Software Architecture cont'd

Table of Contents

Welcome

**083 Shane: Hi and welcome back to the SEI virtual event, Software Architecture Trends and New Directions. Again, we have Ian Gorton with us now, going to give our next talk on software architecture for big data systems.

Ian Gorton

Software Architecture for Big Data Systems

Ian Gorton
Senior Member of the Technical Staff - Architecture Practices

Ian Gorton is investigating issues related to software architecture at scale. This includes designing large scale data management and analytics systems, and understanding the inherent connections and tensions between software, data and deployment architectures in cloud-based systems.

I've written a book in 2006, Essential Software Architecture, published by Springer-Verlag. It sold well and has had several excellent reviews in Dr Dobbs and ACM's QUEUE Magazine. A 2nd Edition was published in 2011. I also co-edited 'Data Intensive Systems' which was published by Cambridge University Press in 2012. I've also published 34 refereed journal and 100 refereed international conference and workshop papers, with an h-index of 28.

**048 Ian Gorton is investigating issues related to software architecture at scale. This includes designing a large scale data management and analytic systems and understanding the inherent connections intentions between software, data, and deployment architectures in cloud based systems. Since obtaining his PhD in 1998, Ian has worked in academia, industry, and government. In 2006, he wrote Essential Software Architecture. A second edition was published in 2011. He also co-edited Data Intensive Systems, which was published by Cambridge University Press in 2012. Ian, all yours.

Ian Gorton: Thanks, Shane. And good morning to everybody Pittsburgh time. Thanks for hanging in on the webinar this far. As I'm sure many of you have been involved in, in the last decade we've seen an unprecedented growth in the scale of the systems that we've been building.

Scale changes everything

Scale changes everything

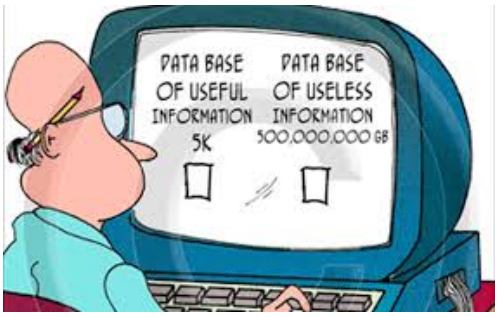
**050 While the most prominent examples of this have been the leading Internet organizations, the pressures of scale are being felt at systems of all levels within business, within governments, and within the Department of Defense.

When we build these scalable, just like urban planners who designed the massive urban connobations such as New York or Shanghai, we need to think differently about our solutions and to adopt software architectures, methods, and mechanisms that can ensure the scalability that our systems require. Of course, one of the driving forces behind the need for scalability is data. And I'm sure most of you have seen a chart such as this that describes the big data problem using the three Vs of volume, the size of data which is continually growing, the velocity of data, the speed at which it arrives and which it changes, and the variety of the data, the heterogeneity of the data types that many solutions are forced to store and fuse to come up with meaningful solutions. So, big data is really a driving force behind the scalability that many systems need today.

What is Big DatA? from a software Architecture Perspective...

WHAT IS BIG DATA?

FROM A SOFTWARE ARCHITECTURE PERSPECTIVE ...



**051 And in this talk, what we're going to do is initially just look at the big data problem from a software architecture perspective, look at some of the quirks that big data brings to the design of systems at this scale. And then I'll briefly describe some of the work that we're doing at the SEI to extend the knowledge base and the approaches that have been developed here over the last two decades into the realm of big data.

Some Big Data …

Google:

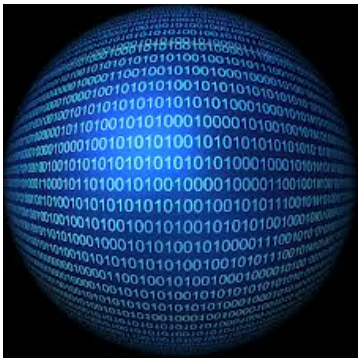
- Gmail alone is in the exabyte range

Salesforce.com

- Handles 1.3 billion transactions per day

Pinterest.com

- 0 to 10s of billions of page views a month in two years,
- from 2 founders and one engineer to over 40 engineers,
- from one MySQL server to 180 Web Engines, 240 API Engines, 88 MySQL DBs + 1 slave each, 110 Redis Instances, and 200 Memcache Instances.



http://highscalability.com/blog/2014/2/3/how-google-backs-up-the-internet-along-with-exabytes-of-othe.html
http://highscalability.com/blog/2013/9/23/salesforce-architecture-how-they-handle-13-billion-transacti.html
http://highscalability.com/blog/2013/4/15/scaling-pinterest-from-0-to-10s-of-billions-of-page-views-a.html

**052 So, just to calibrate what we mean by big data, it's very hard to get these numbers. A lot of them are commercially in confidence. But if you dig around a website such as highscalability.com, which is a valuable source of information in this area, you'll find information such as this. So, Google's Gmail alone now extends into the Exabyte range for data storage. That's just Gmail, no search information, none of the other services that Google offer. That's a lot of data.

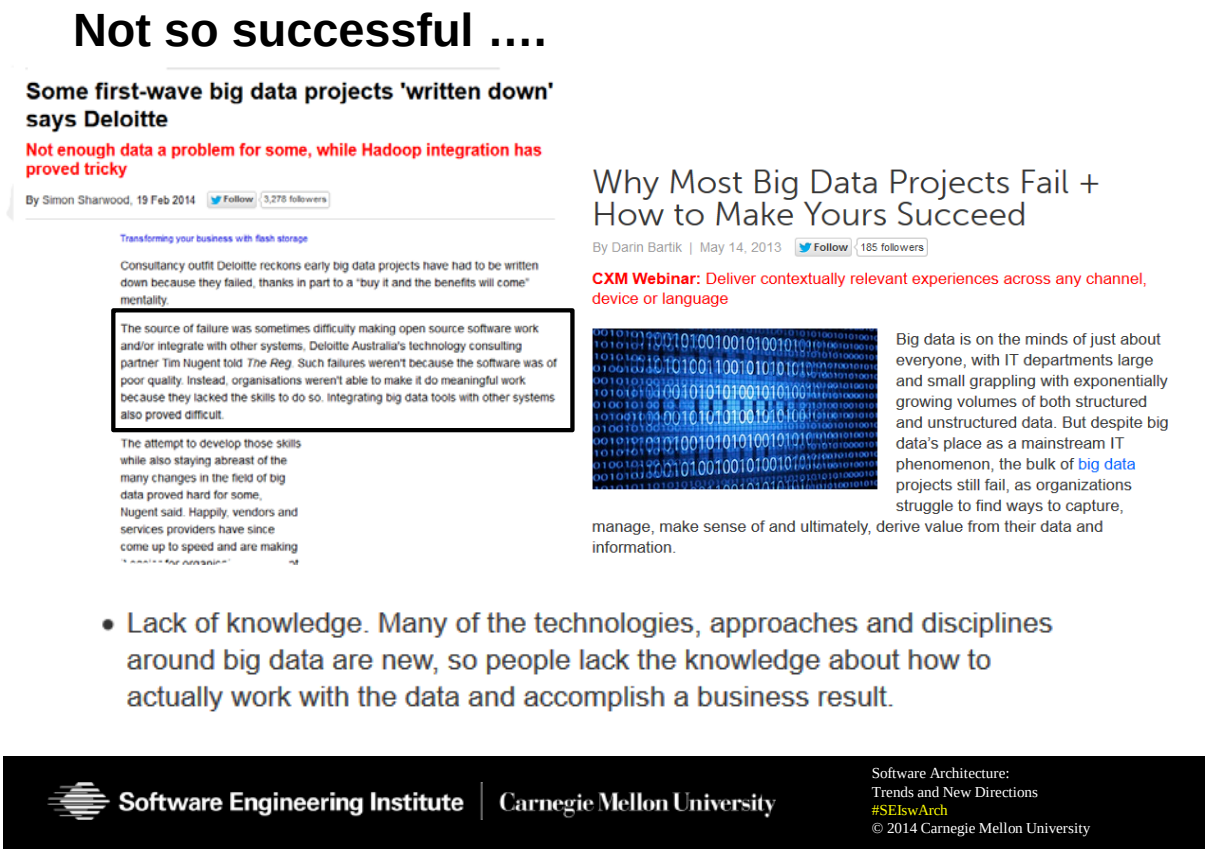
Salesforce.com, one of the leading businesses on the Internet, handles one point three billion transactions a

day. And they have a lot of data. We don't really know how much.

Pinterest, a social media site, which, if you're not familiar with, I guarantee your teenage children will be, is an interesting case study. It grew in two years from a couple of engineers and one MySQL server to now, or when this report was written anyway, a hundred and eight webservers, two hundred and forty API engines serving business logic, and eighty-eight MySQL databases instances all with slaves, plus in memory databases, Redis, a hundred and ten instances of that, and two hundred Memcached instances. This is just to serve location data, not just pins on Facebook. So, it's a lot of infrastructure there, a lot of software.

But like any good story, there's two sides to it. This is high scalability expresses many of the success stories.

