EPISODE 577

[INTRODUCTION]

[0:00:00.3] JM: Phones are constantly tracking the location of a user in space. Devices like cars, smartwatches and drones are also picking up high volumes of location data. This location data is also called geospatial data. The amount of geospatial data is rapidly increasing and there's a growing demand for software to perform operations over that data. Geospatial datasets are often massive, so it's non-trivial to perform operations over this data.

Geospatial data can consist of something as simple as a set of latitude, longitude data points, and a single lat/long coordinate pair can be enriched with information about what zip code that data point is in, or how far that data point is from the other data points in the set, or where the nearest coffee shop is in relation to that data point.

Ram Sriharsha created Magellan, a geospatial analytics library for Spark. In today's show, Ram describes the set of problems within the domain of geospatial analytical engineering. Ram also works as a product manager for Apache Spark at Databricks. What I loved about this episode is that I had a perception of geospatial data as being difficult to work with, or some domain-specific field like you had to be an expert in cartography, or geography to understand how to use this data.

We talked through some fairly straightforward applications. If you're interested in building some big data application, geospatial data seems a ripe field to just take a data set and start hacking around with and looking for opportunities, whether it's a business opportunity, or project opportunity. From that point of view, I think you may find some value out of this episode.

If you're looking for older episodes about Spark and lots of other big data processing systems, you can check out our back catalogue at softwaredaily.com, and you can also find our mobile apps there at softwaredaily.com which have all of our episodes in a searchable format.

With that, let's get to today's episode with Ram Sriharsha.

[SPONSOR MESSAGE]

[0:02:26.4] JM: The Casper mattress was designed by an in-house team of engineers that spent thousands of hours developing the mattress. As a software engineer, you know what kind of development and dedication it takes to build a great product.

The result is an exceptional product and when you put in the amount of work and effort that went into the Casper mattress, you get something that you'd use and recommend to your friends. You deserve an exceptional night's rest yourself, so that you can continue building great software.

Casper combines supportive memory foams for a sleep surface that's got just the right sync and just the right bounce. Plus, its breathable design slips cool to help you regulate your temperature through the night. Stay cool people. Stay cool.

Buying Casper mattress is completely risk-free. Casper offers free delivery and free returns with a 100-night home trial. If you don't love it, they'll pick it up and give you a full refund. Like many of the software services that we have covered on Software Engineering Daily, they are great with refunds.

Casper understands the importance of truly sleeping on a mattress before you commit, especially considering that you're going to spend a third of your life on that mattress. Amazon and Google reviews consistently rank Casper as a favorite mattress. Try it out. Get a good night's rest and upvote it yourself today.

As a special offer to Software Engineering Daily listeners, get $50 towards select mattress purchases by visiting casper.com/sedaily and using the code SEDAILY at check out, you'll get the select mattress purchases. If you go to casper.com/sedaily and enter the code SEDAILy at check out.

Thank you, Casper.

[INTERVIEW]

[0:04:32.3] JM: Ram Sriharsha, you are a product manager for Apache Spark at Databricks and you're the core maintainer of Magellan, which is a geospatial analytics library built on Spark. Thanks for coming on Software Engineering Daily.

[0:04:45.3] RS: Thanks to be here, Jeff.

[0:04:46.6] JM: You have been working on Apache Spark for a while. When did you start and what drew you to the Apache Spark project?

[0:04:53.5] RS: That's a good question. I actually started, I think when Spark was Alpha. It was a project out of UC Berkeley at the time. I had just joined Yahoo at the time I came across Spark, and I remember being at one of the early Hadoop summits where Matthew was actually talking about this project Spark. It was still not available, except as a library that was being built inside Berkeley, the amp lab at the time.

Right around that time, a group of us at Yahoo were actually trying to build an in-memory caching layer on top of hive, because the query latencies at that time were not fast enough for us to serve some needs in advertising. We're designing something that turned out had a good synergy with Spark, so we dropped our work.

I reached out to Reynold at that time, who was prototyping something called Shark, which was a sequel front-end to Spark. We collaborated together at that time, and that's how I got into Spark. It's been several years now.

[0:05:52.5] JM: Take me back to those days at Yahoo when you were operating with the – I guess, that was a pretty mature point in the Hadoop ecosystem, but what Spark did was to put this in-memory working set the distributed, resilient distributed data set. I think this was the core breakthrough of Spark that allowed people to have fast access to their large distributed data sets in contrast to the HDFS model of things, where everything is on disk, and you have to have a distributed MapReduce job to pull everything off of disk. It sounds like you were trying to solve those problems at Yahoo with a distributed caching layer. I imagine that is not an easy problem to solve.

[0:06:42.6] RS: Yeah, so a lot of us were trying to solve similar problems, so basically what happened is with Hadoop, it was possible to ingest a lot of data. Earlier, if you had to use traditional databases you had to have a lot of discipline around what data you can actually ingest, what you need to clean upfront, the time taken for insights would be very, very large, the latency to insight would be large, because anytime new data comes in, you have to clean it, you have to schematize it, you have to make sure that the schema is backward compatible, you have to size up your storage. There's a lot of things you had to do.

