Building Scalable Systems Using Data as a Composite Material

Monday, November 16, 2009 at 8:50AM

Todd Hoff in Strategy

Think of building websites as engineering composite materials. A composite material is when two or more materials are combined to create a third material that does something useful that the components couldn't do on their own. Composites like reinforced concrete have revolutionized design and construction. When building websites we usually bring different component materials together, like creating a composite, to get the features we need rather than building a completely new thing from scratch that does everything we want.

This approach has been seen as a hack because it leads to inelegancies like data duplication; great gobs of component glue; consistency issues; and messy operations. But what if the the composite approach is really a strength, not a hack, but a messy part of the world that needs to be embraced rather than belittled?

They key is to **see data as a material**. Right now we are arguing which is the best single material to build with. Is it NoSQL, relational, massively parallel, graph, in-memory, or something else entirely? It all seems a bit crazy. Each material has both limits and capabilities. What we need to think of building is a composite material that combines the best

characteristics of what is available into something better.

Composite Materials in the Meat World

In material science materials have a natural capacity, depending on their structure, to resist failure due to the forces of: shear, torsion, compression. These capabilities limit how high you can build a building or how much weight something can carry. To get around these limits composite materials are engineered that combine different materials together to create something new that has the best characteristics of its components.

To get a feel for the difference materials and engineering can make compare Medieval Romanesque churches against Gothic churches. Romanesque churches are short, thick, small windowed, and dark. Gothic churches in contrast are tall, open, and light, marking a new more confident and prosperous era. Gothic churches could be so much taller because of the invention of the pointed arch, ribbed vaults, and the flying buttress, all of which handle loads better which make it possible to build taller more open buildings. Better engineering and materials led to better buildings.

Skip ahead a few centuries. Tall, majestic skyscrapers, one of the most muscular symbols of modern civilization, were made possible in part by the invention of new compound materials. As we've seen in the past, buildings were height challenged because they needed walls thick enough to support their weight. In 1891, for example, a 16-story building constructed in Chicago had walls 6 ft thick at the base.

Skyscrapers became possible with the invention of steel-frame construction where a rigid steel skeleton supports a building's weight. Walls are now demoted to decoration and not the meatier job of support.

Another invention that made skyscrapers possible is reinforced concrete: the magical mixing of concrete + steel rebar. As a foreshadowing of how we keep creating larger components out of parts which themselves go into building more complex components, these materials are themselves composites: concrete is a mixture of cement and stone aggregate; steel is an alloy consisting mostly of iron and carbon; cement is clay and limestone.

What concrete brings to the mix, because of its chemical makeup, is rigidity. Rigidity makes concrete able resist compressive forces, which are forces that flatten or squeeze a material. A very good quality for tall, heavy buildings. Unfortunately that same rigidity makes concrete brittle when bent. Twisting concrete breaks it. Not a good thing for buildings subject to the natural twists of wind and earthquake. So concrete alone won't work. This is where rebar comes in. Steel shines because the interaction of carbon atoms gives it the ability to return to its original shape once a load is removed. When a force like the wind causes steel to twist, the steel will return to its original shape once the load is removed. A very good property for a building.

By combining concrete and steel a new material has been created that has the best qualities of each, it resists both compression and bending, which allows buildings of almost any shape to be created. This new material brought on an entirely new wave of architecture. The modern cityscape would be unrecognizable without it.

Composites in the Web World

Let's look at our material, data, in a little more detail. Data doesn't exist separate from purpose. No building technique treats data in a similar way. We have normalization, denormalization, unstructured, semi-structured, with relationships, without relationships, documents, key-value, and so on.

Which view is right? None of them. They are all views built on a purpose. It's the **purpose that shapes the material**. All views have a point because all the views are meant to carry out a different function. A compound material will need to **take data and allow it exist in many forms simultaneously**.

The key concept to see here is: there is **never a representation of data that is the one true representation**. Data is represented how it needs to be represented to carry out a particular function. The job of a website is to make sure data flows correctly between all these different functions.