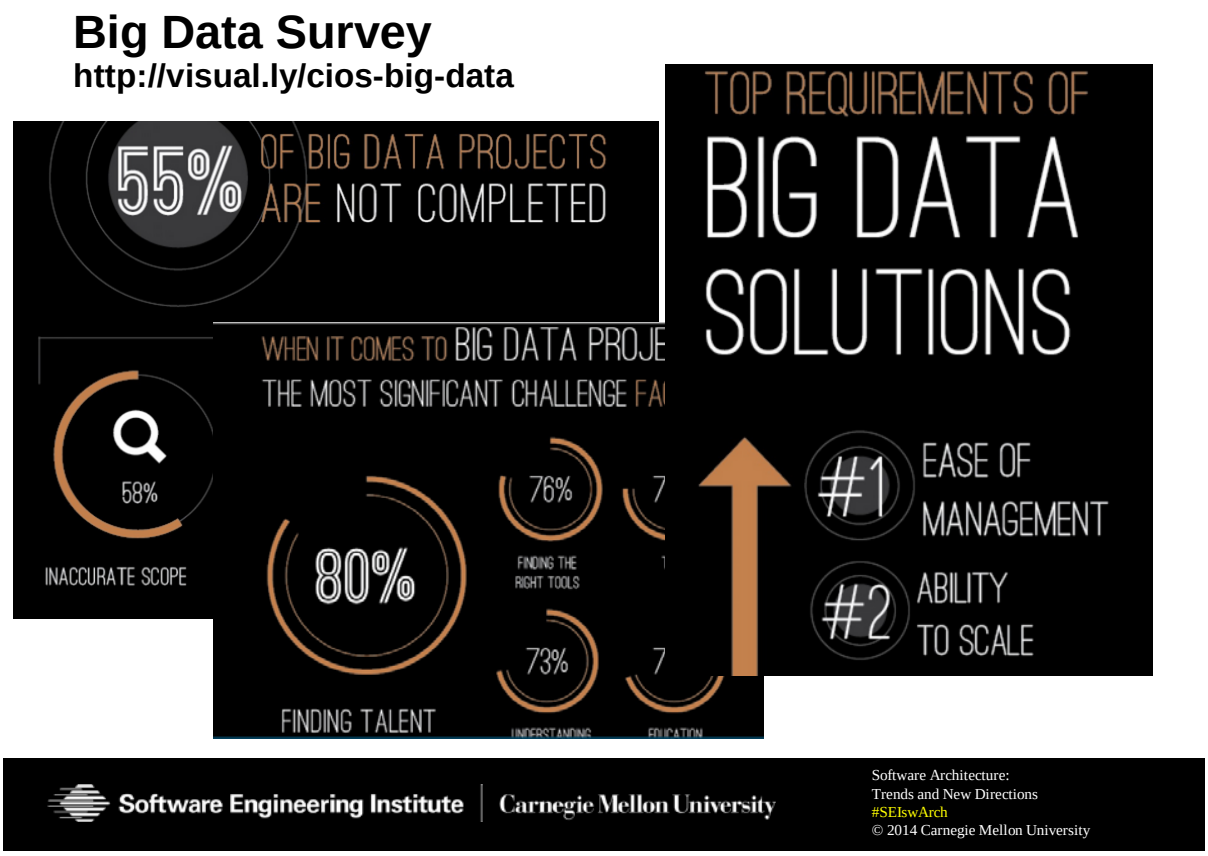
Not so successful…



**053 But the world is not full of success stories. And if you dig around a little bit more, you'll find lots of information like this. On the left hand side, for example, a report from Deloitte which emphasizes the complexity of adopting the new open source technologies that are available for building big data systems. As the quote says, it's not that the open source technologies themselves are particularly frail or imperfect. They're actually very good, solid technologies. It's just that they're complex to use. And the community of software engineers are just not used to this level of complexity with their technologies.

This is emphasized on the right hand side where it mentions exponential growth of data as being a major problem, and in the bottom where the lack of knowledge of software engineers in deal with the problems that scalability brings are at the core of the problems that we have in building these software systems and the failures that people have.

Big Data Survey



**054 This is emphasized again in a survey from the end of last year, if I remember correctly. The URL's on the slide there, that emphasizes that fifty-five percent of big data projects are not completed. The scope of the systems is obviously difficult to

derive. Requirements gathering becomes difficult at scale. But also the technological road blocks and access to data become key issues when building big data systems.

The most significant challenges, eighty percent of organizations reported that finding talent was the most significant challenge that they had. And also the use of technologies is a major one. And what are the major drivers? For solutions, ease of management because when you're building a very large distributed data system, you need to be able to manage it effectively and evolve it and grow it and handle failures, but also the ability to scale. In this realm, if your system can't scale to handle the influx of data and workload that's put on it as time evolves, then it's just not going to succeed. And the underlying fabric to your system has to be scalable.

Big Data – State of the practice
"The problem is not solved"

Building scalable, assured big data systems is hard

- Healthcare.gov
- Netflix – Christmas Eve 2012 outage
- Amazon – 19 Aug 2013 – 45 minutes of downtime = $5M lost revenue
- Google – 16 Aug 2013 - homepage offline for 5 minutes
- NASDAQ – June 2012 – Facebook IPO

Building scalable, assured big data systems is expensive

- Google, Amazon, Facebook, *et al.*
  - More than a decade of investment
  - Billions of $$$
- Many application-specific solutions that exploit problem-specific properties
  - No such thing as a general-purpose scalable system
- Cloud computing lowers cost barrier to entry ***– now possible to fail cheaper and faster***

**055 So, despite organizations such as Google, and Amazon, and Facebook investing we don't know how much, but billions of dollars is likely in building their solutions that serve as information on the Internet, the problem of building big data systems is far from solved. A few instances mentioned on this slide here, even Netflix, and Amazon, and Google have outages. It happens reasonably regularly and is quite widely reported.

We're probably all familiar with the problems of healthcare.gov at the end of last year. Some of these

stemmed from integration into backend data sources, from changes in workload from read heavy to more write workload late in the project. And also the use of technologies, a technology that was offering certain services that just didn't scale as well as expected when the system went live.

We also are familiar with things like the NASDAQ failures during the Facebook IPO, again caused by the scale of the load exerted on the systems. So, one of the bonuses of the Internet organizations investing so much time and effort and money in these big data solutions is that there's now a bunch of technologies available for us to exploit in our systems as we build our solutions.

Polling Question #1

Are you using NoSQL technologies in your current systems?

**056 And here you'll see first polling questions. So, if you can answer these, that'd be great. And are you using NoSQL technologies in your current solutions?

NoSQL – Horizontally-scalable database technology

Designed to scale horizontally and provide high performance for a particular type of problem

- Most originated to solve a particular system problem/use case
- Later were generalized (somewhat) and many are available as open-source packages

Large variety of:

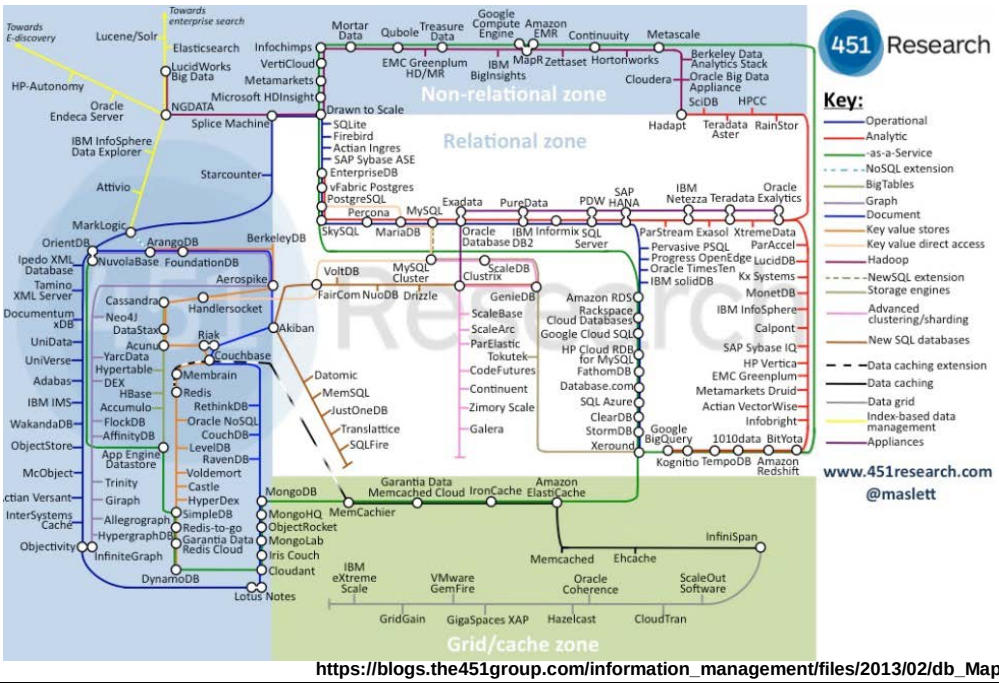
- Data models
- Query languages
- Scalability mechanisms
- Consistency models, e.g.
  - Strong
  - Eventual

**057 Because no SQL technologies have rippled down from many of the innovations from the Internet organizations. Amazon with their Dynamo DB work, Cassandra has come out of Facebook. And there are now open source technologies that we can use to build very highly scalable data systems.
They don't make the whole solution simpler, though, because there's first of all a very large variety of data models. No longer do we just have relational models to consider and SQL. We have key value stores and graph stores and column stores and document stores to consider. And all of these different underlying data

models. They have different underlying query languages. The technologies are designed to scale in different ways.