With Hadoop, a lot of that got simplified, because it could ingest massive amounts of data. The problem then was how do you analyze this data fast? Running MapReduce jobs on Hadoop, they had their own latency issues, especially because just startup times of these jobs are pretty high. Also, the query processing on top of MapReduce was written more for batch analytics as opposed to interactive analytics. It is a very different scenario that people are interested in.

Now often what people used to do was take the data from HDFS and roll it up into tables that was smaller and aggregated, then maybe have a middleware that queries this hierarchical tables and so on. That just wasn't working out for us, because again, the problem was we were ingesting data – a lot of data and we were ingesting new dimensions all the time, so we wanted to be able to query on that fast.

Having this in-memory caching layer would be super useful. Again, it brings its own challenges. What Spark did really well was not only have this concept of a resilient distributed data set, which was a good level of abstraction for working with distributed collections, it also had the ability to not have to checkpoint your data. You don't have to constantly keep reading from disk and writing back to disk during intermediate stages of a Spark job.

What you could do is use the lineage to be able to recompute at any point, if some node went down. Nodes going down in the middle of a computation was a pretty common problem, because people are running things on commodity hardware. It's all a lot of these problems and gave us a very elegant API that we could use. That's why we were interested in Spark at the time.

[0:08:58.0] JM: Yeah, that fault tolerance model of being able to recompute data by just looking at the query and recomputing the data set based off of that, it's one of those things where you see it and you're like, "Oh, why didn't I think of that?"

[0:09:14.4] RS: Yeah, I think to my mind that's like the code innovation. In memory caching, there's a lot of people who are thinking about it, but the APIs that made the in-memory abstractions easy to use, as well as the lineage concept that made it easy to recalculate from a given point, I think those were the two huge innovations in Spark.

[0:09:32.0] JM: I've heard that Mate's goals around Spark are – they're about making large-scale data processing more usable and more accessible and in the machine learning realm this is reflected in his efforts to think about how to improve the APIs rather than researching better algorithms for architecting neural nets.

He seems very interested in accessibility and more widespread adoption of data processing tools, because there's so much low-hanging fruit. For those of us who are maybe less familiar with the Spark ecosystem, what are the tools and the APIs in the Spark ecosystem that make data science more approachable?

[0:10:19.3] RS: Yeah, so a lot of people who are coming into Spark for data science want to scale out their queries on big data. Most people start with data science in R or Python. They're already familiar with concepts like data frames, Panda has a concept of data frame, R has concepts of data frames and so on. They have a very intuitive high-level abstraction for working with data.

What Spark does really well is especially with since Spark 1.4 where we started using data frames and little datasets, it allows you to have this table abstraction over data and be able to declaratively specify what are the transformations that you want to do. Then the underlying engine figures are the best way to do this transformation. Sometimes it may require distributing that query over multiple nodes, sometimes it may just require running everything on the driver and so on.

Now you as the end user has spared the complexity of figuring out how to do this efficiently. Whereas, the level of abstraction that you get is very similar to what you're familiar with using single node tools and so on. I think that's really powerful, because as data scientists, I think what you should really be doing this trying to derive new insights from data. You shouldn't be wrangling with data so much, and it shouldn't be so much of a headache to basically transform the data sets into features that can be used in machine learning and so on.

Oftentimes, the actual machine learning algorithmic component of any data science problem is pretty small. It's 90% of the time is just spent in wrangling the data. That 90 percent of the time if we can make it a lot more efficient, I think that's where the value for – that's where the low-hanging fruit for data science is.

[0:11:57.5] JM: You've been working on geospatial analytics problems. What is geospatial analytics?

[0:12:03.5] RS: Yeah. This is actually an old problem. Basically geospatial analytics is wherever you have geospatial data that is latitude, longitude, or even altitude, some point in space and you want to figure out some context around it. For example, if I give you a bunch of Uber rides and you want to figure out what neighborhoods did these cross, that's a geospatial problem, because you are trying to map points to some regions.

Similarly, you may have shipping containers and you may have even ships that are going from one place to the other and you may want to track what regions did they cross in the sea, so that's again a geospatial problem. There's a lot of problems where you have basic point in space information, and then you want to put some context around it which involves figuring out whether this point lies in a certain region, or whether you're looking at points nearest to a given set of points, or you're looking at the intersection of certain regions. Are you looking for – you have a bunch of points on a trip and you want to know what are the roads that this particular journey took place on. All those are geospatial analytics problems.

[0:13:03.2] JM: Geospatial sounds a big intimidating word, but I think we can boil it down to a simple type of data set. That is a latitude-longitude point that tells you where somebody is on a globe, or maybe not where somebody is, but something is. If I just have a set of lat/long points, latitude and longitude, what kinds of interesting work can I do with that set of points?

[0:13:34.2] RS: Yeah, so it really depends on the question that you're asking. As you just noted, latitude and longitude is just too simple a piece of information. Often, what you want to do is you may want to know a given set of points, do they lie on a particular road? A given set of points, do they cross multiple neighborhoods? A given set of points, are they all within a certain region of each other? For example, are they all within a mile of each other?

There's a lot of these questions that you may want to know. Depending on the type of question, the analysis may be a little bit different. That's basically what geospatial analysis is. The information that you start off with the latitude and longitude is actually a pretty small self-contained piece of data. The geospatial part of it comes in where you are trying to tie it in with some regions, or some roads, or some other geometric objects.

