Four Principles for Engineering Scalable, Big Data Systems

featuring Ian Gorton interviewed by Suzanne Miller

Suzanne Miller: Welcome to the SEI Podcast Series, a production of the Carnegie Mellon University Software Engineering Institute. The SEI is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University in Pittsburgh, Pennsylvania. A transcript of today's podcast is posted on the SEI website at sei.cmu.edu/podcasts. My name is Suzanne Miller. I am a principal researcher here at the SEI.

Today I am very pleased to introduce you to Ian Gorton, a senior member of the technical staff at the SEI whose work research focuses on big data. We have spoken with Ian before on other topics related to big data, so this is continuing in that realm. In today's podcast, we are going to be talking about four principles that he has come up with for engineering scalable, big data systems. This is very exciting to me because whenever we can move from the specific to generalized principles that can be applied in multiple contexts, that is huge. It is something that, if you are not in research, you may not appreciate just how big it is that we've gotten to this point with big data. So, I'm very excited about this.

Before we get into that, a little bit about Ian. Ian's research explores issues related to software architecture at scale. This includes designing large-scale data management and analytic systems and understanding the inherent connections' intentions between software, data, and deployment architectures in cloud-based systems. Since earning his doctorate in 1988, he has worked in academia, industry—both Microsoft and IBM have had the benefit of his services—and government funded R&D labs including CSIRO [The Commonwealth Scientific and Industrial Research Organisation], PNNL [The Pacific Northwest National Laboratory], NICTA [(National ICT Australia) is Australia's Information Communications Technology (ICT) Research Centre of Excellence], and since 2013, the SEI. He likes acronyms. He has also authored the book, *Essential Software Architecture*, which is published by Springer Verlag. Welcome today. Thanks for being here. I am excited about this.

Ian Gorton: Thanks, Suzanne. Thanks for inviting me.

Suzanne: We have done this before but reintroduce the idea of big data to our audience, and help them understand why big data so important today.

Ian: Sure. Yes, it does seem that most of the solutions to the world's problems can be solved by big data if you listen to the hype that is around at the moment. You will hear people talk about the Vs, the three or four Vs that are associated with big data. The problems of the volume of the data; the velocity of the data, or its rate of change; the variety of the data and the different types of data that you have to analyze; and also its veracity, kind of the integrity of the data, the quality of the data. All of this is kind of amorphous. It really comes down to one really simple problem, which has caused this paradigm shift from our traditional data processing world to the world of big data. It is now simply that there is an availability, very large and complex data sets that have become impossible to process with traditional databases and data processing techniques. This has forced us to look at new ways to process this data.

Suzanne: The fortunate thing is, at the same time this evolution has been going on, we are also seeing evolutions in hardware processors and multi-processor cores and things.

Ian: That is right.

Suzanne: So, the time is right for us to actually be able to look that this issue of big data in the context of architecture, hardware, etcetera.

Ian: Yes. That is right. There's a fusion of software and hardware here that is needed.

Suzanne: So that brings us to then, what are some of the challenges that architects in particular and engineers face in designing and building these systems? Because everything starts with the architecture when you are looking at this level of data.

Ian: Sure. That is very much our perspective, the challenges of building these systems. If you look at some of the surveys that are around and look beneath the hype, you find that roughly 50 to 60 percent of all big data initiatives are not succeeding at the moment. The major influences on this are essentially the lack of skills that are available to deal with the new technologies and the new designs and the new architectures that these systems require. All big data systems are inherently distributed. They require a whole new set of technologies that will exploit the distributed computing infrastructures that a big data solution needs. These skills are not widely disseminated across the industry. They simply just haven't had to be addressed before in the problems that most people face. So, this lack of familiarity with the advanced technologies that are available for big data problems is really the essence of some of the difficulties that we face with big data.

Suzanne: So, that is another reason why having some principles that people can apply that are moving into this area have at least somewhere to look.

Ian: That is right.

Suzanne: It is like, *OK. I need to learn about this principle*. Where do I go for that? So, why don't you tell us about those four principles. What is the first one?

Ian: Well, this first one is actually really intuitively very simple. And it essentially says you can't scale your efforts and costs for building a big data system at the same rate that you scale the capacity of the system. And so let's illustrate this with a simple example for a minute. A century or so ago when the telephone was becoming prevalent in the United States, in the old days, people used to have operators who would operate the switch. So, you would make a call, and someone would connect you, manually connect you to the exchange so that you could make your call. Someone estimated that as telephones became common in every household, about 30 percent of the U.S. population would be working as telephone operators. So, this obviously is not scalable and, hence, the invention of automated switching systems.

In big data systems we face exactly the same problem. If I estimate that the capacity of my system in terms of, maybe, the volume of data it's going to store or the amount of processing it is going to carry out, is going to quadruple in a year, I am not going to be very popular with the person who is funding my project if I tell him I have to grow my team by a factor of four in that same year, and the costs to operate the system are going to grow by a factor of four.

