How to Succeed at Capacity Planning Without Really Trying : An Interview with Flickr's John Allspaw on His New Book

Monday, June 29, 2009 at 3:08AM

Todd Hoff in Book, capacity planning, operations

Update 2: Velocity 09: John Allspaw, 10+ Deploys Per Day: Dev and Ops Cooperation at Flickr. Insightful talk. Some highlights: Change is good if you can build tools and culture to lower the risk of change. Operations and developers need to become of one mind and respect each other. An automated infrastructure is the one tool you need most. Common source control. One step build. One step deploy. Don't be a pussy, deploy. Always ship trunk. Feature flags - don't branch code, make features runtime configurable in code. Dark launch - release data paths early without UI component. Shared metrics. Adaptive feedback to prioritize important features. IRC for communication for human context. Best solutions occur when dev and op work together and trust each other. Trust is earned by helping each other solve their problems. Look at what new features imply for operations, what can go wrong, and how to recover. Provide knobs and levers to help operations. Devs should have access to production machines. Fire drills to train. No finger pointing - fix stuff first. Design like you'll get woken up first when there's a problem. Say you're sorry. Not easy - like any relationship.
Update: Operational Efficiency Hacks Web20 Expo2009 by John Allspaw. 131 picture perfect slides on operations porn. If you're interested in that kind of thing.

Dream with me a little bit. Your startup becomes wildly successful. Hard work and random chance have smiled on you. To keep flirting with lady

luck your system must scale. But how much stuff (space, hardware, software, etc) will you need to handle the growth, when will you need it and when will you need more?

That's what Flickr's John Allspaw helps you figure out in his ground breaking new book on capacity planning: The Art of Capacity Planning: Scaling Web Resources.

When I read statements about *The Art of Capacity Planning* like *capacity planning is a term that to me means paying attention*, *All the information you need to make an educated forecast is in your historical metrics*, and *startups that are going to experience massive growth simply don't have time for anything but a 'steering by your wake' approach*, I get the same sea change feeling I felt when the industry ran from waterfall design and embraced agile design. Current capacity planning is heavy. All up-front. Too analytical and too divorced from real life.

Other capacity planning books assault you with models, math, and simulations. Who has the time? John has developed a common sense, low math approach to capacity planning that works using the system you already have. John's goal is to have you say: *Oh, right, duh. That's common sense, not voodoo.*

Here's my email interview with John Allspaw on *The Art of Capacity Planning*. Enjoy.

Please tell us who you are and what you've brought to show and tell today?

I'm John Allspaw. I manage the Operations team at Flickr.com, and I've written a book (The Art of Capacity Planning: Scaling Web Resources) about capacity planning for growing websites.

After spending a good chunk of your life writing this book, can you summarize it in just a few sentences so people will know why it should matter to them?

This book is basically a guide to adaptive capacity planning for growing websites. It's an approach that relies much less on benchmarking and simulation, than on the close observation of production loads to guide future decisions. It's not rocket science, and I'm hoping people can use it to justify the what, why, and when of getting more resources to allow them to grow as fast as they need to. It's worked really well for me at Flickr and other organizations.

Give me your ripple of evil. What happens without capacity planning?

Capacity planning is a term that to me means paying attention. Web applications can fail in all sorts of dramatic ways, and you're not going to foresee all of them. What you can do, however, is make use of what you do know about what happens in your real world on a regular basis. Things like: my database can do X queries per second before it keels over. Or my cache can only keep Y minutes worth of changing objects. You're not going to predict every failure mode of the whole system, but knowing the failure modes of individual pieces should be considered mandatory. Armed with that, you can make decent forecasts about the future.

I'm a guy or gal at a startup who's freaking out because my boss asked me how much hardware we need for the next quarter/ year? What do I do now?

Buy my book? • All the information you need to make an educated forecast is in your historical metrics. You do have system and application-level statistics, right? Tying your system level stats (CPU, memory, network, storage, etc.) to application-level metrics (users, posts, photos, videos, page views, widgets sold, etc.) is key, because then you have history to back up the guesstimates. Your business, product or marketing team also has their own guesses for some of those application-level metrics, so you should get the two forecasts together and see where/how they match up or differ. Capacity should enable the business, not hinder it.

So your book, is it the best book on Capacity Planning or the greatest book on Capacity Planning?