[0:14:23.5] JM: Right. I've done a number of shows where people talk about this enrichment pipeline, where they will take some minimal set of geospatial data. For example, coming off of cellphones, or coming off of a Fitbit, and they'll have a data enrichment pipeline where they take that data and they'll find the nearest store, or they'll find the state that this person is in, or they'll take a set of points and they'll find the acceleration that a person is taking across some domain. What are the typical patterns around enriching these lat/long data sets?

[0:15:06.7] RS: Yeah, it depends on whether you want to do this real time or not. These days, more often it's people want to do this real-time. You're getting some mobile information from either a person or a car, or something that's moving in time. You may want to know what are the regions in close proximity to it.

A typical ingestion pipeline for that, especially something that uses Spark would be take your data as it comes in and use Spark's framing a structured streaming to basically enrich these points. For example, you do a lookup to say is this point close to a particular neighborhood? Then you enrich the points information with that particular neighborhood.

You may dump it into a cloud storage for later processing, or you may just run dashboards out of it if you are interested in certain hotspot regions and stuff like that. A typical ingestion pipeline for this would be a streaming pipeline, followed by what I would call a geospatial lookup, though it's not a simple hash map lookup or something. It’s more complicated, but it's still a lookup. Then you enrich your data set with that and you offload it either for further batch analytics processing to a cloud storage, or maybe real-time processing through dashboards and other means.

[0:16:19.7] JM: This problem has been around for a long time. I think there's always room for new tools to be developed to solve old problems, but we actually have a new set of problems which is one, this real-time nature. Two, the volume of data that's coming in. Three, I think you could say the number of people who actually care about solving this type of problem. Since there's so many new kinds of customers that are solving this problem, you have to develop tools that are more accessible to smaller teams and a wider array of problems. What are the shortcomings of the geospatial tooling landscape, the shortcomings that led you to starting this project Magellan?

[0:17:06.7] RS: Yeah, great question. There is actually a lot of tools already available. I think the main problems that I see with a lot of them are we have new types of problems to solve and the volume and velocity of data is much, much higher than what the existing tools can do with it. The main problems that exist in the space are a lot of people use geospatial formats. The data formats that are in play for the geometric data sets is very old. By old, it means it's harder to parallelize reading this data, so it doesn't take advantage of modern distributed processing. Most tools in the space can only read this data from a single node. That doesn't scale at all.

The other problem is there's a lot of techniques to index datasets, so that when you are when you are dealing with latitude-longitude data and you want to look up what region it is in, then you can use indexing to do a fast lookup. The problem has a lot of indexes in the space are again single machine, single node type indexes, so they don't scale at all.

Mainly, there are two types of problems that were already there. Those who are legacy formats that just don't scale in terms of data ingest, legacy indexing techniques and databases that were did not scale to Big Data. The third problem is actually, we are also democratizing this geospatial analysis. Meaning, geospatial data is becoming a component of some bigger analysis, and more and more people are using that information.

When that happens, what tends to happen is they have an existing pipeline that's already a big data ETL pipeline, and now we want to enrich it with geospatial information. If you look at what tools exist already in this space, you always have to move data from TTL Big Data pipeline into this other tool, which may be specialized for geospatial; do your processing there and pull the data back into your original pipeline scope for the processing and so.

This adds a lot of overhead, both from the processing overhead point of view, but also overhead in terms of new tools that you need to understand how to integrate with the rest of your pipeline. These are the problems that I wanted to solve. I wanted to basically make sure that geospatial Big Data analytics could just be plugged into any modern ETL pipeline. When you do so, you also get scale and speed.

[SPONSOR MESSAGE]

[0:19:36.7] JM: We are running an experiment to find out if Software Engineering Daily listeners are above average engineers. At triplebyte.com/sedaily you can take a quiz to help us gather data. I took the quiz and it covered a wide range of topics; general programming ability, a little security, a little system design. It was a nice short test to measure how my practical engineering skills have changed since I started this podcast.

I will admit that, though I’ve gotten better at talking about software engineering, I have definitely gotten worse at actually writing code and doing software engineering myself. If you want to take that quiz yourself, you can help us gather data and take that quiz at triplybyte.com/sedaily.

We have been running this experiment for a few weeks and I’m happy to report that Software Engineering Daily listeners are absolutely crushing it so far. Triplebyte has told me that everyone who has taken the test on average is three times more likely to be in their top bracket of quiz scores.

If you're looking for a job, Triplebyte is a great place to start your search, it fast-tracks you at hundreds of top tech companies. Triplebyte takes engineers seriously and does not waste their time, which is what I try to do with Software Engineering Daily myself. I recommend checking out triplebyte.com/sedaily. That's T-R-I-P-L-E-B-Y-T-E.com/sedaily. Triplebyte, byte as in 8-bytes.

Thanks to Triplebyte for being a sponsor of Software Engineering Daily. We appreciate it.

[INTERVIEW CONTINUED]

[0:21:34.8] JM: What was the initial spec for Magellan when you started building this? This is again, for people who are just trying to wrap their minds around what we're talking about here. We're talking about a geospatial analytics tool on top of Spark. What was the spec for what you were trying to build with the initial product?

