→

RECENT POSTS

Kinsta, Bridgecrew, IP2Location, StackHawk, InterviewCamp.io, Educative, Stream, Fauna, Triplebyte

- Architecture of Max reHIT Workout
- Sponsored Post: Wynter, Pinecone, Kinsta, Bridgecrew, IP2Location, StackHawk, InterviewCamp.io, Educative, Stream, Fauna, Triplebyte

Monday
Aug152016

How PayPal Scaled To Billions Of Transactions Daily Using Just 8VMs

MONDAY, AUGUST 15, 2016 AT 8:56AM

How did Paypal take a billion hits a day system that might traditionally run on a 100s of VMs and shrink it down to

run on 8 VMs, stay responsive even at 90% CPU, at transaction densities Paypal has never seen before, with jobs that take 1/10th the time, while reducing costs and allowing for much better organizational growth without growing the compute infrastructure accordingly?

PayPal moved to an Actor model based on Akka. PayPal told their story here: squbs: A New, Reactive Way for PayPal to Build Applications. They open source squbs and you can find it here: squbs on GitHub.

The stateful service model still doesn't get enough consideration when projects are choosing a *way of doing things*. To learn more about stateful services there's an article, Making The Case For Building Scalable Stateful Services In The Modern Era, based on an great talk given by Caitie McCaffrey. And if that doesn't convince you here's WhatsApp, who used Erlang, an Akka competitor, to achieve incredible throughput: The WhatsApp Architecture Facebook Bought For $19 Billion.

I refer to the above articles because the PayPal article is short on architectural details. It's more about the factors the led the selection of Akka and the benefits they've achieved by moving to Akka. But it's a very valuable motivating example for doing something different than the status quo.

What's wrong with services on lots of VMs approach?

- **Services use very small VMs and produce very low throughput for each VM.** Actor based reactive systems shine at efficiently using compute resources. So you can shrink your system way down rather than rely on the typical auto-scaling monstrosity.
- **Puts a lot of pressure on network and routing infrastructure.** As services tend to be interconnected, requests can go through a lot of hops, which increases latency and decreases the user experience.
- **Larger is more costly.** Services spanning hundreds of VM have an high inherent cost in terms of management, monitoring, and ineffective caching.
- **Smaller is more agile.** It takes a long time to deploy services across hundreds of VMs.
- **Make better use of more CPUs per VM**. Since CPUs aren't getting faster your infrastructure needs to able efficiently exploit more CPUs per VM.
- **Microservices need to be built upon loosely-coupled nanoservices that are easy to maintain**

and quick to build. You don't want layers and layers of complexity. You need good visibility into what a service does. You should not have to dig into layers and layers of code to figure it out.

Given the above forces PayPal wanted a system with the following characteristics:

- Scalable, both horizontally to hundreds of nodes and vertically to very many processors, handling billions of requests per day
- Low latency, controllable at a very fine grain
- Resilient to failure
- Flexibility in adjusting the service boundaries
- A programming model AND culture encouraging scalability and simplicity, including clean failure and error handling.

It's clear PayPal wanted a thinner stack. They didn't want a stack with lots of layers and moving parts. Akka and state based systems in general are good for that as they collapse a good chunk of the stack down to one technology. PayPal chose Akka over Erlang because they have a lot of Java experience and Akka runs on Java. For many having to learn Erlang is a non-starter.

With Akka they could:

- write code that is easy to reason about
- write code that's easy to test
- handle errors and failure scenarios more naturally when compared to the traditional model used on the JVM
- write faster, resilient, and simpler code with streamlined error handling and fewer bugs

So of course PayPal immediately wrote their own framework on top of Akka, as one does, called squbs, rhymes with cubes, that creates a modular layer for building nano-services called “cubes”. Cubes are symmetric to other cubes, the interdependency between cubes are loose and symmetric, and only expose the messaging interface already provided in Akka.

The article brings up the difficulty of programmers adapting to the non-linear nature of Akka code, so you have to hire people that can be trained program in Akka/Scala.

Since most services do similar things--receive requests, make database calls to read/write the database, make other service calls, call a rule engine, fetch data from cache, write to cache--they were able to abstract that out

using patterns like the Orchestrator Pattern and Perpetual Stream.

Squbs has become the standard for building Akka-based reactive applications at PayPal. So if you haven't considered stateful systems for your team, give them another look. It has worked for PayPal, Facebook, Uber, and Microsoft.

Related Articles

- On HackerNews / On Reddit
- Squbs sample java application
- The Inevitable Rise of the Stateful Web Application
- Akka in Action

Reader Comments (6)

"..who used Erlang, an Akka competitor..." ... LOL.. WAT?!

That's a pretty gross misrepresentation of the Erlang - Akka relationship isn't it? They don't compete, and if they did, Akka would be Erlang's competitor based on primogeniture :)

Jonas Bonér: "We borrowed heavily from good ideas around computer science. We borrowed, for example, actors from Erlang. That forms a foundation of Akka, both in terms of concurrency and scalability. Remote actors are an excellent tool for doing almost transparent location, independent distributed computing and also for fault tolerance in which we embraced the "let it crash" supervisor hierarchy model, in which the system can watch itself and repair itself as it's running—self-healing in a way." - http://www.artima.com/scalazine/articles/akka_jonas_boner.html

August 15, 2016 | Tristan Slominski

This is the sort of article I really enjoy seeing here - useful, informative, not-too-intimidating-to-beginners. Good work!

August 15, 2016 | Neil Lopez

What do you mean by "transaction"?

August 17, 2016 | asd

"transaction" means someone buys or sells something, using PayPal.

August 22, 2016 | Kukil

Using "Transaction" always annoy me - when its not clearly defined.
A "Transaction" can be a tiny atomic unit of work, or a large block.
Consider these two simple transactions:
"Am I logged on"
"Transfer $1000 from my US checking account to my Cayman Islands Account, and convert to Euros"

Clearly, they take a different amount of time - perhaps quite considerably.

December 17, 2016 | Alain

"Microservices need to be built upon loosely-coupled nanoservices that are easy to maintain and quick to build. You don't want layers and layers of complexity."

Isn't this a contradiction? Microservices already suffer from layering complexity (see the famous Amazon rant by a Google employee) so the approach to solving it is to...further break them down into "nanoservices"? Wouldn't that introduce even more layers, and thus even more complexity? Or am I misinterpreting this new buzzword?

August 23, 2017 | zen