I have to be smart, and I have to introduce efficiencies that mean that I may be growing my costs and effort in some sort of linear way, maybe by 5 or 10 percent, to deal with this growth of the factor of four.

Suzanne: So, you have to find leverage points.

Ian: You do. That is right.

Suzanne: That has got to be a challenge in this arena because it is so new.

Ian: It has got to be a conscious thing that pervades through the design choices that you make. So, if you are going to expand your data collection, for example, what is the cost of adding a new node to your database, so you can store more data? If someone has to physically walk in and install a system and copy data around, this is a really expensive thing to do. You need to just automatically install a node and the data just moves automatically to that.

Suanne: So, some of the kinds of data processing conventions that were the standard way we did things in the past really have to change to allow for that capacity not to be on a linear kind of function with the big data volume that you want to deal with.

Ian: Yes, these things simply don't hold anymore. So, you really have to seek out these efficiencies to ensure that your costs grow as slowly as feasible, and your capacity grows as fast as your requirements dictate.

Suzanne: So, that leads us to, what is the second principle then?

Ian: The second one is very much tied to the issue of the technological complexity that big data systems require. There is a huge landscape of solutions from databases to analytical software and stream processing that people can use for building big data systems. Of course, this is all made more complicated by the marketing information the vendors give you and the hype behind many of the open source technologies. So, it is a really difficult place for a team to navigate through this technological space to choose the right solutions. The heart of the problem is you have got to understand; the more complex a solution, the less it will scale.

So, simplicity is essential for scaling. This is why, for example, we've seen the introduction of weak consistency models in databases that have replaced the strong consistency models that are prevalent in relational databases. Strong consistency is a great thing to have. But, it is expensive, and it is difficult to implement in a distributed scalable system. Weak consistency enables us to scale our data much more efficiently. Of course, there are costs and tradeoffs to make that you have to, maybe, embed more application logic to handle this weak consistency. But, essentially weak consistence problems will scale better than strongly consistent solutions.

Suzanne: So, you are talking about satisficing solutions*: I'm not going to get perfect consistency. I need to have just enough consistency in the case of this attribute to meet whatever the need is for that application, that business function that I want to provide.*

What we are talking about is [that] a lot of the function of big data is to provide us with sort of aggregate views of things. One of the aspects of big data is that, at a certain point, a lot of these consistency issues actually are taken up in the law of large numbers.

So, you can afford to have less consistency or other attributes you may be able to satisfice on. So, simplicity, reducing the complexity, is really not just about not introducing it but reducing complexity of current approaches as part of that principle.

Ian: Sure and we see it with processing of big data. Many people are using Hadoop now. I think many don't realize that Hadoop is a batch-processing system. You fire up a Hadoop job, and when it finishes is dependent on so many factors that it is not going to finish quickly. If you need fast responses from your data, Hadoop really probably isn't the solution.

There are others solutions that you can look at. You have just got to choose carefully and wisely. When you are met with a conflict on a decision, think about the underlying principles about what's happening in that mechanism, and choose the simplest if you want to scale.

Suzanne: OK. That actually applies to a whole lot of life doesn't it?

Ian: It does, yes. Principles are like that.

Suzanne: What about the third principle, what comes next?

Ian: The third one is concerning state management. Many people who build systems are used to using frameworks such as Hibernate or JEE which promotes[this]: When you query a database create things called stateful objects within your middle tier or within your business logic. And, these things are super convenient for programming because all of the data that you need to satisfy the set of queries that are used is going to submit, is already in memory. You can quickly access it. You don't have to go to the database. But, when you start to scale things, statefulness is a real problem because it's really hard to balance the load of the stateful objects across the available servers. It's difficult to handle failures because if you lose your state what do you do? You have to somehow reproduce it.

It's difficult to scale systems by just adding resources because you've got all the states lingering around in your system. So, the simplest solution, again, tied back to the second principle is to use stateless objects and stateless techniques; whereby, a request comes in and you somehow retrieve the data either from a database initially or maybe from some sort of caching mechanism that you have.

Suzanne: Right, an intermediate.

Ian: Right. And, you store some of the state in your clients and send it with every request. This means any of your services can handle any request at any time. There are no states that routes them to specific servers and makes your architecture much more difficult to scale, especially in the face of failures.

Suzanne: That actually has a lot of implications for programming practices…

Ian: It does.

Suzanne: ….in the database arena. So this is one that, especially for programmers who come out of, I will call it traditional data processing, they have to not just relook at the architecture but also relook at the implementation practices to achieve that.

Ian: Yes. And, look at how these frameworks promote statefulness and avoid it, probably.

Suzanne: One more. We have got four; so what is the last one?

Ian: The fourth one is kind of tied to some of the work that is going on in some of the big internet companies and the lessons that have been learned. As we build these very large systems, two things, especially, become really hard to implement. The first is that you will have failures in your system. The more nodes, the more hardware you have, the more software you have, just the law of averages is going to dictate that things will fail. You have to handle this. The bigger your system, the more things will fail. So, failures become common.

Suzanne: Right. We also call them normal failures.