[0:21:54.6] RS: Yeah, a great question. I had basically three things in mind. One was it had to be able to be fast on the most common geospatial queries. The most common ones – there's a lot of very sophisticated geospatial analytics, but what is 80% to 90% of the problem is point in polygon computations. Figuring out whether a certain geometry is inside some other geometry.

I wanted to make sure that this competition could be done at massive scale, and it could be done extremely fast, and it would just laterally scale. That was the number one constraint. When I started doing this, project Spark SQL was still a very new thing. Spark was pretty mature at that time, but Spark SQL and data frames were very new. In fact, they just started coming out at the time I started doing Magellan.

I liked the idea behind data frames. The reason I liked it a lot was I wanted end users of Magellan to be able to express the computation they wanted to do; they wanted to do pointed polygon, so they could express it. I didn't want them to worry about how this calculation actually used them, because a lot of them don't have the expertise and may not even care about how to actually do this at scale. What they want to be able to say is do it for me. It's a level of abstraction that you get in SQL for example. You declaratively specify what you want to do with the data and an engine figures out how to do this optimally.

I wanted to keep the same level of abstraction and that's actually hard to do on top of Spark RDDs and the original primitives of Spark, but it's much easier and much more scalable to do this on top of data frames. That was my other spec, which is to make Magellan work on top of data frames. It has given us a lot of benefit actually.

It turned out to be a very good decision for the project. The last thing was I wanted people to be able to use this with any kind of data. I didn't want them to have to transform data into a special format to be able to start using Magellan, so we actually support all common geospatial formats out of the box. In many cases, we actually figure out how to parallel read them as well. There is really no overhead to using Magellan. You can just point at any geospatial data and use it.

[0:24:01.0] JM: Explain what data frames are.

[0:24:04.0] RS: Yeah, so data frames are basically, you can think of them as data with schema. You can think of them as a table, and each column has a type and some specification about the column. That's a high-level abstraction of how to think about a data frame. A data frame is just like a table with some schema. Now this table may be a logical abstraction for data that's distributed across a cluster, or data that's maybe sitting in parky on a cloud storage and so on. The abstraction as a programmer for you is basically that it is a table.

Now any operation on data frames would basically be table transformations. They would basically be either adding columns to this table, or doing something with the column on the table, or maybe aggregating, or grouping by a column and computing an aggregate and so on.

The nice thing about a construct this is it allows you to specify the calculation that you want to do. For example, if I wanted to compute the average age of a person in some data set of people in a data set, I would basically just say data frame dot average by some metric, maybe average by age. This is declarative. It doesn't specify how to actually do this computation, but under the hood, there is a powerful engine in Spark called catalyst, and catalyst figures out that because average is a associative and additive operation, I can actually do part of this computation on each node and then send the final computation to a single node for aggregation. This allows catalyst to do this query in a very optimal way.

Now if you are a programmer and you had to actually specify how to do this, that would be a lot of overhead for you to do it right. Whereas, keeping the level of abstraction at this level allows the engine to do all this for you. That's the way I think of data frames. Data frames is basically a table abstraction, and you can do whatever transformations you can do on tables using data frames. There is an optimizer that figures out how to optimally compute this on the fly.

[0:26:06.0] JM: What is it about the abstraction of the data frame that made it easier to build a geospatial analytics tool on top of it?

[0:26:14.3] RS: Yeah, that's actually a great question. It actually has its pros and cons. The pro is basically that every optimization that is available in catalyst and in Spark is automatically inherited by Magellan. For example, if you have a geospatial query that has a lot of skew, for example I am looking at taxi cab rides in New York State, but most of these are let's say green taxis. Now we know that green taxis mostly operate in Manhattan, which means most of the taxi cab rides are going to be in the Manhattan area and very few are going to spread out from Manhattan into the rest of New York State.

If you are looking at this as a geospatial query, it has a lot of skew where the points that belong into this neighborhoods in Manhattan are more computationally involved than the rest of the points. Now it turns out that because of the way Magellan is written on top of data frames, any skew algorithm that works with data frames automatically works with Magellan. If there exists a skew joint algorithm that works on data frames, Magellan automatically inherits it so we don't have to write one.

Similarly, if the set of neighborhoods is really small and the set of points is really large, one way to do this joint between points and polygons is to cache the polygon data set, broadcast it to each node and just stream the data, stream the point data set on each node. Now this is again a computation that we automatically leverage from Spark SQL because of the way Magellan is written on top of data frames.

Magellan didn't have to make this decision. We didn't have to be clever enough to figure out when data sets are small and large and when to use broadcast joins, versus when to use software joins and so on. That's the pro to the positive side of it. On the flip side, nothing comes for free. On the flip side, what we need to do is if you do this wrong, you can actually have a lot of overhead, because the data frame abstraction is great for things that are similar to SQL.

If you have geometries and you want to represent them in data frames, one of the main things you need to make sure of this that you are not unnecessarily serializing and deserializing geometries, because each time you serialize a large geometry, you're paying a lot of cost to do that operation. When you're trying to go pedal to metal on geospatial queries, you need to make sure that you don't serialize and deserialize unnecessarily.

Now the great thing about the way catalyst is written is it allows developers who want to extend catalysts that hook to be able to tell the engine when it's safe to serialize something and when it's safe to not say realize something. There's a lot of advantages to writing Magellan on top of data frames.

