Twitter Shifting More Code to JVM, Citing Performance and Encapsulation As Primary Drivers

Posted by **Charles Humble** on Jul 04, 2011

Community
Java
Topics
Object Oriented Design ,
Performance & Scalability
Tags
Scala ,
JVM ,
Ruby on Rails ,
Lucene ,
JRuby ,
Java SE

While it almost certainly remains the largest Ruby on Rails based site in the world, Twitter has gradually been moving more and more of its stack to the JVM. The change is partially motivated by oft-cited advantages of the JVM, such as performance and scalability, but is also driven by a desire for better encapsulation of individual services, and other architectural concerns.

Last year the company announced that both its back-end message queue and Tweet storage had been re-written in Scala, and in the spring of 2010 the search team at Twitter started to rewrite the search engine. As part of the effort, Twitter changed the search storage from MySQL to a real-time version of Lucene. More recently the team announced that they were replacing the Ruby on Rails front-end for search with a Java server they called Blender. This change resulted in a 3x drop in search latencies.

InfoQ spoke to Twitter engineer Evan Weaver to find out more about the background to the change.

Twitter Architectural Overview

One of the overall observations one can make from looking at Twitter's architecture is that many of the design decisions are admirably pragmatic. So for example, Twitter's back-end uses both MySQL and Open-source distributed database Cassandra extensively. Gizzard, its own open-source framework for creating distributed datastores, is used to partition MySQL. This is "mainly used for highly structured, very high SLA data", Weaver told us, "because it is relatively inflexible".

All run-time data is served from either Gizzard/MySQL, or Cassandra. Twitter also uses HDFS in Hadoop extensively for off-line computation, and is bringing online a system that uses Gizzard to partition the key-value store Redis.

> We had existing schemas in MySQL that worked, so we kept and sharded them through Gizzard rather than moving to a system with a different performance profile. But Cassandra is so much more flexible, we've had a really good experience using that for new things.
>
> For the mid-tier, more flexible kind of data concerns, like new product features, time series data particularly, things that need a really high write velocity, like serving of derived calculations from Hadoop, that kind of thing, we use Cassandra very extensively.

Communication between front-end and back-end services uses the Facebook developed Thrift as the RPC mechanism, and JSON over REST as the public RPC, which is also used for Twitter's own clients including the new Twitter website.

Language Choices

A similar pragmatic approach can be seen in language selection. The first class languages at Twitter are JavaScript, Ruby, Scala and Java. They also support C, but rarely write new services in it. Generally, Weaver told us, developers coming from a Ruby background tend to prefer working in Scala, whilst developers coming from a C or C++ background choose Java.

In the case of the search team, since they do a lot of work on Lucene, which is Java-based, they have a lot of experience in writing Java code. As such it is more convenient for them to work in Java than Scala or another language.

To allow developers to choose the best language for the job, Twitter has invested a lot of effort in writing internal frameworks which encapsulate common concerns. Finagle, for example, is a library for building asynchronous RPC servers and clients in Java, Scala, or any JVM language. It is written in Scala, but also supports a highly Java-idiomatic API.

As the back-end code is being pulled towards the JVM, the front end client code, in common with many contemporary web based applications, is gradually making heavier and heavier use of browser-based JavaScript. In consequence the Ruby component is shrinking.

> We were originally a Rails shop, and I believe we are the largest Rails site in the world, but as we've grown as an organization, and as a service, performance and encapsulation have become very critical. I wouldn't say that Rails has served as poorly in any way, it's just that we outgrew it very quickly. So there are two things about Rails that make it no longer ideal for our situation.
>
> First, the Ruby runtime is slow, particularly in comparison to the JVM. We've worked hard on the garbage collector to get reasonable performance.
>
> And also the LAMP model that Rails embodies, where you have a set of tiers each of which only talks to the one above and below, and no vertical encapsulation, doesn't serve a large organization like us very well.
>
> As we've been focusing on performance and encapsulation, we've fixed performance problems as necessary, with caches, or working on the VMs.
>
> The majority of requests on Twitter go through Rails right now, but as we build new services, if we choose to build them from scratch, in order to achieve better encapsulation we move them into the JVM, because the performance concerns outweigh any sort of productivity or agility downside those languages might have. So when we re-built Tweet storage we built it in Gizzard as a homogenous service, it exposes a domain interface, and that's a Scala system that partitions and manages uncoordinated MySQL nodes. So that effectively eliminated ActiveRecord use for tweets from the core Rails stack.
>
> The same with the queue; when we wanted to rebuild it and re-encapsulate it for performance reasons we wrote it on JVM. So as those kind of lightweight, service-oriented projects proceed, more and more concerns are being taken out of the core Rails application.
>
> On the opposite side, as we've moved the render code into browser-based JavaScript, we no longer get much benefit from Rails' templating model for building web pages. So we're pulling concerns out from both sides, and when rewrite them it makes sense to rewrite them in a faster stack, because performance is so critical for us. We're one of the largest websites in the world, but run on a very small hardware footprint compared to other big dynamic sites.