Ian: Yes. That is right.

Suzanne: We have to account for normal failure.

Ian: Yes.

Suzanne: We do not like that term. And, we do not like to think about that, but that is really what you are saying is that failure is a normal part of dealing with volumes of data at this scale.

Ian: Yes. And, the second one is a really thorny one because you can build all these test cases for your code and generate really good synthetic test data and test really thoroughly your system, but, as soon as you deploy your system, the volume of data that it is working on will grow way beyond the envelope that you have tested. Hence, how do you know that your test cases and the actual code that you have built are going to work anymore? The answer is you do not, and you never will.

The only feasible solution to these things is to build in what has become called in the industry "observability," essentially what we used to call monitoring. But, observability is much more than just monitoring. It is the ability to pull data out about your system, both at the application layer and the system layer, and aggregate this data to so that operators and developers can gain insights into the behavior of the system at runtime. So, when failures occur, they may be warned about them in advance because you see decay in certain functions. It is very easy to look at your code, understand where inefficiencies are, and then work towards streamlining the solution. But, without this observability, you just cannot build these systems because they will fail in unexpected ways, and you will have no idea how to fix them.

Suzanne: That is a clear architectural implication.

Ian: Absolutely.

Suzanne: Because you cannot have observability if you do not plan for that in your architectural choices.

Ian: Yes.

Suzanne: So, that is another area where, if you are not accustomed to dealing with big data, you are likely to miss that as one of your architectural tradeoffs that you are making. How much observability do I need? [How much observability] can I afford in comparison to other attributes?

Ian: Observability is a really thorny one because it very quickly becomes a big data problem in itself.

Suzanne: I can see that.

Ian: We taught some internet companies, and they managed, literally, half a million data points a second.

Suzanne: Oh, my.

Ian: And, this is an amazing number. Some manage many more than that. This all has to be logged and aggregated. These low-level measures really do not mean very much by themselves. You have to actually process them and build up measures that mean something to a developer or an operator.

Suzanne: Right. So, that you can develop triggers, because that is what you are really looking for are triggers that highlight that you need to be looking at this part of the system…

Ian: Exactly.

Suzanne: …or that part of the system for decay and other issues.

Ian: Yes. So observability is a really tricky one. It becomes even trickier because there are really no out-of-the-box solutions that can help people right now.

Suzanne: Is that an area that you are researching in terms of how do we improve observability practices? Because I can tell you right now, I do not have a set of observability practices in my mental model of how you architect systems.

Ian: Yes. That an area that myself and my colleague, John Klein, intend to look at in next year's funding. The ability to create a customized observability framework that has got the ability to measure just the measures you want from your application—so that you can streamline the measurement, streamline the aggregation, and get effective visualizations of what is going on—is an area that we want to start to build tools to help organizations very quickly be able to put these observability solutions together.

Suzanne: There is sort of a layering that is implied there. I have that first layer of, *Here are the essential things*, if I get indicators out of that that there is a problem, I want to have that next layer of detail available that I can look at and say, *Well, if it's over here, now I want the data that relates to this in particular*. So this is not just a single framework problem.

Ian: No.

Suzanne: You are going to have some fun.

Ian: This is a framework. It is data storage engine. It is a set of visualization techniques. It is also the ability to customize because organization A is going to have very different observability

needs to organization B because their applications are different. It is very hard to pull something out of the box and make it work for both organizations.

Suzanne: Yes. You are also starting to see more COTS [commercial-off-the-shelf]-based big data kinds of things. Now you have got a different kind of observability issue because now I have, essentially, black box. So how do I deal with that? So that is another. I'm giving you more research topics aren't I?

Ian: All hard ones I'm afraid.

Suzanne: I'm really good for that. OK. Well, it sounds like you are going to be busy. And I'll get to talk to you in a few more months, I think, and hear about how some of these things are going. I think everything you've said makes great sense to me. That is wonderful for me. I don't have to work in these areas. I was thinking about my brother. I have a brother who worked in big data in the mortgage industry, at one point, and, remembering some of the complaints that he was talking about, they relate directly, especially, to that observability principle of trying to detect failure. So, I think you've done us all a great service by trying to think about these things.

Ian: Well, hopefully we'll have some solutions one day.

Suzanne: Well, this is a start. You are not the only one that has to think about how these principles are implemented. Getting people to think about how they do this stuff is one of the ways, start sharing that in the community and we all learn.

Ian: I would love to hear from other people who have other principles because I'm sure there's more.

Suzanne: There you go. There's an invitation. Maybe a little blog activity would be in order. Well, thank you so much for joining us today, Ian.

For more information on the approach that Ian and John have developed, please check out his recent blog post at blog.sei.cmu.edu click on Ian Gorton's name, that is G-O-R-T-O-N, in the author category.

This podcast is available on the SEI website at sei.cmu.edu/podcasts and on Carnegie Mellon University's iTunes U site. As always, if you have any questions, please don't hesitate to email us and info@sei.cmu.edu.

Thank you for listening.