My book is the best book on capacity planning. If it were the 'greatest' book, it'd be a lot bigger than it is. It's good for scaling your hardware. It's not good for flattening out posters.

My site is gonna explode and I don't know at what point it's going to die. What do I do now?

Argh! Panic! Handle the current explosion, and make it priority one to find out the limits of your capacity when the emergency is over.

Try to panic gracefully. If it's dying right now, and you can't easily add any more capacity, then try some of the tried and trusted WebOps 101 tricks mentioned everywhere, including this blog:
- disable features (preferably the heavier load-causing ones)
- cache previously dynamic content into static bits
- avoid loaded backend calls by serving stale content

Of course it's easier to do those things when you have easy config flags to

turn things on or off, and a list to run through of what things are acceptable to serve stale and static. We currently have about 195 'features' we can turn off at Flickr in dire circumstances. And we've used those flags when we needed to.

Having said all of that, knowing when your resources are going to die should be mandatory, not optional. Know how many qps your databases, webservers, caching systems, and storage can handle before degradation. Test that stuff. With production traffic. If at all possible, with live production traffic, not just recorded and replayed production loads.

How do you compare your approach with well known approaches like Guerrilla Capacity Planning? Isn't focusing on queuing theory, Little's Law, and Possion arrival rates misguided? What will really help people on-the ground?

My approach is a bit different from Mr. Gunther's, although his book was an inspiration for mine. I do think that having a general understanding of queuing theory and the mathematics of open and closed systems can be important, don't get me wrong. But many of the startups that are going to experience massive growth simply don't have time for anything but a 'steering by your wake' approach, and done right, I think that approach will serve them well. I think some people recognize this, but in my experience, I'd say the development timelines are even tighter than most people realize.

Almost any time and effort in constructing and running a simulation, model, or benchmark that involves the main moving parts of a web site's back-end is pretty much wasted due to how quickly application logic, use cases, and even hardware configurations can change. For example, by the

time I could construct a useful model to capture the webserver-database interactions for Flickr, the results won't resemble production load, since the development cycle we have is so tight. As I write this, in the last week there were 50 code deploys of 550 changes by 19 Flickr staff. I realize that's a lot more than a lot of organizations, but that rate of change requires that the capacity planning process is easily adjustable.

I also found the existing books on the topic to be pretty dry with the math. The foundations of queuing theory are interesting, but proofs of Little's Law isn't going to quickly justify to my finance guy that we need 11 more webservers.

I have a cloud, do I really need to capacity plan anymore? That's so old school. Can't I decrease my administrator-to-server ratio and reduce the cost of managing systems by removing capacity planning resources?

Nope, just trust The Cloud™ that everything will be all right. No need to pay any attention at all. Oh wait, that's a magic unicorn talking. Cloud services are a resource just like any in-house capacity: they have limits that should be paid attention to. The best use cases of cloud computing both recognize the benefits (shrinking 'procurement' times, for example) and the limitations. And those limitations are going to vary from application to application, organization to organization.

You can build a real brand around a name like "Guerrilla Capacity Planning." Don't you think you could have come up with a better name for your approach?

Yeah, I guess I should have a name for it. How about "Paying Attention

Capacity Planning"?

You recommend testing the limits of your hardware and software on a production site. Are you crazy?

It's very possible that I'm crazy. But, using production traffic to define your resources ceilings in a controlled setting allows you to see firsthand what would happen when you run out of capacity in a particular resource. Of course I'm not suggesting that you run your site into the ground, but better to know what your real (not simulated) loads are while you're watching, than find out the hard way. In addition, a lot of unexpected systemic things can happen when load increases in a particular cluster or resource, and playing "find the butterfly effect" is a worthwhile exercise.

Where does capacity planning sit in the stack? It seems to impact the entire application stack, but capacity planning is more of an operations thing and that may not have much impact in engineering.

Capacity planning is a process, and should sit in the stack along with the other things that happen on a regular basis, like bug scrubs, like weekly meetings, etc. I'm spoiled as an Ops manager because we've got developers at Flickr who very much think like operations people. We're all addicted to graphs, and we're all pretty aware of how close each piece of the infrastructure is to becoming too 'hot', and we act accordingly. Planning, procurement, and measurement of capacity might lie on operations' shoulders, but intelligent use of it is everyone's business. I'm also blessed to have product management and customer care teams who are also aware of the state of capacity. As projects pop up that warrant capacity to be part of the considerations, it is. You simply can't launch

things like FlickrVideo and the Yahoo Photos migration without capacity being part of the requirements, so we've got a good feedback loop going with respect to operations and capacity.