[0:28:48.7] JM: I want to give people a concrete example that we can explore through the lens of Magellan and through the lens of Spark. Let's say we've got a Fitbit-style company and we've got a hundred users and they've all got this lat/long data that's streaming into our Fitbit-style platform all day long and we want to get this data into a platform where we can do interesting geospatial things with it. For example, let's say we're going to serve these Fitbit users ad through some ad networks based on that lat/long data, not saying a Fitbit company would actually do that, because that would probably be some breach of the terms of service. Or maybe not, but I don't really know. Let's just say we've got this set of lat/long points that are streaming into our system all day long from a hundred users. How would we get that into Magellan?

[0:29:48.8] RS: Yeah, there is many ways to do this. Now one common way to do this would be as your data streams in maybe through Kafka, or Kinesis, or any streaming source, you could basically set up Spark structured streaming, or Spark streaming to read from that source so as the data is being streamed in there, is a Spark job that actually reads it on the fly. Every new batch of data is basically read, and within that batch maybe a common thing that you do is in order to serve ads, one of the common things you need to do is figure out what zip code a particular person is in right now, or what shops, or what advertising related things are around, so you can actually advertise the right – give the right advertisement, so things like this.

The next step in the process would be as you ingest this data, you need to – for this, let's say zip codes that we are interested in, then you already need to have data set of zip codes. Now zip codes are basically geometric regions. What you would do is you would already have a distributed data set in Magellan, of zip codes that you read from somewhere. Every point that comes in, you just have to figure out which zip code is this in. Now that's a joint. Basically, every new batch of data that comes in through the streaming pipeline, you join that small batch of data with your zip code data set using Magellan.

What you get out of it is for each latitude and longitude, the zip code. Now that particular data set can go downstream, where you analyze that for these zip codes, it makes sense to serve these ads versus some other ads. That's a very common scenario.

The other scenario is which is actually something that I worked on, is where you have similar data that comes in, except you want to serve search ads. This happens where someone is typing on a browser and they are searching for something. Now imagine going to Google and typing a search query and you want to – when you type a search query, you get not only the search results, you also get some search ads.

Now the search ads up making money for Google only when you click on those ads. You want to serve the ads that are highly likely to be clicked. The way to do that is to have the context information about where is this person currently as they are typing this query, so that you can actually serve the right ad. Because queries carry meaning that may depend on where you're querying from. For example, if you are querying from somewhere in Arizona and you type canyon, it's likely Grand Canyon, right? Ads for hotels, or motels around Grand Canyon make more sense.

In that scenario, what you would do is similarly you set up a pipeline where you read the data through a Spark streaming job, you use Magellan to index the zip codes, or the regions and figure out that this particular lat/long belongs to this particular state, or this particular city and so on. You have may have a machine learning algorithm downstream that actually uses this extra bit of information to figure out how likely is it that this particular ad is going to be clicked on, given that the person is in this particular region. Then you use the probability output from that machine learning algorithm to serve the app. That's a very common use case for geospatial.

[0:33:09.2] JM: We talk about the simple example, where you've just got a set of zip codes that are associated with lat/long perimeters, and we've got users that have lat/longs associated with them at any given time, and we've got to figure out what zip code they're in. Can you give me a perspective for what goes on in the Spark execution engine at a lower level to process those queries?

[0:33:36.5] RS: Yeah. If you have, let's say a single lat/long and let's say 100,000 zip codes, the naive way to figure out which particular lat/long belongs to which zip code is to look at the lat/long for all zip codes and to say does this lat/long belong to the zip code or not? Now zip code is actually a geometric region. The way the zip codes are represented, it's actually like a jagged poly. The analysis here would be given a point and given all these polygons, which polygons contains this point, right? Now the point within polygon calculation is actually not easy to do. The time taken for a point within polygon calculation depends on how jagged and how many edges the polygon has.

Now you can imagine US zip codes, the zip codes are not regular, so it's actually pretty complicated to figure out for each point which zip code does it lie in. The basic advantage of Spark is in paralyzing this problem. Now one way to paralyze the problem is to just say distribute the zip codes onto as many machines as possible, and then have each point go to all of these machines, where you check which point lies in which particular zip code.

That does turns out to not be the best way to do this, because you're shuffling each point to all of these nodes, and at some scale this becomes problematic. The real challenge here is in terms of figuring out which point should be checked in the first place against each of these polygons. You don't want to check every point against every polygon, because that's not scalable.

Now, this is where Magellan comes in, where when you read a data set using Magellan, not only does it distribute the polygons in the right way, it also indexes them on the fly. On the fly, it basically divides up the latitude-longitude coordinate system, which is the earth, it divides up that coordinate system into recursively into grids and figures out for each polygon what are all the grids that contain this polygon? Similarly for each point, what grid is that particular point in?

Once you have divided up these regions like this, now a point within polygon calculation becomes – I have these grids. Which grids overlap each other? That's basically it. Once you know that two grids are overlapping each other, you know that the points that belong to that grid and the polygons that contain that grid have to be examined at the same node. This not only cuts down the amount of computation that you have to do into other one, instead of looking at each point and each polygon, you're looking only at the polygons that matter.