And they offer different levels of consistency. We're very used to, in the relational world, having strong consistency. When we make an update to data item, that update is visible subsequently to all other queries. In scalable systems, sometimes this just isn't possible to achieve. And we have to settle for weaker eventual consistency whereby we make an update to an instance of a data item, and other instances that replicated for availability purposes are not instantly updated.

You'll see this if you go to Facebook all the time. I see eventual consistency when I log on to Facebook all the time because I'll have emails telling me that someone's commented on a post. But when I look on my homepage, there's no instances of that comment for few minutes until Facebook updates.



https://blogs.the451group.com/information_management/files/2013/02/db_Map_2_13.jpg

**058 This evolution of technologies in the database world has created this complex landscape for organizations to choose the technological fabric from for the data layer. We jokingly call this the Tokyo subway map. And it's probably far too small for most of you to read, but it really does express the complexity of the database world now and how it's evolving in many different directions with different models, different query languages, and a whole bunch of different technologies to work with.

Horizontal Scaling Distributes Data
(and adds complexity)

Distributed systems theory is hard but well-established

- Lamport's "Time, clocks and ordering of events" (1978), "Byzantine generals" (1982), and "Part-time parliament" (1990)
- Gray's "Notes on database operating systems" (1978)
- Lynch's "Distributed algorithms" (1996, 906 pages)

Implementing the theory is hard, but possible

- Google's "Paxos made live" (2007)

Introduces fundamental tradeoff among "CAP" qualities

- Consistency, Availability, Partition tolerance (see Brewer)
- "When Partition occurs, tradeoff Availability against Consistenc Else tradeoff Latency against Consistency" (PACELC, see Abadi)

"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable"

**059 But all of these technologies introduce horizontal distribution to achieve scalability. So, essentially, we're taking our database which we're familiar with being on one machine, and we're horizontally distributing it, basically breaking it up into chunks and running it on clusters of low cost machines. And by doing this, we create a distributed system. And for many of you, this is a bunch of theory listed on this slide which you'll be familiar with. Or for some people quite often, like myself, it's theory that you once knew and then forgot.

It's complex stuff. Building distributed systems has never been easy

because there's difficult fundamental problems associated with distribution. The CAP theory from Brewer at Berkeley kind of nicely encapsulates some of the tradeoffs that we have to make when designing a big data system. Whereas if we have our data replicated, and there's a break in the network connectivity, we have to make a tradeoff between having the data still available and having it consistent if it's replicated across the partition.

And there's a very nice example of the complexity of translating theory into practice in the paper from some folks at Google, "Paxos made live" in 2007. Paxos is a distributed consensus algorithm for achieving consistency. And it's expressed in the original paper in about a page or so of pseudo-code, looks pretty straightforward to understand once you get your head around it.

The Google guys tried to implement this algorithm in their infrastructure. And their paper beautifully describes the complexities of translating a very clear expression of an algorithm into a system of the scale that Google were deploying and how, two years later, they were still seeing failures in their implementation because of the complexity, and who it was designed to mask failures.

So, building distributed systems is just not easy. And it's something that many of us in the software world are just not intimately familiar with because it's just not been something we've had to work on so far.

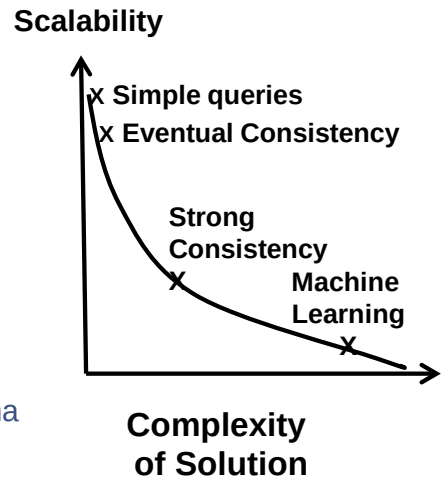
Rule of Thumb: Scalability reduces as implementation complexity grows

Workload

- # of concurrent sessions and operations
- Operation mix (create, read, update, delete)
- Generally, each system use case represents a distinct and varying workload

Data Sets

- Number of records
- Record size
- Record structure (e.g., sparse records)
- Homogeneity/heterogeneity of structure/schema
- Consistency



**060 There's one rule of thumb which is definitely worth considering when you're building these systems. Scalability has many dimensions, workload, increasing the workload, read and write workload, increasing the size of the data, the size of the files, the handling of consistency across all of the instances of the data. But if you think of scalability in general, you can essentially say that the more complex your solution to a problem, the less it's going to scale.

So, let's take a couple of examples here. So, if I have a system that has eventual consistency, I may have three replicas of my data, when I

make an update, I just update one of them instantly and wait for the updates to the other instances to ripple through in the background controlled asynchronously by my database. That's going to be fast because I'm just doing one update.

If I require strong consistency such as in acid transactions that we're familiar with in relational technologies, and also in some NoSQL technologies, this is going to be slower. It may be implemented beautifully and reliably and as efficient as possible. But the underlying mechanism to replicate and ripple the updates through to the replicas in an assured way so that strong consistency is achieved requires more communications, more logic, more ability to handle failures. It's going to be slower. And there's very little you can do about that. It's a fundamental tenet of software.

Another example would be answering simple queries. Again, think when you go to Facebook to your homepage, essentially what's happening is a lot of very simple queries are being fired off to Facebook's infrastructure. Results are coming back. And they're being aggregated to form your webpage. All this happens very quickly.

But if you're then going to do some complex analysis of machine learning such as Netflix's recommendation engine for finding the movies that you're most likely to want to watch, then this isn't going to be as quick

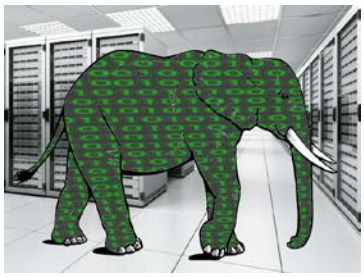
because it has to do queries and smart statistical algorithms across very large collections of data. And so, that sort of query is not going to respond quickly. Hence, the solution is going to be less scalable. And you think about ways you can maybe cache the results so that when you want to do a recommendation, you don't actually use the machine learning algorithm, you use the results of it that were generated a few minutes or hours ago.

So, there's this complexity here to the solutions. But the simpler your solution, the more your system is going to scale.

Big Data –
A complex software engineering problem

Big data technologies implement data models and mechanisms that:

- Can deliver high performance, availability and scalability
- Don't deliver a free lunch
  - Consistency
  - Distribution
  - Performance
  - Scalability
  - Availability
  - System management
- Major differences between big data models/technologies introduce complexity



**061 So, the big data problem is a difficult software engineering problem. Many of the building blocks are starting to become available in the open source world and available commercially. And we can use these to compose our systems, pull different pieces of software together to build solutions. But don't expect to have a free lunch. You've still got to think very carefully about consistency, distribution, how fast your system's going to perform and scale, what happens when failures occur so that you can ensure availability of your data. And then how you manage this complex collection of nodes that form your

distributed system, how you manage, evolve, move data around, back data up, etc. So, this is not an easy world to build solutions for.

Software Engineering at Scale

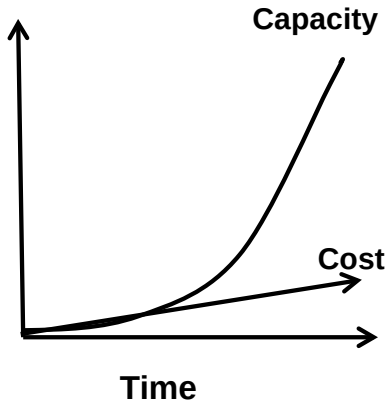
Software Engineering at Scale

Key Concept:

- system capacity must scale faster than cost/effort
  - *Adopt approaches so that capacity scales faster than the effort needed to support that capacity.*
  - *Scalable systems at predictable costs*

Approaches:

- Scalable software architectures
- Scalable software technologies
- Scalable execution platforms



**062 Let me just finish this section with one more insight that's worthy of consideration. If you look on the graph on the right hand side there, you see that capacity of the system is going to grow over time. And in many systems, this grows with some exponential function. So, as time continues, then the rate of growth accelerates. So, this is what's to be expected of a big data system that can scale.

However, it's really important that as you build these systems and their capacity grows exponentially over time, your costs don't. If your costs grow exponentially over time, then you're in trouble because it's going to cost you an awful lot of money to deliver these systems. So, we want a solution that can grow the capacity exponentially, but our costs grow linearly and hopefully at a nice low angled slope such as on this slide here.

And to do this means that you have to adopt approaches in your solutions that can scale your system's capacity without a great deal of effort being exerted on those scalability mechanisms. And the approaches to do this are all related to building scalable software architectures that can be easily extended without massive code changes, adopting the right technologies that enable you to scale things easily without doing massive reconfigurations of your system. And also, acquiring and deploying your system on scalable execution platforms such as cloud technologies whereby you can elastically grow your demand instead of having to have engineers come in and install systems for you. So, this is a real key factor with any massively scalable system is that they engineering approach that you follow must ensure that your costs are controlled so that you can build a scalable system with predictable costs.

