hey, new jobs have been posted, maybe you want to bid on these new jobs?"

Very simple, scalable system, but it has that asynchronicity where you have the jobs that are being posted the SQS and then, asynchronously, these workers, these CPU workers are pulling jobs off of the SQS thing and matching them with pros who might be interested in those jobs. That is an inherently asynchronous architecture and you wanted to go to asynchronous architecture. Why didn't t ha original architecture suit your needs?

[0:37:20.1] XC: Yeah, the asynchronous architecture was basically designed for flexibility in terms of letting us do whatever we wanted with our matching. So what I mean by that is let's say we wanted to compute various regression models, let's say we wanted to pull from various online sources of data and, you know, and hit a bunch of back ends to get additional data to sort of enrich the data before pro selection and ranking.

It gave us flexibility to do all of that very, very slowly. Which is obviously not good but in this case, didn't matter because it was an asynchronous user experience and so just having the flexibility to do it slowly actually meant that we could focus our engineering effort on the hard part. which was sort of the matching algorithm and modeling instead of trying to compute it really fast. It actually give us a lot of room there. Now, obviously, when you need to return results instantly to a customer, it looks a little bit different.

It looks a little more a search where a customer is directly searching over an index of jobs and so now we want to – What happened was, we sort of said, "Okay, well, we wanted to build this index of jobs and return results instantly." That actually basically required us to change the whole back end, right? Because like an asynchronous worker architecture looks very different from a synchronous search stack.

So actually, the first thing we did and sort of the spirit of moving fast and scrappy as a startup, the first thing we did was we decided we would run a test without changing the architecture where we sort of just like wired up the results coming through these SQS queues and wired them back to the website and have sort of the front end clients pull in and you had these awful like 10 second, 95th percentile latency, which I sort of am even embarrassed to say here. But we ran like a small test this way because with super-fast. We're able to get it out within weeks. It sort of proved that instant results are better than slow custom results.

[0:39:30.4] JM: So you hacked the infrastructure to build a minimum viable product of the new instant matching service, but the back end was not exactly where you needed it to be at. You just use this slightly modified infrastructure to validate that this was something that people were even going to want, and you moved to this instant matching service which required a new back end.

So this new instant matching service, what was required in the re-architecture, to actually make those modifications to fulfil the new requirements for the synchronous matching service?

[0:40:11.0] XC: Yeah, so effectively we switch to essentially what looks like a modern search stack. So it's backed by Elasticsearch and we put some of the core jobs index information into Elasticsearch. We removed the SQS queues and things like that, and then now we have our matching service querying Elasticsearch. So they're sort of, you can think of the search process in sort of three steps.

So there is retrieval, where we do sort of the basic filtering by location, by job type, and then we enrich that data with further calls to other back ends. For example, we might call the quoting service back end to get prices for that job. Then finally, we do ranking and so once we have kind of all of these priced jobs, we again, look at sort of these two sides of the search problems. Which professionals are going to be responsive and actually go out to the person's home and do the painting job and which professionals are going to be most interesting and relevant to the customer in sort of balancing these two sides of the search?

[0:41:24.3] JM: In the asynchronous model, as you mentioned, you had not infinite time but you had much more time to process and do the matching properly to get the right pros, presented with the right jobs and do the matching more effectively. In this situation, it's much more latency sensitive.

So what was the offline – the things that you could essentially put in the request, the loop in the asynchronous process when the user posts their job, "I want to get my house painted" and you can do all kinds of processing around that request to be like, "Okay, let's figure out, who are the best pros that we can recommend for this job," and you've got plenty of time to do that. How does that change when you have to move to this synchronous model and you've got to do all that stuff offline before the requests actually come in?

[0:42:25.5] XC: Yeah, there are a couple of big thing that we had to add to make this work well. You know, one of the big ones was setting up this streaming near real time updater system for our Elasticsearch cluster and so imagine professionals are changing their preferences all the time when they change their preference. You don't want to serve a customer a job from that professional where their preference is outdated.

You know, someone says, "I don't do Italian cooking anymore. I'm moving on to whatever, modern American." Then, you know, two hours later you serve them in a search result for Italian, that looks really bad, right? You can't do that, and so what happens is you have to basically take all of these changes coming from a product front ends and then essentially aggregate them through this indexing pipeline, this real time indexing pipeline and then update the Elasticsearch index in your real time.

You know, that was pretty challenging to get right, right? Because you have to do a fairly high volume of writes to the index. If you have some writes that fail, you also want to have some sort of back up job. So we actually have like essentially a backup job that runs every night to sort of validate the consistency of the index against our other databases. You know, even the way that you roll this out was pretty tricky. So we actually spent a lot of effort — they were sort of two ways where we spent a lot of effort.