Keeping the hardware footprint small has advantages in terms of cost, but also avoids some of the secondary scaleability concerns, such as the performance of the TCP stack, that can impact sites with larger hardware demands.

You might assume that the move to the JVM was largely driven by performance and scalability concerns, but in fact the existing Twitter codebase performs well. As such the company isn't being forced into a ground-up re-write to allow it to continue to grow. Rather, the move to JVM is driven as much by a need for better developer productivity as it it for better performance.

> The primary driver is honestly encapsulation, so we can iterate faster as a company. Having a single, monolithic application codebase is not amenable to quick movement on a per-team basis. So when we decide to encapsulate something, then because of our performance concerns, its better to rewrite it in the JVM for most systems, than to write a new Ruby system.
>
> That aside, because we rely on Ruby so heavily, we've put a lot of investment into our existing infrastructure, and it works well for us.

Search: From Ruby to Java

The transition from Ruby to the Java-based Blender server was effectively done in two stages. The first was to replace the MySQL back-end with a real-time reverse index the search team developed based on Lucene, called Earlybird. Earlybird doubled the memory efficiency and provided the flexibility to add relevance filtering for search, helping to support the rapidly growing demand on the search service. According to an engineering blog post

> In 2008, Twitter search handled an average of 20 TPS [tweets per second] and 200 QPS. By October 2010, when we replaced MySQL with Earlybird, the system was handling 1,000 TPS and 12,000 QPS on average...However, we still needed to replace the Ruby on Rails front-end, which was only capable of synchronous calls to Earlybird and had accrued significant technical debt through years of scaling and transition to Earlybird.

To solve this the team began development of the Java Blender server. Blender is a Thrift and HTTP service built on Netty, a highly-scalable New I/O (NIO) client server library written in Java that enables the development of a variety of protocol servers. Netty allows Twitter to create a fully asynchronous aggregation service, which can aggregate the results from several back-end services such as the indices for real-time, top tweet and geo. From the engineering blog:

> Netty defines a key abstraction, called a Channel, to encapsulate a connection to a network socket that provides an interface to do a set of I/O operations like read, write, connect, and bind. All channel I/O operations are asynchronous in nature.
>
> This means any I/O call returns immediately with a ChannelFuture instance that notifies whether the requested I/O operations succeed, fail,

> or are canceled.
>
> When a Netty server accepts a new connection, it creates a new channel pipeline to process it. A channel pipeline is nothing but a sequence of channel handlers that implements the business logic needed to process the request.

These pipelines are then mapped to a set of back-end services, automatically handling transitive dependencies between them. Throughout the workflow process, there are no thread busy-waits on I/O, making efficient use of the CPU and allowing support for a large number of concurrent requests. In addition, many requests to the back-end services can be handled in parallel, significantly reducing latency.

Performance Gains

Twitter's search is one of the most heavily-trafficked search engines in the world, serving over one billion queries per day. The impact of Blender has been dramatic

> Following the launch of Blender, our 95th percentile latencies were reduced by 3x from 800ms to 250ms and CPU load on our front-end servers was cut in half. We now have the capacity to serve 10x the number of requests per machine. This means we can support the same number of requests with fewer servers, reducing our front-end service costs.

Static Typing as a Productivity Boon

While performance and scalability are often cited as benefits of using a JVM-based language, Twitter is also finding benefits in static typing, at least for its back-end services. Weaver told us

> I would say about half of the productivity gain is purely because of accumulated technical debt in the search Rails stack. And the other half is that, as search has moved into a Service Oriented Architecture and exposes various APIs, static typing becomes a big convenience in enforcing coherency across all the systems. You can guarantee that your dataflow is more or less going to work, and focus on the functional aspects. Whereas for something like building a web page you don't want to recompile all the time, you don't really want to worry about whether in some edge condition you are going to get a type you didn't expect. But as we move into a light-weight Service Oriented Architecture model, static typing becomes a genuine productivity boon. And Scala gives you the same thing.