Polling Question #2

Are you planning to evaluate or adopt No SQL databases in the next 12 months?

**063

SO WHAT ARE WE DOING AT THE SEI?

**064 So, let me tell you a little bit about what we're doing at the SEI in the world of big data and scalable systems.

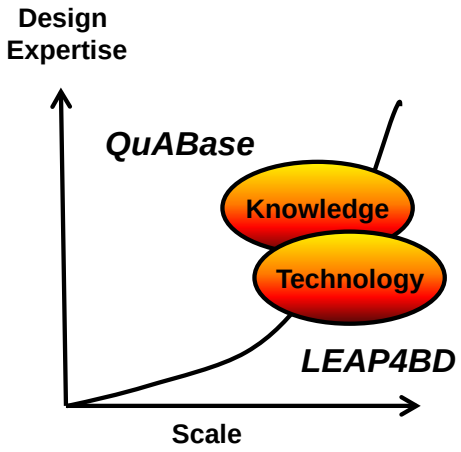
Enhancing Design Knowledge for Big Data Systems

Design knowledge repository for big data systems

- Navigate
- Search
- Extend
- Capture Trade-offs

Technology selection method for big data systems

- Comparison
- Evaluation Criteria
- Benchmarking



**065 As I've already kind of eluded to on the earlier slides, two of the major issues that are facing organizations when building big data systems revolve around the design expertise that's required to build systems that scale, and also the technologies that you need to deeply to build big data systems because many of these technologies are new and have mechanisms in them that people aren't familiar with.

So, as the scale of your solution grows, then this design knowledge become more and more critical. Very small areas of your architecture may introduce bottlenecks which are

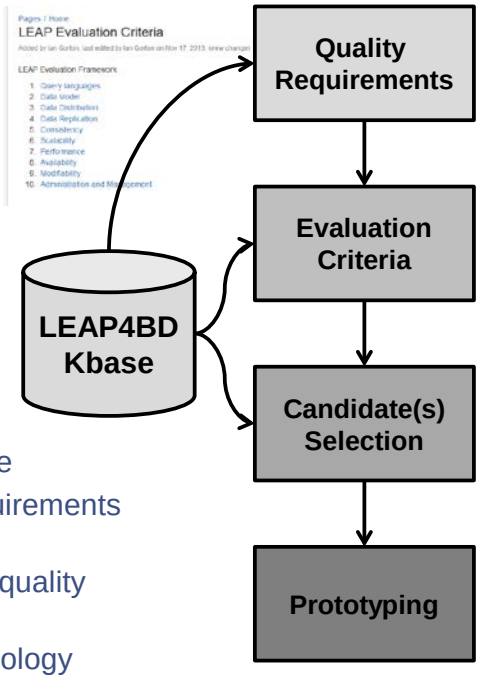
exposed as the scale of your system grows. Or you may find that small failures can ripple through and cause very large failures or backlogs unexpectedly as your workload or your data size grows. So, the knowledge becomes critical as your systems grow in scale.

But you can have the best theoretical knowledge of building distributed systems and big data systems in the world, but if you then choose inappropriate technology to build your system on, technology that's not designed to scale to support the scalability that you need, then your system's not going to succeed. And it's this confluence of knowledge, design knowledge and technology knowledge, where we're starting to do some work in this space.

And so, in the rest of this talk, I'll briefly design our work in the QuABase which is a knowledge repository of design expertise for big data systems that we're developing this year, and also our approach called Leap4BD, which is a technology selection method that can be used for big data systems. And I'll start with Leap4BD first.

LEAP4BD

Lightweight Evaluation and Architecture Prototyping for Big Data (LEAP4BD)



Aims

- Risk reduction
- Rapid, streamlined selection/acquisition

Steps

1. Assess the system context and landscape
2. Identify the architecturally-significant requirements and decision criteria
3. Evaluate candidate technologies against quality attribute decision criteria
4. Validate architecture decisions and technology selections through focused prototyping

Software Engineering Institute | Carnegie Mellon University

Software Architecture:
Trends and New Directions
#SEIswArch

**066 So, the easiest way to think about Leap4BD if you're familiar with some of the SEI's methods is as an evolution of the ATAM method for big data systems, but with two real key differences. The first is that it's targeted only at the data layer. So, ATAM is a method that will enable you to assess the whole architecture of your system. Leap4BD is much more lightweight in that it focuses very much on the data layer of your system and enabling you to choose solutions for that layer that are appropriate to achieve your scalability aims.

So, any system is going to need a set of quality requirements in terms of scalability, consistency, availability, system management, etc. that are going to drive the selection of a database technology. And in Leap4BD, what we've done is we've built a canned set of evaluation criteria for big data systems. And there's just the major headings are on the slide there next to the quality requirements box.

So, we have I think it's ten major criteria. And within each of those areas, there's very extensive lists of detailed evaluation features that you can look at when you're trying to choose a database system and map these back to the quality requirements for performance and scalability in consistency that your solution needs.

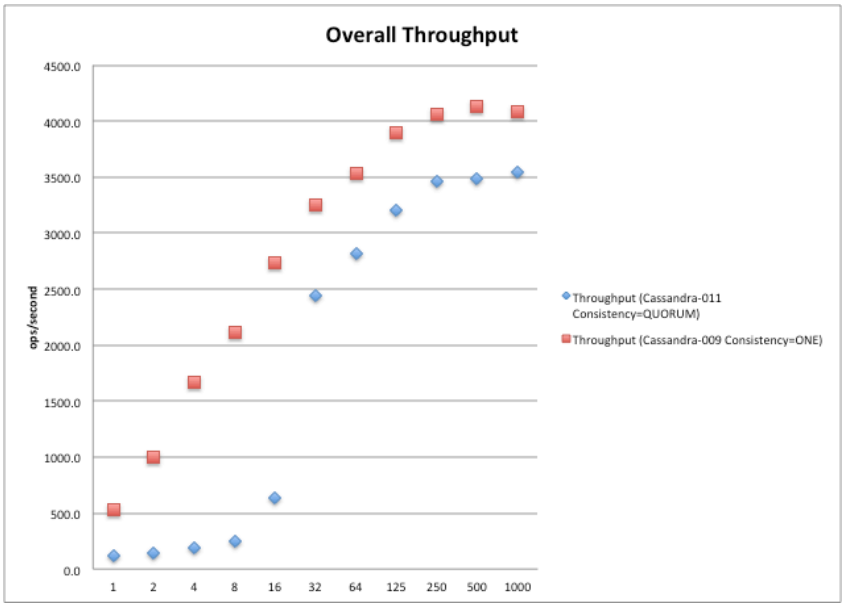
In Leap4BD, what we're doing is we're building a knowledge base so that, as we work with big data technologies, we assess each of these technologies against the criteria in the Leap4BD framework. And so, right now the evolving knowledge base has evaluations of technologies such as Cassandra, MongoDB, Riak, Neo for J. And other technologies we'll be working with soon will be incorporated into this reusable knowledge base that an organization can just pick up, relate their quality requirements to the detailed criteria, and very quickly choose some candidate technologies that might be suitable for their solution.

What's very important at that stage is that because of the complexities of scale is that you don't just work it on paper and choose your solution. It's that you do some prototyping. And in Leap4BD, we have a prototyping phase where we're working on developing some technologies that make it easy to prototype, build very quick performance and scalability prototypes that can be deployed on a cloud such as Amazon's EC2 or Open Stack, and run real benchmarks against the technologies that you've deemed as appropriate candidates for your solution.

So, to do this, we're actually right now leveraging a technology called YCSB, the Yahoo Cloud Server Benchmark. And we're extending that to provide much more custom features for big data systems.

Let me just show you some of the importance of prototyping in this space.

Some Example Scalability Prototypes - Cassandra



**067 This is just some work we've done with Cassandra, a column oriented database from Facebook which is widely used in a lot of systems. This is a plot of performance in terms of throughput and transactions per second, requests per second. As the load on the x-axis grows from one client to a thousand simultaneous clients all firing off requests continuously, as soon as one set of results is received, a client sends that next request. So, a thousand clients are actually generating a lot of load. The Cassandra instance in this case is a single instance serving just ready only loads. So, this is a read only

test. And it's comparing a single instance against an instance of Cassandra which is using three nodes. So, we're replicating the data across three different nodes in an Amazon EC2 cloud.

And here you can see, this is a fairly straightforward plot. It's exactly what you'd expect to see with a good technology. As the load increases on the x-axis, the throughput increases until at around, I think it's hundred and twenty-eight, you're starting to see saturation of the server or network capacity, of some elements of the system. This is why you're seeing a straight line.

At lower loads, one server is actually quicker than three, which you'd expect because there's complexity in managing replicas in Cassandra. Your request may be sent to a replica that doesn't own the data, for example. It has to be redirected. But as the load increases past like two fifty-six or five twelve, we see the benefits of having three servers rather than one. The performance of our single server is starting to decrease, whereas the performance of our three servers is pretty much leveling out.