[SPONSOR MESSAGE]

[0:36:13.3] JM: Azure Container Service simplifies the deployment, management and operations of Kubernetes. Eliminate the complicated planning and deployment of fully orchestrated containerized applications with Kubernetes.

You can quickly provision clusters to be up and running in no time, while simplifying your monitoring and cluster management through auto upgrades and a built-in operations console. Avoid being locked-in to any one vendor or resource. You can continue to work with the tools that you already know, so just helm and move applications to any Kubernetes deployment.

Integrate with your choice of container registry, including Azure container registry. Also, quickly and efficiently scale to maximize your resource utilization without having to take your applications offline. Isolate your application from infrastructure failures and transparently scale the underlying infrastructure to meet growing demands, all while increasing the security, reliability and availability of critical business workloads with Azure.

To learn more about Azure Container Service and other Azure services, as well as receive a free e-book by Brendan Burns, go to aka.ms/sedaily. Brendan Burns is the creator of Kubernetes and his e-book is about some of the distributed systems design lessons that he has learned building Kubernetes.

That e-book is available at aka.ms/sedaily.

[INTERVIEW CONTINUED]

[0:37:48.7] JM: Somewhere in that explanation, you said you were – Spark will distribute the zip codes intelligently. What do you mean by that?

[0:37:58.3] RS: There is a couple of things that need to be done here. There is a partitioning scheme in Spark where you can specify how to partition your data set. If you think the data set is going to be uniform, you could use a hash partition or a uniform partition. You could use other means of partitioning depending on how you expect your data distribution to look like. That's what I meant. That I meant is Spark allows you to specify how to actually partition the data among the set of nodes.

[0:38:24.7] JM: If you want to find, let's say you've got those zip codes partitioned and you've got a given lat/long and you want to find what zip code that lat/long coordinate pair belongs in, what is the – is that a linear operation, or is there some more efficient way of iterating through those zip codes?

[0:38:46.8] RS: Yeah, naively it's going to be a linear operation for every given point, you would have to look at all zip codes. The way this is done in Magellan is each of the zip codes would actually recursively subdivided into these grids. You can think of the earth as being just a bunch of pixels, a bunch of grids. The grids that we are talking about are something called geohashes. You can geohash the earth into a certain position, and depending on the position you get a number of boxes.

Now you can think of these polygons as being enveloped by those boxes. Once you can compute for each polygon what are all the boxes that taken together comprise that polygon, now you can actually hash partition on those boxes. You can say, “Oh, for each of the boxes that are contained in this polygon, I'm going to send that particular data set to a particular node.”

Similarly, for each of the boxes that a point is contained in, I'm going to point to the same node. Now what happens here is that only – the points only go to a particular node, the polygon has a chance of being there.

[0:39:50.4] JM: Yeah, so it's like a part – you can have basically partition keys. Well, I don't know if that's the right term for it, but you have to – you just have to do a calculation of I guess the words – Yeah, it wouldn't exactly be a partition key, it's the – like a range. I mean, how would you put it?

[0:40:11.0] RS: It is basically you are geohashing the earth. You're computing all the geohashes that a particular polygon lies in. You're computing the geohash of the point, and then you're just hash partitioning by the geohash. That is really technically what happens.

[0:40:24.1] JM: Geohash. Can you define that word?

[0:40:26.3] RS: Yeah, so this is actually – I don't actually recollect who came up with this, but basically geohashes if you think of the earth in terms of latitude and longitude coordinates, it basically is a square that goes from minus 180 to 180 and minus 90 to 90. If you think of a square as with these dimensions and you start recursively dividing them up into cells of size into four grids. You start initially with one grid, which is the entire earth, you now divide it up into four, you divide each of these four pieces into four recursively and so on.

What ends up happening is you get this recursive subdivisions of four cells each, and they taken together comprise the whole earth. Any point on earth can be specified by which cell is it contained in. The recursive subdivision process also gives you a way of calculating that particular each cell in a very efficient manner.

One way to calculate that particular cells coordinate is to basically do this thing called Z ordering, where you label each cell starting with 01, where zeros is to the left and one is to the right, similarly above and below. The first time you divide up the earth into four cells, you're going to get 00, 01, 10 and 11. Second time we divide, 00 is going to become 000, 001. You see what I'm saying? That recursive subdivision gives you a binary representation of every point in this earth.

Now if you fix a particular position, then the binary representation of that particular position is what's called a geohash. It's a bit technical, but basically you can think of it like recursively dividing up a given region of space into cells, into four cells at each time, and then figuring out what the position of that particular cell is in terms of a binary representation.

[0:42:23.4] JM: This is very interesting, because it's given me a notion for how efficient you can get with some of these geospatial operations that I was imagining all kinds of really inefficient ways of processing them, but it sounds you can get things to be pretty inexpensive too. If you wanted to find – another example I was thinking of is let's say you had – you wanted to find – let's say, you've got you got these 100 users, or a million users, or however many users we want to be talking about, and we want to find a geospatial circle with a one-mile radius that has the most users in that circle.

Let's say we want to make, we want to send some message to all of these users that are in this one-mile radius and say, “Hey, there's a ton of users around you. Maybe you can socialize with them,” like some application like that. How efficient can you get that operation?

