Ben Hindman on Apache Mesos

Jeff Meyerson

**EPISODE 235** of Software Engineering Radio features a conversation between host Jeff Meyerson and guest Ben Hindman, cocreator of Apache Mesos, an open source project that abstracts away many of the hassles of managing a distributed system. Ben spent time as a lead engineer at Twitter and now works at Mesosphere. Topics not covered here because of space include a high-level explanation of Mesos and a granular discussion of the abstractions Mesos provides and how to leverage them. Download the entire episode at www.se-radio.net.

—Robert Blumen

Jeff Meyerson (JM): In a talk you gave, you began with the statement, “You’re building a distributed system,” paraphrasing Mark Cavage. Before we dive into Mesos, I want to get a feeling for how you think about the modern distributed-system architecture.

Ben Hindman (BH): Right now, pretty much everybody is building a distributed system. Most people don’t necessarily realize that they’re building distributed systems, but it’s inherent in most of the software designs we have today. Maybe it’s because the applications we’re building need to be able to handle a lot of traffic, and you need to spread them across a lot of machines, or you need to have many of them running at the same time to deal with failures. What’s really interesting is that distributed systems are not taught in most formal computer science classes. Most people pick it up from their colleagues or books or websites. And it’s tough stuff—it’s kind of like parallel computing except there are failures, so it’s even harder.

I think we’re at a place where you still have to do your own memory management, and not with C or C++. I mean when you actually had to pick which regents of memory you wanted, because the operating system wasn’t providing any abstractions. Eventually an operating system provided VMMs [Virtual Memory Managers], which made things a lot easier. That’s kind of where we’re at today with distributed systems. Everybody has to do most everything manually, and almost everybody has to reinvent the wheel, which makes all the distributed systems we build that much more brittle, that much harder to build, and that much more likely to fail. But I think it also means there’s a big opportunity to make building distributed systems even easier.

JM: Can you tell us about how Mesos acts like the VMM of the datacenter operating system?

BH: There are some things that most every distributed system ends up having to redo. For example, if you want to launch an application or process on another machine, you program it yourself for the most part. One example is leader election. Most distributed systems have some notion of who the current leader or coordinator is. You need some pretty sophisticated algorithms...
for actually determining who can be a leader. It’s even tougher in the face of failures. There are tools like ZooKeeper and sed, which make the problem much easier. But there are still tricky cases you have to solve when using those systems.

Another example is dealing with failures. What does it mean when I’ve distributed my work and I have some tasks on some machines, and then those machines fail? Do I actually know that the tasks have failed? Can I say, that task has failed, so I’m going to launch another task reliably? What happens if that machine comes back and my task is actually still running? Do I need to take care of the fact that I killed that task because I launched it on a different machine? That’s a pretty typical problem in distributed systems.

These are the types of things we try to provide as an abstraction layer with Mesos. You don’t have to think about how you’re going to get your task to a different machine with Mesos. You just tell it, “Run this task using these resources; those resources are tied to a particular machine,” and then Mesos takes care of getting the task there, starting the task, and watching it while it’s actually running on that machine. Mesos provides primitives that somebody building a distributed system can take advantage of. They don’t have to rebuild the code themselves, so they can focus on their business logic of the distributed system. The goal is to make people more productive. That’s really what operating systems and kernels provide—abstractions that make it easier for people to build their software and systems without everybody having to build the exact same thing.

I’ve distributed my work and I have some tasks on some machines, and then those machines fail? Do I actually know that the tasks have failed? Can I say, that task has failed, so I’m going to launch another task reliably? What happens if that machine comes back and my task is actually still running? Do I need to take care of the fact that I killed that task because I launched it on a different machine? That’s a pretty typical problem in distributed systems.

I think the VMM is an interesting example because so many people never believed it would be successful because there was no way that something could actually abstract something like this away and still have good performance or always be correct. But computer scientists pushed it and said, “This is the way we should actually be abstracting this and writing our software in the future.” And now it’s something we all take for granted. I think there are some similarities with distributed systems as well. I think we’ll look back and say, “There were a lot of abstractions we needed to put in place that really helped us and made us more productive, and made our codes safer and more robust.”

JM: When you were working on Mesos, what ideas did you want to be able to leverage once you had these problems abstracted away? What solutions did you want Mesos to make available to programmers?

BH: When we started the project, we were really focusing on Hadoop [an open source software framework for storing data and running applications on clusters of commodity hardware]. Our initial goal was to be able to run multiple Hadoops at the same time. We were focusing on cluster managers of the past because there’s been a lot of cluster management and resource management research. In fact, PBS [portable batch system] dates back to 1991. Could we use these things to achieve our goals of running multiple Hadoops at the same time? We realized there was an opportunity to make cluster managers much better. We thought, if we’re going to be putting this level of abstraction in place between the machines that are actually running and the distributed systems, could we make it even easier to build new distributed systems?

That’s why one of the first distributed systems we built on Mesos was Spark, which is now an Apache project that’s really popular for big-data analytics. The really cool thing about Spark is that when we were building it, we decided to put everything in memory. Often with MapReduce, you’d run one iteration of your job, and it would read everything from disk and then write back to disk. When you’d do the second iteration of your job, you’d have to read everything from disk again. With Spark, we kept the output in memory so that when you run a second instance of your job, it can read from memory, which is much faster. So, we could do these iterative jobs that were typical in machine-learning applications 100 times faster than Hadoop. Here was this new distributed system that was using fault tolerance and was highly available, and we could run it across thousands of machines at around 1,000 lines of code. Spark
has evolved considerably since that
day, but a lot of the core functional-
ity of cluster management came from
the first 1,000 lines of code.