The ability to iterate faster is obviously critical for the company. In the case of search, the team has added relevancy filtering and personalisation to the Twitter website to improve the quality of results, and have extended Earlybird's data structures to support efficient lookups of entities contained in Tweets, such as images and videos. They are working on relevancy filtering for mobile, and are continuing to work on improving the scalability of the search infrastructure and the quality of its results.

Ruby MRI vs JRuby

Whilst in many cases it makes sense for Twitter to simply move away from Ruby to Java or Scala, there are services where Ruby is still the best choice. Twitter is currently based on Ruby MRI (also known as CRuby) version 1.8, albeit with a heavily re-written garbage collector. They are planning to evaluate the effort involved in moving to Ruby 1.9, and re-implementing the custom garbage collector, versus the effort involved in moving the Rails code they don't plan to re-encapsulate to JRuby.

> The big issue is that the performance impact of moving to JRuby is bound up in the quality of the various clients that you rely on. So, for example, our memcached client for CRuby is extremely fast. JRuby clients, as far as my understanding goes, are not even within an order of magnitude as performant. So even if JRuby itself is twice as fast, moving to a slower memcached client would destroy all that performance benefit.

To fully evaluate JRuby Twitter would need to re-write their Thrift client, memcached client, possibly their MySQL client and so on, before they can tell if it really is a benefit for them.

> That's not a fault of JRuby; it's just that at the moment the surrounding ecosystem is still kind of immature. CRuby's was too; we put a lot of investment into it, which we can now take advantage of, and we would have to do the same for JRuby.

Conclusion

The combination of Ruby on Rails and MySQL has been a popular one for start-up companies for the last several years. It is a sound choice in many cases, allowing a company's engineers to rapidly try out small new ideas and see which of them find traction in the market place. It does however come with well known costs, both in terms of performance and scalability, and perhaps also the relative maturity of the libraries and tool chain. In addition, the experience at Twitter suggests that the Ruby on Rails stack can produce some significant architectural challenges as the code base grows.

About the Author

Charles Humble (@charleshumble on twitter) is the CTO for PRP*i* Consulting with overall responsibility for the development of all the custom software used within the company. He has worked in enterprise software for around 15 years as a developer, architect and development manager. He co-founded Conissaunce, a UK based enterprise computing consultancy focused on the retail and financial services industries, and remains a director of the firm. He spends as much time as he can with his young family, and writes music with twofish.

11 comments

Watch Thread Reply

Conclusion is wrong by matt mcknight Posted 06/07/2011 09:18
Re: Conclusion is wrong by Brian Edwards Posted 06/07/2011 11:39
Re: Conclusion is wrong by matt mcknight Posted 07/07/2011 00:00
Re: Conclusion is wrong by Eric Weise Posted 07/07/2011 11:55
Re: Conclusion is wrong by William H Posted 08/07/2011 03:12
Re: Conclusion is wrong by Daniele (Dan) Mazzini Posted 11/07/2011 07:30
Re: Conclusion is wrong by matt mcknight Posted 13/07/2011 10:10
Re: Conclusion is wrong by William H Posted 13/07/2011 03:35
Re: Conclusion is wrong by matt mcknight Posted 13/07/2011 10:07
Is Erlang is a better one than java by dileep stanley george Posted 13/07/2011 02:44
Small companies should definitely go for rails. by Surendran Sujith Posted 17/07/2011 12:00
Sort by date descending

1. Back to top

Conclusion is wrong

06/07/2011 09:18 by **matt mcknight**

Your conclusion is completely wrong and horrible advice for just about anyone. What makes it most clear is that you offer no positive alternatives. Are you actually suggesting that people write their own web server like Twitter did? No, you're not suggesting that because you write of the importance of the maturity of the toolchain...

What Twitter found is that they wanted to rewrite their search stack to go from a synchronous to asynchronous/evented architecture. They wrote everything from scratch on top of netty. It's not like they switched to Spring MVC, they wrote their own web application server (Blender). They didn't even switch the main application, which is still trucking along...

The very telling point is that if you haven't gotten to the point of using memcached on your project yet, your web stack probably isn't the bottleneck.

If you want to switch to a more asynchronous architecture, why not look at Goliath? github.com/postrank-labs/goliath It's a great implementation of the reactor pattern. Or Node.js?

Reply

2. Back to top

Re: Conclusion is wrong

06/07/2011 11:39 by **Brian Edwards**

Well I enjoyed the article. I guess Matt read the article while in a bad mood.

Reply

3. Back to top

Re: Conclusion is wrong

07/07/2011 00:00 by **matt mcknight**

I found the source Twitter articles useful, and had read them previously. I found Mr. Humble's attempts to extrapolate general conclusions from them quite weak. His biases show through so clearly, that reading this article will lead people to misunderstand the source material.