We spent a lot of effort validating that the numbers and the results were correct, right? Because you're actually switching essentially the core index that Thumbtack runs on from one set of tables to another. We actually – So what we did was we launched both side by side and we would look at the diff in the numbers and anytime the diff didn't match, we would ping our engineers and say, "Hey, the diff didn't match," and kind of give a bunch of debug logs and we'd look into that. We got to the point where we were looking at diffs.

[0:44:27.7] JM: You diffed two requests? Just two requests to the same service, to the same system?

[0:44:33.3] XC: We diffed — for one customer request, we ran it through both the old, the slow index and we ran it through the new fast index.

[0:44:40.9] JM: Yes.

[0:44:42.7] XC: The purpose of that was to make sure that the indexes matched because you have this real time updater system that could potentially drop and update every now and then or maybe there's some replication lag and you know, between clusters. Actually, what ended up happening is like, you get down to it and we actually found a bunch of bugs in the old system because we'd get these diffs and we're like, "Huh, this doesn't make any sense."

We got down to the point where we were finding differences of like five pros out of, you know, hundreds of thousands of candidates, right? Sort of like, "Oh okay, these five pros are slightly different in this adverse offset. How did that happen?" Actually, when we dug in, we actually found a couple of latent bugs in the old system that, you know, it's actually kind of hard to find if you don't have a comparison like this.

[0:45:34.5] JM: Did you have a different data store before you moved everything into Elasticsearch for the purpose of this new matching service?

[0:45:44.1] XC: Yeah, one of the primary data source that we used was DynamoDB and we had essentially created a bunch of kind of one-off indexes using just like simple key value stores. For example, our GeoIndex was using this concept of S2 cells, which is like a Google thing where you divide the earth into this grid and you can map from this grid to a list of pros.

So basically, you know, we had like a bunch of these indexes there but that was actually quite slowS so during kind of like a series of DynamoDB queries to a bunch of these one off indexes was actually pretty slow compared to Elasticsearch.

[0:46:20.9] JM: Tell me some of the challenges of running a big Elasticsearch cluster, or I guess, first migrating everything into an Elasticsearch cluster?

[0:46:29.9] XC: Yeah, one of the big challenges we had was basically some of the writes would just fail because we're trying to – especially our batch jobs overnight, we're just trying to write so much data. Pros can have a pro can have lots of services. For each service, they might have lots of preferences. Let's say, you know, have specific zip codes they want to work in.

While New York City metro area has like thousands of zip codes and then, you know, multiply that by hundreds of services that a pro could offer. Multiply that by, you know, may pros and you sort of get the idea and so like the volume of updates was actually quite large. We actually worked very closely with our SRE team to basically tune our clusters so that we could handle all of those writes.

[0:47:23.6] JM: What do you have to do to tune an Elasticsearch cluster to handle lots of writes?

[0:47:27.7] XC: I mean, it's basically the composition of the machines and the way that you do the writes. We use Spark as our offline data aggregation framework. Spark, if folks don't know, Spark is basically a map reduce framework and it's written in Scala and so we use Scala and Spark to kind of do these aggregations and so – Effectively it becomes a bunch of mac reduces, right? So you're basically doing a bunch of parallel writes into your cluster and so at some point, it's actually very easy to write a Spark job that blows up the cluster, Right?

For example, we do things like we'd add like client side throttling on the Spark jobs and just change the way that we were actually doing the map reduces to put less load on the cluster.

[0:48:19.5] JM: So if I understand correctly, you got a bunch of writes that are coming in that are updates to the pros. Like new pros, their preferences are changing, they're updating their profiles and whatnot. You use Spark as an in-memory working set where you're just throwing these writes into it and batching them up and then you're writing these batches to the Elasticsearch cluster out of the Spark cluster.

Do I have it right?

[0:48:50.6] XC: Yeah, so to be a little more specific, the online real time system actually takes updates from product front ends, puts the updates in a bunch of queues and then there is a streaming service that reads from the queues, decides what it needs to refresh, and then you know, reloads that data into Elasticsearch. The scale of that side is actually easier to deal with in some sense because we're not doing these like massive batch writes. That tends to be more of a steady stream of stuff.

What we also have is sort of this backup job because we, especially early on in this system, this is – the system is all of our revenues floating through the system, right? So it can't break. We need redundancy and so, you know, what we did was we also wrote a batch job that would basically take the data from the product services and use Spark to batch write that into Elasticsearch and we would run that every night. That kind of guarantee that any discrepancies that happened, any hiccups in the real time streaming system that happened during the day would not just build up over time, right?