We were really thinking about
analytics at the time, so we were
thinking about MPI [message pass-
ing interface], which is still used pretty heavily in national labs and at
universities. Then we started think-
ing, “Hey, we could run stateless
services, web servers, Web services,
anything that’s in the service-oriented
architecture,” and that’s when Twit-
ter picked up Mesos. They were in-
terested in decomposing their appli-
cation into something like an SOA
[service-oriented architecture], and
they realized they could use Mesos
to run all these services. We’re en-
abling you to run any distributed
system, any application, somewhere
in your datacenter.

JM: We’re starting to get all these
building blocks of distributed sys-
tems, such as Kafka, Storm, and Zoo-
Keeper. Mesos is leveraging these; for
example, it’s leveraging ZooKeeper
to do some leader election. And then
you also have things such as Docker.
So where are we going? What will a
distributed system look like in three
to five years?

BH: I’d like to see us at a point where
we can figure out the POSIX [Portable
Operating System Interface; a set
of standard OS interfaces based on the
Unix OS] interface to all these primi-
tives that you’d expect to be provided
for a distributed system so I can just
build my application, assuming I have
services like Pub/Sub, message queue,
coordination, or leader election. If
one organization chooses to use Zoo-
Keeper and another organization uses
sed, that’s totally fine. I can run my
application in both places. That’s
what we don’t really have today that
is critical for going forward for our
industry as a whole—to be able to re-
use all these systems effectively.
When you think about POSIX
today, there are a lot of things we take
for granted. We have pipes, so we can
pipe data from one application to a
different application. We have a file
interface, so we know we can share
information between applications
through files. We have the concepts
of processes, threads, and allocating
memory. For the most part, we can
can then run those applications across any
POSIX system, whether it’s my Mac
laptop or my Linux server machine.
There are exceptions, and there
are some ways in which POSIX has
been bent. But the basic idea of hav-
ing an interface behind which there
is many different implementa-
tions of the actual interfaces leads
to us to be able to build more appli-
cations that we can share. Today, if
you build a distributed system in one
organization—let’s say Yahoo! builds
ZooKeeper—which is then run in an-
other organization, it’s difficult to fig-
ure out how to make it work in some-
one else’s setup based on the way
they want to do things. And then that
other organization says, “This isn’t
worth the effort; I’m just going to
go ahead and go build my own.” I’m
looking forward to a future where
you can build a distributed system in
one organization and start running it
in another organization, just like how
easy it is to build a Linux app at one
company and then start building that
same app at a different company.

JM: What are the missing pieces?
What’s the disparity between where
we are now and that optimistic future?

SOFTWARE ENGINEERING RADIO

Visit www.se-radio.net to listen to these and other insightful hour-long
podcasts.

Recent Episodes

- 240—Cédric Champeau discusses the latest and greatest in the Groovy
  language with host Josh Long.
- 239—Host Josh Long speaks to Pivotal’s Andrew Clay Shafer about the
  state of platforms as a service.
- 238—Author, speaker, and independent consultant Linda Rising talks to
  host Johannes Thönes about the agile brain.

Upcoming Episodes

- 242—Host Sven Johann talks to Dave Thomas about how to fix and live
  with legacy systems.
- 245—Author John Sonmez speaks with host Charles Anderson about soft
  skills: how to market yourself and manage your career.
- 248—Axel Rauschmayer joins host Johannes Thönes for a discussion of
  the history of JavaScript, the standardization process, and JavaScript as
  a language.
BH: I’m obviously pretty biased. But I think one of the big ones is having that first layer that abstracts away the machines and provides these primitives for distributed systems. Our goal with Mesos is to be that abstraction layer. You build all your distributed systems on top of that, and then anybody running Mesos can take the application they’ve built in one organization and run it in another organization that’s also running Mesos.

A great example is one of the distributed systems we built on top of Mesos, called Chronos, which is a distributed cron. Cron is a big pain for a lot of folks because they’ll set up cron on a couple of machines, but when that machine fails, they have to set it up on another machine. Chronos is a solution for that, and it only runs in Mesos. Chronos was built at Airbnb, and they ran it on Mesos. Another organization that was also running Mesos was able to pull down Chronos and deploy it in their cluster immediately. It was as easy as downloading an app on an iPhone or on Android.

The abstraction layer gives people the ability to build these things and then run them in their datacenters. The goal of Mesos is to make it easier to build distributed systems by abstracting away tough things like dealing with failures, distributing tasks, and leader election. To be honest, we’re not 100 percent there. It’s still pretty easy to build Chronos, but we need to make it even easier. For example, a lot of people don’t use threads, because although it’s a good abstraction to expose, it’s lower-level, and you need even higher-level abstractions on top of threads. That’s one of the things we’re doing at Mesosphere today—this is the first step in making it easier to build distributed systems by exposing the primitives, but we’re going to make it even easier to use those primitives with higher-level SDKs [software developer’s kits] and higher-level things on top. I think that’s going to make even more people want to build distributed systems on top of something like Mesos.

JM: Where can people go to find out more about Apache Mesos and about you?

BH: You can go to mesos.apache.org, which is the website for the Apache Mesos project. You can also go to mesosphere.com. We have a bunch of documentation there. We even expose a lot of prebuilt packages for Mesos that you can download from the Mesosphere website. You can also follow me on Twitter @benh. ☀️

JEFF MEYERSON is a software engineer and host of the Software Engineering Daily podcast. Contact him at softwareengineeringdaily@gmail.com.