There are a number functions common to sites: time series, geolocation, reading, writing, streaming, counters, aggregated data, graph navigation, ad-hoc query, monetization, job queue, complex analysis, tagging, logging, searching, events, social graphs, operations, and so on. There are also a number of cross cutting concerns that are common to each function and the interaction between functions: transaction model, programming model, cache coherence, data model, caching, real-time, batch, data size, input rates, output rates, bandwidth, storage, memory, CPU, availability, testing, and so on. If you look you'll find some sort of product in all these spaces.

Often the proposed solution to do away with this confusing world is to create a one-ring to control them all, as if that will finally bring order to the chaos. In the database world the one-ring is a product that does everything using just one external view of the data. In some towers if you even suggest that the one-ring has faults you are shouted down by the Ring Wraiths who say just wait, it all can be done by one system, have a little faith, or else it's Mount Doom for you.

Composing components is after all how real websites like Amazon, Flickr, Digg, and Facebook are put together. Look at the Real Life

Architecture section on HighScalability.com, you'll see company and after company building their website from a long list of components, most are usually Open Source, many are custom made for their situation. The secret sause of a site is often knowing how to forge all these components together into a single working system.

Ravelry is a good example. They use: Ruby on Rails, MySQL, Gentoo Linux, Capistrano, Nginx, Xen, HAproxy, Munin, Nagios, Tokyo Cabinet, HopToad, NewRelic, Syslog-ng, S3, Sphinx, Memcached, a bandwidth provider, a hosting provider, and a server provider. Across most websites this theme is the same, even if the components change.

Building specialized software is usually done when it involves a site's core competency. Digg and Linked In, for example, build highly custom in-memory social networking databases. That's their biz, so it makes sense. Facebook builds specialized software for image serving, chat, caching, replication, logging and everything else that really needs to scale. Again, that's their biz.

Custom software must always be fit into the rest of the system. You may have your own social networking database to make social networking operations fast, but you may also use a transactional database for financial data; a search platform like Sphinx to handle site searches; a cache for fast keyed lookups; and so on.

The alternative is to force functionality where it is implemented poorly. Is search really best handled in your RDBMS because you are worried about how to keep data consistent between the database and the search engine? Is it really better not to have a denormalized cache for fast access because it's hard to keep consistent with the normalized/transactional part of your system?

You could force one database to handle all these requirements. One mega-system could handle search, key-value access, and transactional functionality, but really, how well does that work? It hasn't worked well. Maybe it's better to build a best-of-breed system for each function and then learn how to keep the components in sync, handle transactions between components, and handle cache coherence between components.

The advantages of a composite architecture are:

- **Quality features**. Instead of accepting lesser functionality from a generic component you can build or install your own best-of-breed component.
- **Asynchrony between components**. Data can be flow between components using a job queue. A user update, for example, isn't made transactionally and synchronously between all components, it is put on a queue for each component and the component process the request when it has resources.
- **Independence under failure**. If a component fails the other components keep on working, so the system is still highly available with respect a large number of features.
- **Component based construction**. Services are abstracted by service layer APIs. Applications are created by weaving together different services. This is a common construction paradigm for modern websites. See Amazon for an example.
- **Partitioning**. A single machine can't handle a large load so an application needs to be broken into parts and requests need to be balanced across those parts. A composite architecture naturally breaks up an application by function, so there is a natural partitioning.
- **It's how you really build stuff anyway**.

Biology uses a similar strategy of building larger systems from components providing very specific functionality. We often think of cells in our body as just one thing, but in reality cells are highly specialized. There are hundreds of different types of cells in the human body. Blood cells are specifically designed to carry oxygen, for example. We talk of neurons like there's only one kind of neuron. There are hundreds. In fact, regions of the brain were first categorized by noticing when cell types changed. Cells are collected together into larger structures called organs. Organs are collected together to make a body. Parts of the body talk to each using the blood stream to carry messages and through direct connections using the nervous system. It's a little messy and chaotic, but is seems to work.

The point of all this is to counter seeing any one technology as the saviour. About now there's a lot of excitement over NoSQL style solutions. There's a rush to toss out the relational database and replace it with one of the many NoSQL options. Replacement, however, may not suffice. Each data model has a weakness. The idea behind creating composite materials is that materials can be combined together to make something better and that this approach to building systems is not a hack, it may in fact be better.