Imagine like. you know, imagine you got a handful of timeouts on Elasticsearch every day and you never do anything about this. Over time, you could actually have a fairly corrupted index of matching data and so we didn't want this to happen. Especially while the system was early, we didn't have a lot of guarantees on the streaming side and so we had this backup, map reduce run every night where we would basically take a snapshot of the data and put that into index and do a diff. You know, that batch job was actually pretty big and that's the one that was challenging to get working well.

[0:50:41.8] JM: So you mentioned you work with the Cyber Liability engineering team to figure out how to get this Elasticsearch cluster tuned properly. Can you talk a little bit more — and I know we're running out of time, it's totally fine. But talk a little bit more about how SRE works at Thumbtack. Because this is a service that people are – it's a service where your uptime is super important.

The reliability of the service is super important because people are making their living off of this service and people are making expectations about the work that they can get off of this service. So how do you look at SRE, and maybe if you have an anecdote that illustrates the SRE team?

[0:51:23.1] XC: Sure, yeah. I mean, our SRE team is amazing and I'm not just saying that to get stuff from them. But they are actually amazing. I would say they are, you know, they're incredibly scrappy and they sort of create the foundation of, you know, they sort of create the foundation of the infrastructure that all of the teams use, all of the product and systems teams used to build out the product effectively. So, you know, for example, in this case, they really paired with us very closely to doing the Elasticsearch cluster and worked with our engineers to get the cluster working well.

I think the high level thing is – And this kind of reflects the culture here at Thumbtack is just, we try to be very collaborative across teams and so it's – we're not like companies where they're sort of this throw it over the wall culture with SRE where it's like, "SRE has to do this, let me send them a ticket and wait." Here it's sort of like we sit basically right next to each other and we just work all the time. Like right now, we're working on capacity and load testing for some of our systems and we kind of just sit together and work on it really closely.

[0:52:36.7] JM: Well let's wrap up. What have been the results from this migration from one form of your market place to the new form?

[0:52:45.3] XC: Yeah, it's actually been really exciting. You know, one thing I mentioned earlier was the old model is sort of inherently supply constrained because you're waiting for people to kind of sit there and click buttons and bid on things.

The new model is inherently a higher scale because we are doing the indexing work upfront. So we can return as many results as we have capacity for. There's two data points that really illustrate that. The one data point is a measure of what percentage of requests do we return at least three quotes on? Like three good quotes where again, the professional is responsive and is willing to go to your house.

So we have this metric and essentially the metric has grown over 2X in the markets where we have started to add instant match. It continues, if you just look at the graphs, it's just like widely diverging in the instant match market just continues to grow which is pretty exciting. The other metric that's interesting is you know, Thumbtack has – because of this bidding model, Thumbtack has historically been supply constrained, right? Because we're looking for really high quality, high intent pros and it's hard to get a lot of them.

Actually, in the new model, what's happening is pros are giving us a capacity of how many jobs they're willing to do. So they're actually giving us a budget that they're willing to spend every week. It turns out that we are now no longer able to spend the vast majority of those budgets and so for pros and instant match, I think when I wrote the blog post in August we were probably only spending about 25% of pros' budgets. We were only using up 25% of our pros' capacity.

That number continuous to drop and I believe yesterday I just looked at something like 12% now where like in this market, we're starting to look demand constrained, which is a great problem to have because we can go out and find more customers too. There's lots of other ways to improve our matching to use the capacity that we have and so, that's also a pretty exciting data point.

[0:54:59.2] JM: All right, well Xing, it's been great talking to you. This is a really fascinating top down discussion and it's highlighted more in this blog post that you made. We didn't get to nearly everything that you covered in the blog post, but I'll put that in the show notes. You've got some great diagrams that help to explain some of the things that may have been easier to understand visually than over audio. But thanks for coming on the show, it's been great talking to you.

[0:55:23.4] XC: Yeah, thanks Jeff. It was great chatting with you.

[END OF INTERVIEW]

[0:55:28.3] JM: Simplify continuous delivery with GoCD, the on premise, open source, continuous delivery tool by ThoughtWorks. With GoCD, you can easily model complex deployment workflows using pipelines and visualize them end to end with the value stream map. You get complete visibility into and control over your company's deployments.

At gocd.org/sedaily, find out how to bring continuous delivery to your teams. Say goodbye to deployment panic and hello to consistent, predictable deliveries. Visit gocd.org/sedaily to learn more about GoCD. Commercial support and enterprise add on's, including disaster recovery are available.

Thanks to GoCD for being a continued sponsor of Software Engineering Daily.

[END]