What happens past a thousand clients, we don't know. The only way to know that is to test. But you can pretty much guarantee that the single server is going to start to decrease in performance. And hopefully, the three servers will just level out for a long time until they are eventually saturated in capacity and

start to fall off. So, that's a really plot showing what you would expect from a good technology.

This is showing a similar load. I think this is a write-- a read/write test, or a write test. I forget which. Again, comparing three servers against one server. And here you can see a very different performance characteristic profile. The three servers again is a little bit slower at low loads because of the overhead of managing three. But very quickly, around thirty-two clients requests, simultaneous client requests, we see the capacity of a single server is overloaded.

It levels out here quite nicely. So, it's still serving a reasonable amount of throughput as load increases. But it can't go any further. It's got no more power left. Whereas, the three servers, again, scale quite nicely. And again that line would hopefully extend far to the right in a flat profile until again it's capacity is saturated.

The third one shows another Cassandra test, but in this one we've configured it to be essentially three data centers. So, even though the three data centers are living on the same EC2 instance, deployment, we've configured Cassandra so that it will actually think each is geographically replicated. So, that when we do a write in one of the lines where we just have a consistency level of one, if we make an update to one of the instances, then the update is recognized at one data center and then eventually

replicated to the others using an asynchronous eventual consistency mechanism. Whereas in the other plot, we actually require all of the data centers to be updated when we make an update to one.

And here again you see the benefits of having a simpler solution. You get much higher performance. Both seem to scale reasonably well. But the simpler solution, again, with lower consistency levels gives you much higher performance.

So, this is the sort of insights we can derive through fairly simple prototyping. Doing this kind of work with the automated tools that we're starting to develop, it is not that complex. And it's essential for organizations to do this to gain deep insights into the technologies they're considering and confidence that they're choosing technologies that can actually be assured to provide the performance and scalability they require.

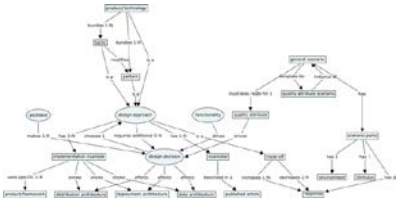
QuABase –
A Knowledge Base for Big Data System Design

Semantics-based Knowledge Model

- General model of software architecture knowledge
- Populated with specific big data architecture knowledge

Dynamic, generated, and queryable content

Knowledge Visualization



**069 Our other area of work is what we've called QuABase or the Quality Attribute Knowledgebase for big data systems. So, one of the things that's been reasonably revolutionary in the world in the last fifteen years or so is Wikipedia. Many years, old people like me are used to having encyclopedias that we looked at. But now, we just go to Wikipedia twenty times a day and refuse to pay for our access when they ask us for it. But it's been an absolute revolution to the world in the fact that it's built up this knowledgebase of information.

But Wikipedia is still very unstructured. And what we're trying to do with QuABase is extend the Wikipedia approach into the realm of big data architecture design. And to do this, we're using a much more rigorous approach. We're extending the core Wiki technologies through extensions to provide an underlying semantic data model, knowledge model, which captures the relationships between design concepts for big data systems. And we're then populating this model with knowledge about design principles and technologies that support those design principles. And the whole system is built so it can be dynamically queried and much of the content is generated dynamically.

So, in Wikipedia, people go in and they type all of the pages pretty much. In our QuABase, you put in some core information. And then much of the information required is generated dynamically in response to queries. And you can see a very small depiction of our idealized knowledge model in the bottom right hand corner there. And this underpins the knowledge that's in the wiki.

Polling Question #3

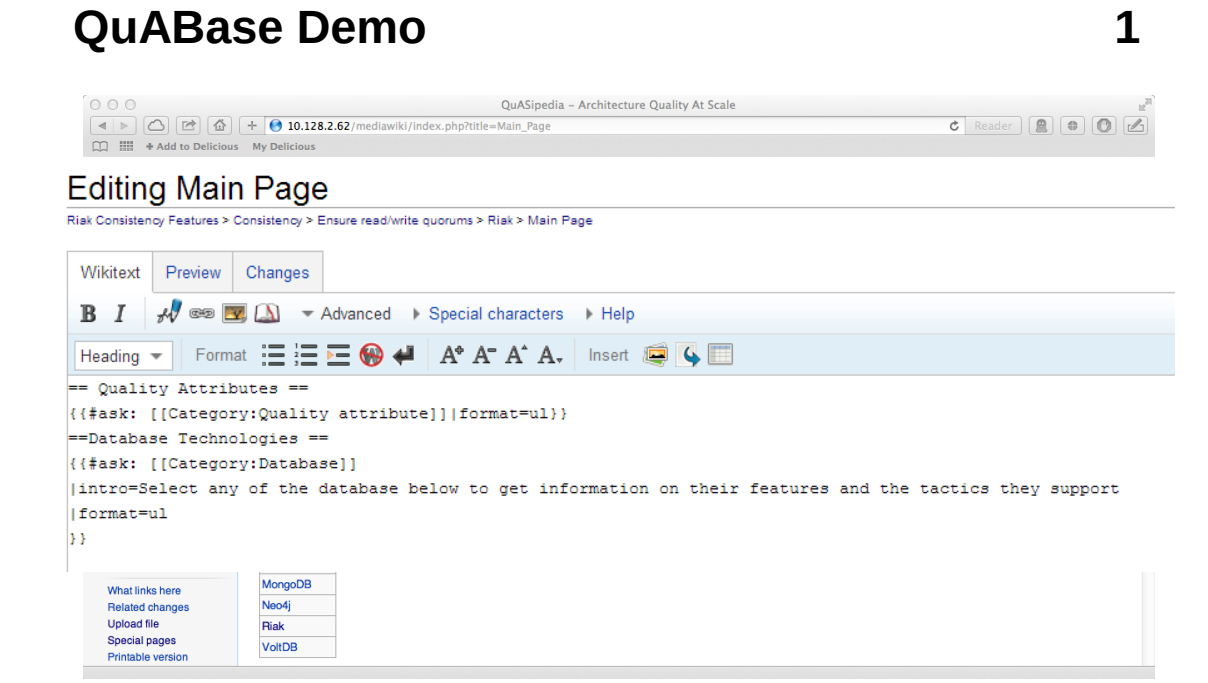
Would you be interested in a training course on software architectures for big data systems?

**070 So, let me give you some examples of what this system looks like.

QuABase Demo

**071 And as the slide says, "Warning, this is very much under development." So, we hope to have this finished by the end of summer, at least the first version. So, you'll see little bits of this are obviously unfinished.

QuABase Demo 1



**072 So, this is what the main page of QuABase looks like. And you can see it just looks like a wiki. And there's two areas which correspond to the chart I had earlier about design knowledge in terms of satisfying quality attributes for scalable architectures, and also database technologies. So, we have a list there of the six databases that we're initially using to populate the knowledgebase with evaluations of their features.

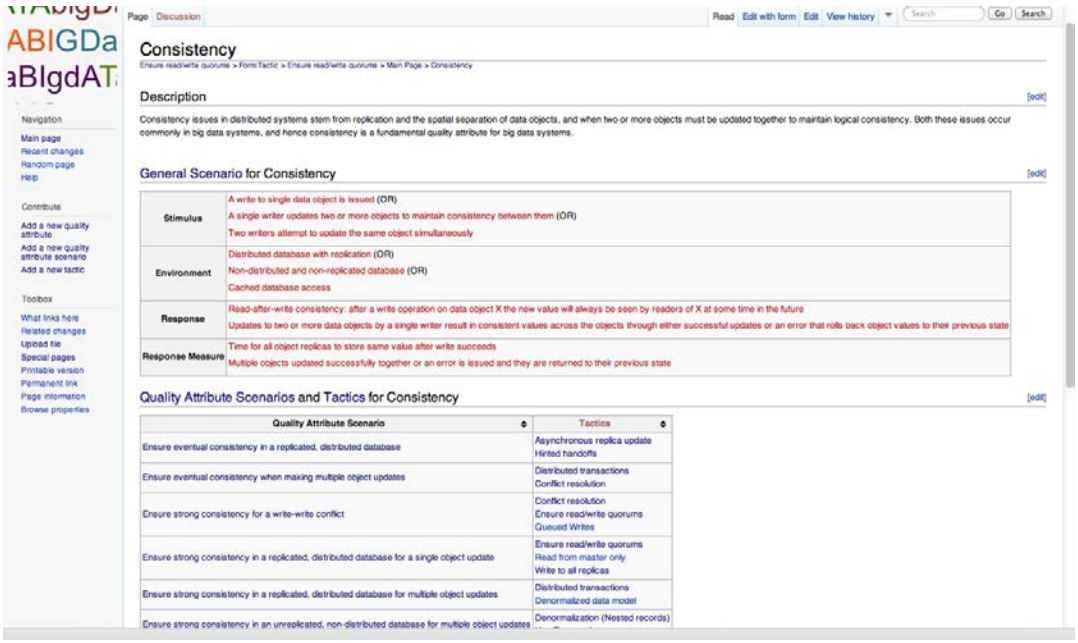
But if you look at what's underneath this page, you don't see text. You see a couple of queries. And these are known as semantic queries. We're

using an extension to Wikipedia, media wiki, I should say, called Semantic Media Wiki, which enables us to implement our knowledge model in the wiki pages. And then using the query language that it supplies, the ASK query language, we can dynamically build pages based on the underlying content.