As an example, Humble states "Rather, the move to JVM is driven as much by a need for better developer productivity as it it for better performance."

This is the absolute wrong conclusion to draw. What the Twitter guy said was that they wanted to break their monolithic application into many smaller applications- encapsulation. This did not require a move to the JVM. The Twitter guy said that when they encapsulated, they moved some pieces to a JVM platform for performance- particularly because where they wanted to use Netty and non-blocking I/O.

Reply

4. Back to top

Re: Conclusion is wrong

07/07/2011 11:55 by **Eric Weise**

As an example, Humble states "Rather, the move to JVM is driven as much by a need for better developer productivity as it it for better performance."

"This is the absolute wrong conclusion to draw."

Huh? What about this comment?
"But as we move into a light-weight Service Oriented Architecture model, static typing becomes a genuine productivity boon. And Scala gives you the same thing."

Reply

5. Back to top

Re: Conclusion is wrong

08/07/2011 03:12 by **William H**

Not only that but the motivation for better encapsulation is to be able to "iterate faster as a company." in other words better developer productivity.

Reply

6. Back to top

Re: Conclusion is wrong

11/07/2011 07:30 by **Daniele (Dan) Mazzini**

You can make a pretty big distinction between "developer productivity" e company wide development productivity: For a single developer, it's very difficult to argue that Ruby on Rails isn't (much) more productive than any Java web development solution.

If you have a large project with many teams, though, individual productivity becomes less important than the general productivity of the whole company, and even small bugs can have a big impact because of the bigger and bigger overhead in communication and coordination between teams.

So it's both possible that RoR is more productive for the developer, and Scala is more productive for the whole company, depending on the task...

Reply

7. Back to top

Is Erlang is a better one than java

13/07/2011 02:44 by **dileep stanley george**

Is Erlang is a better one than java

Reply

8. Back to top

Re: Conclusion is wrong

13/07/2011 10:07 by **matt mcknight**

As an example, Humble states "Rather, the move to JVM is driven as much by a need for better developer productivity as it it for better performance."

"This is the absolute wrong conclusion to draw."

Huh? What about this comment?
"But as we move into a light-weight Service Oriented Architecture model, static typing becomes a genuine productivity boon. And Scala gives you the same thing."

Because you don't get static typing by running on the JVM- see groovy, jruby, clojure, jython, etc.; you get it from languages. You can get static typing from, say, c++. So, the argument by the author of the article that the JVM is a developer productivity boon does not follow from the statements you are basing it on.

Reply

9. Back to top

Re: Conclusion is wrong

13/07/2011 10:10 by **matt mcknight**

"So it's both possible that RoR is more productive for the developer, and Scala is more productive for the whole company, depending on the task..."

That's an interesting response to the encapsulation issue- which is language independent. You could build a monolithic application in any language. Most languages allow you to build a modular application.

Reply

10. Back to top

Re: Conclusion is wrong

13/07/2011 03:35 by **William H**

> That's an interesting response to the encapsulation issue- which is language independent. You could build a monolithic application in any language. Most languages allow you to build a modular application.

Again to be clear the reason for the desire for better encapsulation comes from a need for better developer productivity as both Weaver and Humble state:
"Having a single, monolithic application codebase is not amenable to quick movement on a per-team basis"

The point that Weaver makes is that Ruby when used in conjunction with Rails doesn't lend itself to the design they found that they now needed:
"...the LAMP model that Rails embodies, where you have a set of tiers each of which only talks to the one above and below, and no vertical encapsulation, doesn't serve a large organization like us very well."
So could you have got there with Ruby? Possibly. You could probably have got there with postscript come to that. Twitter clearly just felt that getting there with Java was simply easier.

Matt - you also state "you don't get static typing by running on the JVM."

This is incorrect - the JVM is, as of today, statically typed. Dynamically typed languages that want to target the JVM have to resort to all sorts of tricks to get dynamic typing to work. This will change a bit with Java 7, when it ships, since it includes an invokeDynamic instruction specifically for this purpose.

Reply

11. Back to top

Small companies should definitely go for rails.

17/07/2011 12:00 by **Surendran Sujith**

I believe Charles Humble has just narrated the twitter experience and explained their reasons for choosing not selecting Ruby for their specific case. In fact one of reasons assigned is the availability and skill of their existing resource. Charles Humble doesn't necessarily argue against Rails but in a way it is a suggestion that small timers would definitely benefit using rails. Further Ruby and Rails has grown much beyond the status of what it used to be. The sole reason that Rails drastically reduces development time and effort is enough for most people to go for rails.