[0:43:20.9] RS: Yeah, so this is actually an interesting question. If want an approximate answer, you can get pretty very efficient with it. The exact answer is actually – is an interesting problem. It's the best circle problem, or something, it's called – I forget the name of it, but it has a solution which is out of N squared. There are endpoints it takes out of N squared time. If you want to be close to accurate, if efficiency is what matters the most, one way to do this is to basically say just divide up the region that you're interested in, again the space that's comprised of those points, divide them up into either circles around a given set of points, or even just the way I described it, just divide them up into grids.

Then all you do is for every point, which grid is it in? Then just count the number of points that lie within a grid and look for the maximum. This is actually a good enough solution. Most commonly, this is the thing that people do. What it is is basically dividing up the region into pixels and then doing a heat map of all the pixels.

[0:44:22.5] JM: We've been giving some toy examples for how people use Magellan. Can you give some – are there any real-world stories you have to tell of people adopting Magellan and finding practical use from it?

[0:44:34.7] RS: Yeah, so most often, people use Magellan where they're doing this point and polygon queries on very large data sets. Data sets ranging from hundreds of millions of points, to billions of points. Polygons data sets of 100,000 polygons and more, this data cannot even be done on a single machine in any scalable way. I see very often people with that data sets come and use Magellan.

One of the interesting problems that I did see people doing with Magellan, which was pretty interesting for me was because right now, Magellan is optimized a lot for point in polygons, or polygon intersections and geometric intersection algorithms. The use case here was this particular organization had millions of data points, so basically they were collecting information about people's driving behavior, so just looking at how people are driving and what roads that they are basically taking. This could be roads. It could even be bike rides. They're just looking at mobile information and mapping them to roads, or pathways, or bike rides and so on.

The challenge here is they have a lot of this. They're taking data of a lot of people. Also, geospatial data, the data that they get in the wild is actually not very accurate. There's a lot of things that can influence the accuracy of latitude and longitude readings, which means even if a car is driving on a particular road and you take and you get information from that car, you have an IoT device on the car that actually gives you information about what its latitude-longitude is at each point.

It's not guaranteed that each of those points are going to be close enough to a road that you can naively map it to a particular road. What they were trying to do was to actually map all these points to meaningful rides as accurate a way as they could and at scale. That turned out to be a very interesting use case of Magellan, because while Magellan is designed to all these joins and this stuff, it wasn't a priority design to deal with inaccuracies and mobile readings and so on.

Usually you have to employ pretty clever algorithms to mitigate this inaccuracy, and algorithms that people employ in the space are things like map matching. A scalable map matching algorithm while it's still not yet available in Magellan, we could do something simpler that solved the problem for the customer in this scenario. That was a pretty interesting use case.

[0:47:04.4] JM: Can you explore the alternative architectures that people could potentially use to process their geospatial data? There are databases that are built to process geospatial queries, I believe. I could be mistaken. There are none that come to mind specifically. Although, I think I saw one at Strata recently, I think like ConNetica perhaps.

[0:47:32.9] RS: That is actually interesting – there is multiple ways you can solve this problem. ConNetica actually comes to mind, because they use GPUs. They actually have a specialized geospatial database that uses GPUs. There is also [inaudible 0:47:46.7], which is a single-node database so it doesn't really look it beyond a particular scale. They are a geospatial database also.

There are research databases that are some of the fastest single node ones, but again, I don't know anything they're actually scales. In the Spark ecosystem, there is actually a lot of tools out there of varying levels of maturity. Magellan has one, that's the tool that I develop, but there is also a lot of other things like GeoMasa, GeoSpark and so on.

The broad categorization I can do here is there are geospatial engines like Magellan, which are not custom built. They are not built from the ground up to be geospatial engines. They take existing distributed engines like Spark for example and try to specialize them to do geospatial. That is one way to build this engine. Magellan does it very optimally. There are also other projects that try to do the same thing in Spark.

However, things like ConNetica and certain research databases for geospatial and so on, they take the other approach, which is they build a database from the ground up to do geospatial analytics. I think both of these are valid approaches and they have their own challenges. If you're building a database from the ground up to do geospatial, you have the opportunity to do a lot of optimizations, both in the way you lay the data out, as well as in the in query processing engine itself, to be highly optimized for geospatial queries.

On the flip side, there is – you have to transform the data into your format, you have to move data from one system to the other, you have your own query language, you have your own query DSLs or whatever it is, and then there's an impedance mismatch between using that and the rest of the pipeline. That's a big challenge.

Whereas, in Magellan you don't have that challenge, so you don't really have to move data from one place to the other. We can actually read data from anywhere. Since we work with Spark SQL, we can actually read any type of data. Spark has support for vast variety of data sources. Once we have ingested the data, we have a common schema, so everything looks the same once it's in the system.

Similarly, anything that Spark SQL does we automatically get for free. Any complex queries and complex filters and things like that, you may apply on the data. It's dealt with in as optimal way as possible, which would be very hard to do with the other systems. On the flip side, for a tool like Magellan, the challenge is just how fast can you get, right? We've decided not to build a specialized geospatial engine from the ground up, but actually build it on top of Spark, which means it has its own overheads. The question is how fast can you make it given that constraint. They are solving the same problem, but they have two different – they have their own challenges. I think that's the way I think about it.

[0:50:31.8] JM: Okay, data layout. We've done shows on columnar versus row-based data stores. What kinds of data layout can you use to optimize for geospatial queries?