So, on this page for example, if we were to add information about another database such as, I don't know, HBase, when a user goes to this new page, we don't have to change this query at all. The new data about HBase would automatically be retrieved and displayed for people to navigate.

So, this is an underlying principle of the design of the QuABase. We're using dynamically generated technology where-- content wherever possible.

QuABase Demo 2



**073 So, let me just delve a little bit more deeply into the pages related to consistency. So, consistency is a big issue in distributed big data systems because we have multiple replicas of data objects so that we can have availability and high performance. But keeping these replicas consistent is a problem. And in some systems, it's a much bigger problem than other systems, obviously. In banking, it's a bigger problem than in social media, for example.

So, here again we're leveraging some of the SEI's innovations in software architecture over the years in terms

of describing architectural qualities using scenarios. We have a general scenario for consistency that enumerates the elements of consistency. And from that, you can compose more specific quality attribute scenarios that are more relevant to a particular system.

And here we're doing the composition of the quality attribute scenario. It's in a more abstract way than you would typically would during an ATAM. So, you can see on the slide there, the table at the bottom is generated dynamically. That's based on a query. And we've basically enumerated a number of quality attribute scenarios that are relevant to consistency. And each of these has a set of tactics, again, built dynamically that are associated with supplying that quality attribute scenario in a system.

So, one of them, it talks about updating single objects and providing this with strong consistency. And one of the tactics is ensure read/write quorums. So, let's dig into that page.

QuABase Demo 3

QuABase Demo 3

Ensure read/write quorums

Riak Consistency Features > Riak > Riak Consistency Features > Consistency > Ensure read/write quorums

Description [edit]

Assuming there are N replicas of any object, a writer may specify that a quorum of the replicas must be updated before the write succeeds. This ensures that a majority of the replicas are updated before the write completes. If all writers perform quorum writes, this also prevents write-write conflicts as only one writer can ever achieve quorum at any instant.

To ensure all readers see the updated value after any write completes, readers must also specify that a quorum of object values must be the same before the read succeeds. This ensures that a reader cannot see a value at a replica that has not yet been updated with the new value.

In either case, if a quorum of replica objects cannot be written to or read from, the operation fails.

The general form or the requests to achieve strong consistency are: Qr + Qw > N Qw > N/2

A number of NoSQL databases provide quorum mechanisms for readers and writers to be able to tune consistency. This is typically specified on a per-write call to enable each write to be tuned accordingly.

Improves Quality	Consistency
Reduces Quality	Performance, Availability
Related Tactics	Hinted handoffs

Implementations [edit]

This tactic is supported by the feature Tunable consistency of the product Cassandra.

This tactic is supported by the feature Tunable consistency of the product MongoDB.

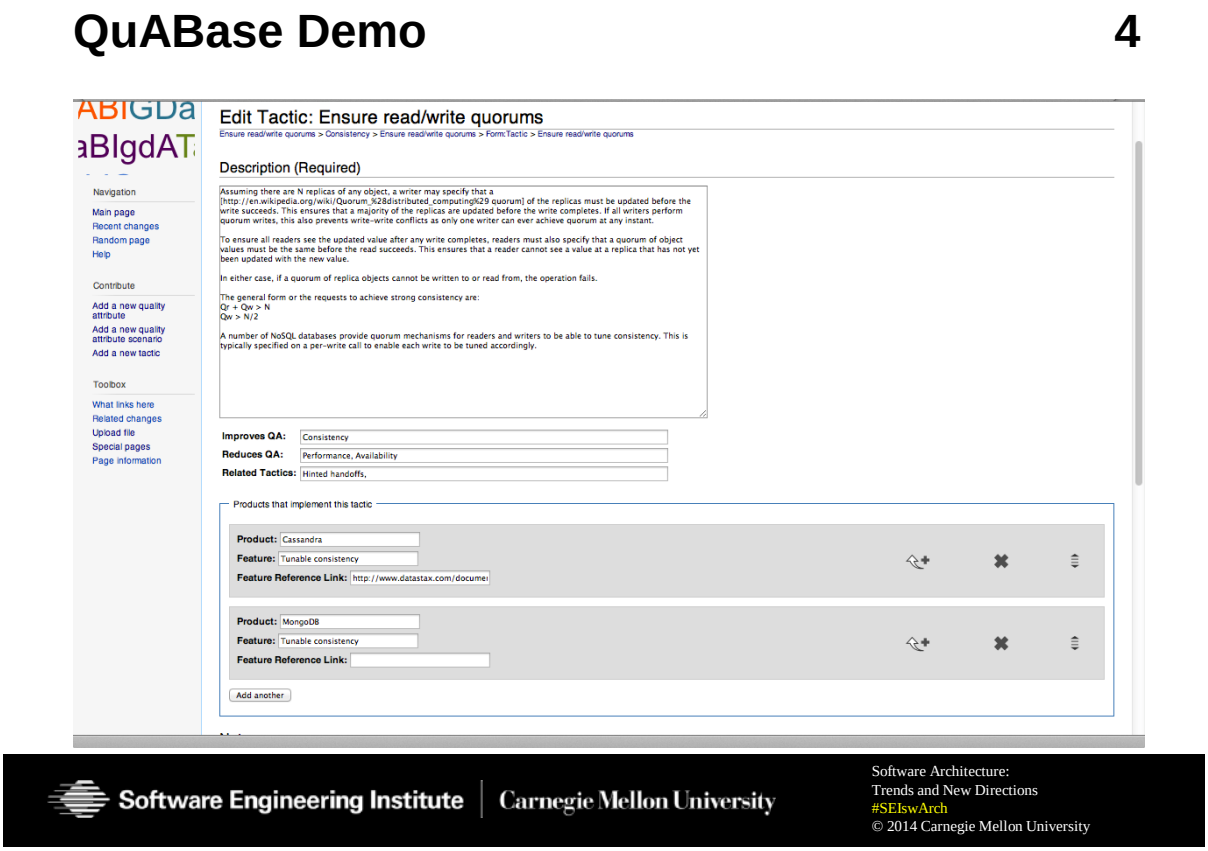
This tactic is supported by the feature Tunable consistency of the product Riak.

**074 Ensuring read/write quorums is essentially an approach for providing strong consistency in distributed data systems. If I have three replicas of an object, a data object, and I want to make an update to it logically, if I make an update to two of those replicas, then I'm guaranteed to get consistency because if at exactly the same time another client was trying to make a request to update the same object, ti would fail because it wouldn't be able to update two of them because I'm updating two of them. So, this guarantees that you get strong consistency. But you have to set up the configuration of your system correctly.

So, the page here just quickly defines the theory behind quorums and providing strong consistency. And then again, it shows you the qualities that this supports, i.e. strong consistency, but the things that it also trades off against. Quorums mean that your system is slower because you have to update more replicas to ensure consistency.

It can also affect availability because if you get a partition, and you don't have enough replicas available to satisfy your quorum, your query will fail. So, there's a tradeoff here. And the wiki captures these tradeoffs dynamically. Again, this is all built through queries.

At the bottom there is where you'll see that we also mock up the pages with the technologies that support quorum type consistency. So, you can see there we've got Cassandra and Riak and Mongo. So, again, all of this is built dynamically.

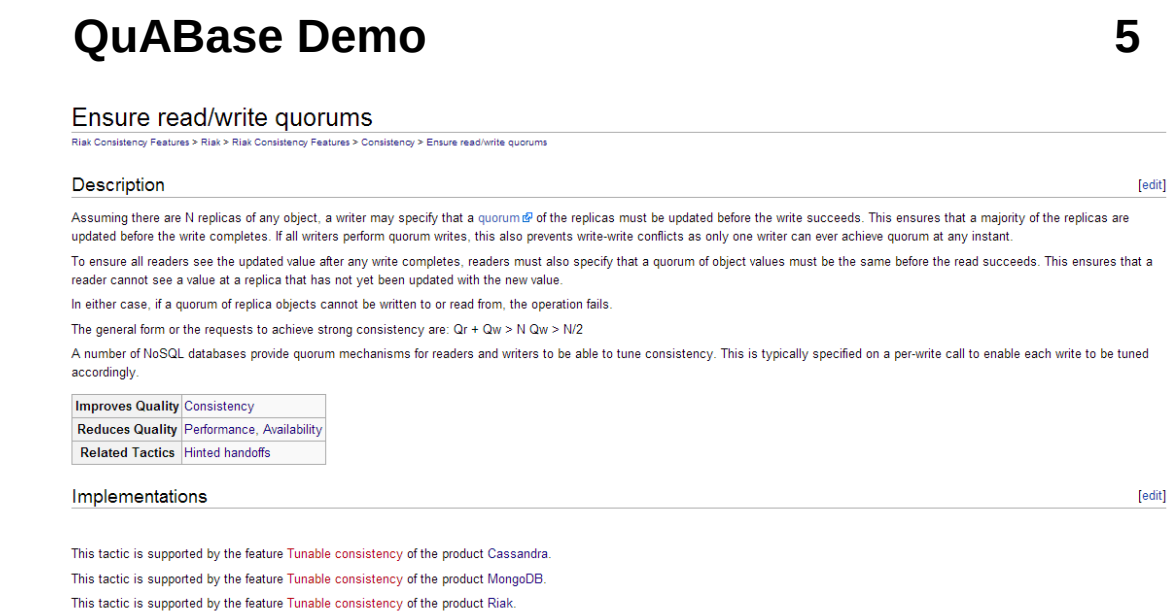


**075 And this is what the page underlying that looks like when we build it. There's essentially a set of forms for every single page. And so, to construct a page, there's essentially a format that you must follow. This cannot-- these pages can't be constructed using random formats such is normal within normal wikis.

So, we fill in the forms. Some of it's just textual. Other elements of the form have selections from dropdown boxes that force you to provide information in the correct format. And so, underlying this again is the semantic model that we're

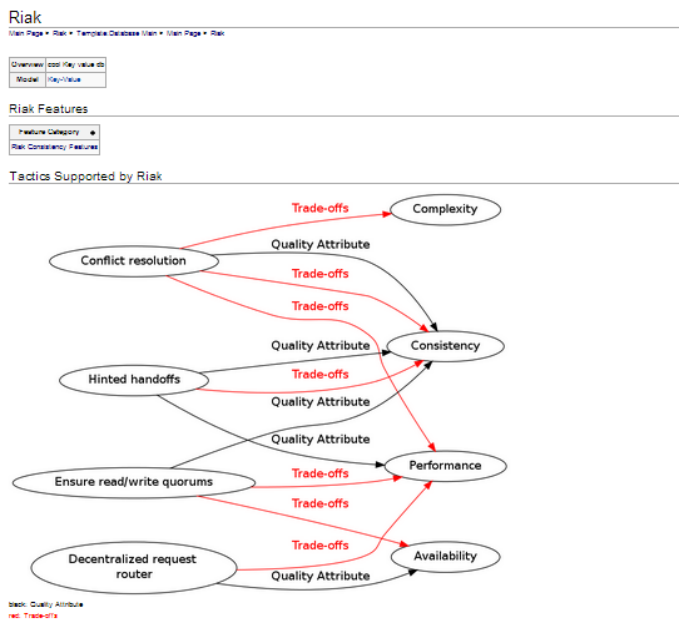
populating. So, when you fill in this page, you're actually populating the semantic model without knowing it. And once we have this information in the wiki, we can query based on that semantic model and build pages. So, it's a very rigorous controlled way of constructing this knowledge.

QuABase Demo 5



**076 So, again, back to the page here. This is the same page as I had earlier. We can see that there's three technologies that support quorum based consistency. One of them is Riak. So, let's just assume I'm interested in Riak.

QuABase Demo 6



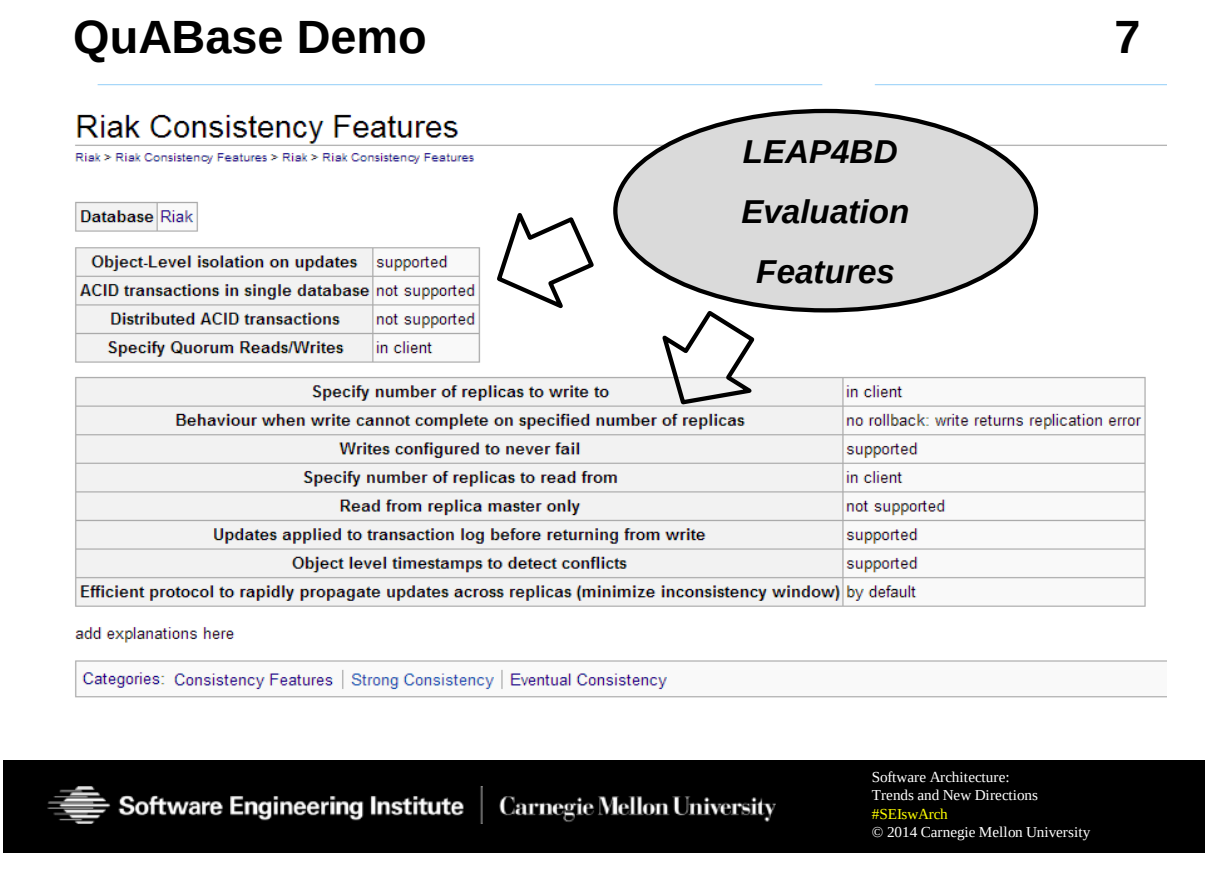
**077 How does this work? So, I click on the Riak page. And again, this is under development so the list of Riak features is minimal at the moment because we just haven't entered all the data or built the forms. But you can see that there's a visualization on the Riak page of the qualities that it supports, consistency, performance, availability and how it supports them through the tactics. And the tactics are actually related to the qualities in two ways, either in supporting or trading off.

And that's a live visualization. So, if you're on the webpage, you could click on the visualization to go to any

of the links that it displays. It's not just an image.

So, this gives me a visualization of the capabilities of Riak and its tradeoffs and its connections to providing the qualities I might be interested in. If I click on the consistencies features box towards the top there, I see a page that right now looks a little bit like this.

QuABase Demo 7



**078 And it's basically an enumeration of generic consistency features that are important within building these systems and how Riak supports them or, in many cases, doesn't support them which would be

expected because these are generic capabilities. And not every database is going to support them.

So, this is how we're building up the knowledgebase. Essentially, there's a set of features associated with databases that can be related to tactics. And the tactics, the solutions to particular qualities, can be dynamically related to the quality attribute scenarios.

What's important though, and this is where we go back to the slide I had earlier with the technology and knowledge overlapping, is this page is the confluence of that. This is where the Leap4BD evaluation criteria and the QuABase wiki approach come together. So, we basically can support people going in and understanding what databases they might be able to use to support their quality attribute requirements through going in through the quality attribute path. Or people can say, "Hey man, I'm just interested in Riak. What can I do with it because I've already got it here already?" and I can go into Riak and then I can navigate back through to understand the design knowledge that's associated with using Riak.

So, this is the work we're actually involved in right now.

Status

LEAP4BD

- Initial trial with DoD client near completion
- Rolling out as an SEI service

QuABase

- Design/development in progress
- Validation/testing over summer

Software Engineering for Big Data Course (1 day) and tutorial (1/2 day)

- SATURN 2014 in Portland, May 2014
  - http://www.sei.cmu.edu/saturn/2014/courses/
- WICSA in Sydney, Australia April 2014
- Both available on request

**079 Leap4BD is actually in a pilot with one of our clients. And we should be completing that pilot relatively soon. The knowledgebase is pretty much populated for the databases that we've been working with. And so, we'll then be rolling this out as an SEI service for people who are interested in doing detailed evaluations of big data technologies to provide high levels of assurance that their solutions will scale over time.

For the QuABase, we're actually actively developing the content right now. Developing the wiki pages actually involves quite a lot of code

development and query development. We hope to have a first version of this finished by the end of summer ready for validation and testing with some friendly users.

And in terms of rolling out this information to broader audiences, at Saturn, as you've all heard about this morning, we actually have a one day course in Portland. I forget the exact day. But we've already got quite a few people enrolled in that course now. So, if you're interested in this information and understanding the quality attributes and the mechanisms associated with building big data systems, this would be a great course to attend.