[0:50:47.3] RS: Yeah, that's a great question. Geospatial queries are not any different than a lot of other big data analytics query. We benefit from columnar data layout as well. In fact, I suggest to a lot of my – a lot of people who use Magellan to actually use Parquet for example and other columnar data formats. Traditionally it has happened that geospatial data formats have not been columnar, which is been a problem because it's hard to efficiently read these data formats. Once you get it into Magellan, you can actually output it as a Parquet or a columnar format. That's what we suggest to a lot of people as well.

Now the other interesting thing about geospatial is you may often want to – the nice thing about databases is they have this these things called indexes, right? Indexes make it easy for you to fetch only the blocks of interest for you. When you have something like Spark SQL, Spark SQL doesn't actually have indexes. What it does is it actually has the ability to push filters down to your data sets, so something like Parquet, which actually has column metadata. All these columnar formats have column metadata on them, so you can actually use the column metadata to say this particular file contains a max latitude of some number and a min latitude of some number.

My point cannot have any relation to this particular file, because my points latitude-longitude is different. It doesn't rely in the fringe. I can use the column metadata to actually prune the files and say, I don't have to actually look at this file at all. I know that my query is not going to be satisfied by this file. You can leverage all that if you are using columnar formats with Magellan.

Magellan also outputs some metadata to these files. Things like indexes are also output by Magellan. When you are running a query on top of these file formats, Magellan can be extra intelligent to say, we have already pre-computed indices for this geometry and my query cannot be satisfied by this file, because the indices in this file do not satisfy that query. There's a lot of things we can do here to avoid reading data necessarily if you use columnar formats.

[0:52:55.7] JM: There are a lot of applications for geospatial analytics today, just by virtue of smartphones. We're now all walking around with devices that have extremely useful geospatial tracking data, that is being emitted to servers all around the world and all kinds of different companies have these kinds of queries that they want to run.

As we get to a world with drones and more IoT sensors and self-driving cars, the number of geospatial data points that are getting recorded is going to balloon. Do you have any ideas for other tools that need to be built, or do you have any visions for what this future of geospatial analytics is going to look like?

[0:53:42.9] RS: Yeah, I think you're totally right. I think the scale of the problem is going to be much bigger very soon. The other thing that's going to happen with drones and all that is right now two-dimensional geospatial analytics is still the most dominant type of queries people run, but with drones you can easily imagine the third dimension, so altitude of the drone is an important dimension as well. Three-dimensional geometry is going to become pretty important, I think.

The scale of the problem is just going to be very, very larger. At systems like Magellan are actually designed for that scale. You could easily imagine getting 10x or a 100x larger volumes of data than we have today, and to like Magellan will perform just fine. Magellan today doesn't use three-dimensional points, or three-dimensional data. There have to be a few changes to tools like Magellan to really support three dimensions well.

[0:54:33.1] JM: You're working at Databricks. What are you working on at Databricks? Tell me about some your progress at the company.

[0:54:39.9] RS: Yeah, so I actually started Databricks about two years and maybe two months back. I primarily work in – I'm a product manager of Databricks. I manage products. Some of the things I have worked on a database are the performance team, where we build cloud – basically we make Spark fast on the cloud. There's a lot of performance engineering that goes into it and I used to manage the product for that.

These days, I'm working on scaling genomics data sets. Genomics is the other big – I consider it the real big data problem, where you have massive amounts of genomics data and this data sets are going to explode in the next few years, but computers not caught up at all. Very similar to geospatial, this is yet another – this is another problem that actually is getting very important.

Unlike geospatial, there isn't good existing tools out here. A lot of it, you have to engineer from the ground up for scale. This is the problem that I'm working on these days. How do you scale geospatial? How do you scale genomics to the cloud?

[0:55:41.7] JM: Yeah, unfortunately for you on the genomics space, there has not been a widespread effort to build ad tech solutions around genomic data sets quite yet.

[0:55:53.0] RS: Oh, no. I actually think genomics is fascinating, because if you do this right, you can personalized medicine and a lot of things that benefit society are right at the corner. I think it's probably the most fascinating problem to solve.

[0:56:05.6] JM: I don't disagree with you. I just mean that people have been studying geospatial at scale for a while now, because there's a lot of profits interest there, but less so unfortunately for the biology point of view.

[0:56:18.9] RS: Who is it that said that the best minds of our generation are working on ad tech?

[0:56:23.3] JM: Well, that was Jeff Hammerbacher, while he was trying to lure people away from ad tech to come work at cloud era.

[0:56:28.8] RS: Yeah. You got it right. Yeah.

[0:56:32.5] JM: Well, the greatest minds of every generation are always speaking in terms of self-interest, so I'll say that. Not that I dislike Jeff Hammerbacher, but I wasn't a huge fan of that quote.

Anyway, Ram thanks for coming on Software Engineering Daily. It's been really great talking to you.

[0:56:46.8] RS: Thank you. Thanks for having me.

[END OF INTERVIEW]

[0:56:51.4] 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 plugins. Use the value stream map to visualize your end-to-end workflow. 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 long-time sponsor of Software Engineering Daily. We're proud to have ThoughtWorks and GoCD as sponsors of the show.

[END]