We also have a stripped down version of the course as a half day tutorial which I'll be presenting in Sydney at the working International Conference on Software Architecture in a couple of weeks. So, if any of you happen to be Australia, come on down. And of course, both of these would be available on request in the-- the near future.

Thank you!

Thank you!

http://blog.sei.cmu.edu/

OCT 21 2013 — Addressing the Software Engineering Challenges of Big Data

JAN 13 2014 — The Importance of Software Architecture in Big Data Systems

This document is available in the event console materials widget

**080 So, that's pretty much it. Thanks for listening. If you're interesting in some of the work that we're doing, there's a couple of blog posts that we've put on the SEI blog quite recently. And there's a paper which I believe Shane is providing through the webinar mechanisms. If you're interested, it's just been accepted by IEEE Software. So, we're quite happy about that. It's trying to describe some of the fundamental characteristics of big data architectures.

Thank you.

Shane: Okay, Ian. Great presentation. Thank you very much. We've got lots of questions coming in. So, we'll dive right into them.

Andre asks how to scale not scare consumers with the data tsunami.

Ian Gorton: That's a big question. I think you've just got to understand what data that your consumers, your users, need and somehow design your system so that you can provide that data, the primary data sources of interest to the user community without overwhelming them with the volume of data. And perhaps, then if you can structure your solution so the primary data sources they're interested in can then enable them to navigate through the rest of this massive amount of data that people are accumulating. Again, so maybe that's just about really understanding your scope. And if you try to answer everybody's question into the data, you're just going to design a system that's going to be very hard to scale.

Shane: Okay, great. Barak asks, "Should we use big data solutions on transaction critical systems?"

Ian Gorton: That's another big question. So, there's a really interesting blog post I think it was from Ed Brewer of the CAP theorem, the originator of the CAP theorem, talking about how banking systems, the classic example of a strong consistency problem, are actually eventually consistent. And the example is if you go to your ATM and

try to get some money out, you can get money out without actually knowing if you've got any money in your bank account. If that ATM is disconnected through a network partition to the backend databases, the ATM will still give you money.

It might not give you as much money as you want. There's a failsafe built into this. It might only give you a hundred bucks or two hundred bucks. But eventually, once you've taken that money out, that transaction will be resolved against your backend system. And hopefully, you've got enough money. And if you haven't, you go into an overdraft situation and get charged some exorbitant fee by your bank.

But it just shows that you can do what we consider a strongly consistent problems in eventually consistent ways. Some of the NoSQL technologies are actually very strong in providing consistency, such as transactional problems that the question alludes to. Things like Vault for BD and Translation DB will provide you with strong consistency mechanisms. Neo for J is a fully consistent, strongly consistent, acid consistent database. But it probably doesn't scale as well because of its nature of handling graphs. So, I think you can do it. Sometimes you have to be creative.

Shane: Okay, Bruno would like to know, "Where did Ian get the quality attributes for big data shown some slides before where it was talking about consistency, distribution, and the like?"

Ian Gorton: Where did they come from? From research essentially. I mean myself and John Kline who's my collaborator on this at the SEI, we've both got extensive experience in building database systems. I've build a lot of systems for scientific data management. And so, these are the-- we know from our experience are the sorts of questions that you have to ask when you're starting to design these systems. And hopefully, our experience and no doubt we don't have the complete experience, and we'll steal information of other people, but when the QuABase becomes available, this information will be available for everybody to consume. And hopefully, the expertise that it collectively expresses will just be instantly available through essentially a wiki to everybody.

Shane: Okay, great. Rob asks, "What is the threshold to become classified as big data, a big data problem, i.e. is it one million lines of code, one billion? Is there a threshold?"

Ian Gorton: I don't think there is actually. Some big data systems are complex because of the amount of data they can handle. But often, you're just querying it in very simple ways. And scaling that is not too difficult. But some systems are much more complex because of the heterogeneity of the data and the way the data kind of fuses together and needs to be navigated in complex ways.

So, we used to build systems in biology where the amount of data actually wasn't that significant. It was in the terabytes, many tens of terabytes range. But the analyses that we had to do and the links we had to provide across these different types of data that came from scientific instruments and analyses from simulations and analytics was very, very complex. So, I would still claim that was a big data solution. We used technologies like HBase to build it. But it wasn't massive in terms of scale. So, I think there's so many dimensions to scalability, it's hard to say exactly what a threshold might be.

Shane: Okay. I think you just touched on this, but I'll ask it anyway just in case. From Sridhar asking, "How's big data changing the way EDW solutions are built? Is there a minimum data size for structured or unstructured data that should be used to use big data design?"

Ian Gorton: I think one of the insights-- I'm not sure about minimum sizes, but one of the nice insights into this world is Martin Fowler's Polyglot Persistence notion whereby we're still going to be using enterprise data warehouses, the big Oracles and MySQL data warehouses and R schemas that we already have. And we're probably still going to be populating these things. But we might also be having many different types of repositories that augment those data warehouses.

So, you might have a very large Hadoop system to do certain sets of analytics that maybe eventually feed information into your data warehouse. Or you may have other types of databases to solve other workloads. And the notion of polyglot persistence is that traditionally we've just used relational databases as a single homogeneous approach to providing persistence at the organizational level. But now in the NoSQL world, the big data world, we're actually starting to pull together systems that use different types of database technology that are specific to solving the particular piece of the puzzle that the enterprise needs to solve.

The data warehouse is still going to be there in nearly every instance, I suspect. But what's around the edges of it may look very different as you start to flesh out and architect your big data system.

Shane: Okay, I think this question was asked before you got to the end, but a reminder's always good. From Victor asking, "Where can I find more information about Leap4BD?"

Ian Gorton: Right now, you'll have to talk to us I'm afraid. So, this is essentially in its initial trial. So, we've been fleshing out the method, making sure it works appropriately with our clients. And so, we'll be kind of making this into a product in the next two, three months as we finish off our initial trial. So, if you're

interested, please talk to us. We'd love to talk to people who are interested in using the method.

Shane: Okay, Rob asks, "How do you intend to keep the QuABase up to date? Technology in NoSQL is moving so quickly."

Ian Gorton: Good question. So, with things like Wikipedia, obviously there's an open contribution model which is kind of monitored by people. And I don't think that will work for something like the QuABase. So, we-- even though you're right that the technology does move very quickly and it's vast. So, covering the vastness is probably going to be difficult. Once you have a particular technology characterized within the QuABase, it doesn't change that quickly to be honest. Releases will come and go and minor details will change. But the fundamental data model of MongoDB isn't going to change. And more than likely, the way that MongoDB does sharding through shard keys and enables you to move chunks of data around is not going to change every two weeks. So, there's an awful lot of stability within this knowledge which we can exploit.

I think we'll have to have some sort of controlled update mechanism where we have experts in particular areas that can send us updates and it gets refereed by some cohort of experts or whatever or controllers of the wiki. Whatever that may be, we haven't actually thought through the

model yet. But I think the pace of change is manageable. And of course, the core design knowledge doesn't change that much. It probably grows, but the fundamentals don't change. Computer science doesn't change once we know it that much.

Shane: Okay, William writes, "Do you have a graphic that shows the difference between NoSQL and SQL performance?"

Ian Gorton: No, but I'm sure if you dug around the Internet, you could find some specific examples. Obviously, because any comparison of performance is based on the actual workload that a particular system is serving, so I read only workload might reveal very different differences between a NoSQL and a SQL technology. A write workload, I suspect if you compared Cassandra's write performance to a relational database, you'd see Cassandra is much faster. But it may only be providing you with very weak consistency.

So, there's always complexities in doing these comparisons. And I think where a lot of organizations fail is that they don't look deeply enough into these comparisons. And there's lots of very superficial blog posts on the Internet, hopefully, ours aren't those, that just say hey man, use this. It's really cool. I did a test where I ran a four line piece of code. And it was really good, man. So, yeah it's very difficult to do this in the abstract.

Shane: Okay, from Christopher, "Does SEI have a body of knowledge for architecture in general similar to QuABase?"

Ian Gorton: There's an awful lot of knowledge from the SEI expressed in the book that have been published. The Software Architecture in Practice book has a third edition that was published last year I think. There's books on documenting architecture knowledge and evaluating architectures. Perhaps the QuABase is a little bit different in that it's very domain specific and technology specific. So, perhaps it's best to characterize the SEI knowledge as being very broad and fundamental up until now. And we're taking a very thin slice of that knowledge and diving much more deeply into it.

Shane: Okay, we'll take two more questions. One was just a question from Joe asking if a recording of these sessions are available. And the answer is yes. The whole day was archived. And recordings more likely will be accessible tomorrow. It was the same login information you used today. Again, all the PDFs of the slides and other materials are available on the materials widget now. And we've got a minute left for Ian. We'll ask one more. We've got lots of other questions that hopefully we can address at another time. But we'll stick to our commitment of twelve thirty. So, from Dan asking, "Will the data about each NoSQL system be available in any standard format like XML or RDF?"

Ian Gorton: Our initial version will record the data inside the wiki. But from the wiki, we can export into RDF because the underlying semantic representation is essentially RDF triples. So, I think the answer is probably yes.

Shane: Ian, excellent presentation. Thank you. Folks, that's going to wrap up our virtual event for today. We thank each and every one of you for attending.

Copyright

**082