Studies say 4/5 of people like a trip to the dentist more than they like capacity planning. Why does capacity planning have such a bad rep?

Because no one wants to guess and be wrong? Because most of the cap planning literature out there are filled with boring math?

Does the move to Web 2.0 make traditional approaches to capacity planning more difficult?

I would say yes, but of course I'm biased. In many ways, use of the site is simply out of our hands. We gather metrics on as many aspects of the site as we can get our hands on, and add more all the time. Having an open API makes things interesting, because the use cases can vary wildly. Out of nowhere, a simple application can reveal an edge case that we hadn't foreseen, so we might have to adjust forecasts quickly. As to it being more difficult: I don't know. When I'm wrong, people can't see photos. If the guy at wellsfargo.com is wrong, then people can't get their money. What's more annoying? :)

I really like your dashboard. Most dashboards say what happened, yours has an estimated days left before capacity runs out, which makes it actionable. How accurate have those numbers been?

Well that particular dashboard is still being built here, but the general design is to capture those metrics on a regular basis, and to use it as a guide. The numbers for some clusters have been pretty trustworthy. But they do fluctuate in how accurate they are, since "days left" are obviously affected by things outside of the forecasting process. Sometimes, a new feature is planned that requires pounding the database shards, or bumps CPU on the webservers by a noticeable amount. It's because of these things that the dashboard is only a piece of the puzzle. Awareness and communications with product management and development have to inform planning decisions, as well as the historic metrics. Again: no crystal balls here.

One of the equations you use is a "UR LIMITZ = Ceiling * Factor of Safety". Theo Schlossnagle has noted the evolution of phenomenal spikes in traffic, even for large sites. How do you have enough capacity as a safety factor if traffic is so spikey? What are the implications for forecasting?

Start with the assumption that no one can accurately predict the future. Capacity planning isn't the only part of successful web operations, and not everything is a capacity issue. Theo nails what happens in the real-world, for sure. The answer is: you estimate the best you can with the history that you have. Obviously, one mistake is to make forecasts ignoring the spikes you've experienced in the past. Something important to consider is how expected (or not) your spikes are. If you sell things, you might have a seasonal holiday spike or plateau in traffic. If you're a content site that frequently gains traction with news-related sites (like Theo's example) and from time to time, you get massive unexpected spikes, then your factor of safety should include considerations for those spikes. But of course the flipside of that might mean that you have a boatload of servers doing

nothing except wasting money while waiting for massive spikes that may or may not come. So it's a balance. Be reasonable with planning, follow the four guidelines that Theo points out in his post, and you've got yourself a strategy to deal with those unexpected spikes.

At what point does it make sense for me to buy my own equipment?

It's a difficult question. I've seen groups go from self-run colocation to managed hosting and cloud services, and others go the opposite way, for all legit reasons. Just as there are limitations with managed hosting, those limitations might not matter until you're massive. I do believe that there is a point at which owning your own equipment makes a lot of sense, because you've blown past the average needs of a managed hosting or cloud customer. Your TCO might be a lot different than mine, so to state that there's a single point for everyone would just be dumb.

Quick fire round. What's your quick reaction to:

1. Let's just go with a working prototype for now. We can change it when we grow big.

Fine, for some definitions of "working", "change", and "big". I can't complain too much about that approach in some sense because that's how a lot of Flickr's backend evolved. But on the other hand, all you need is to fail a couple of times with prototypes to figure out what sort of homework needs to be done before launching something that is potentially explosive. So again, there's a balance. Don't be lazy, but don't be rigid and too fearful.

2. Our VC told us that we're worrying about scalability too early. They doesn't want us to

blow our scarce resources on preparing for success.

Your VC is smart. If they're really smart, they'll also suggest worrying about scalability before it's too late. This answer isn't a cop-out, it's just reality. Realize that being scalable means being able to easily add capacity wherever you need it, whenever you need it. Buying too much equipment too soon is just as insane as hiring people that sit around and do nothing. The trick is knowing what "too much" and "too soon" is, and it's going to differ from company to company, from product to product.

3. Premature optimization is the root of all evil.

I've not made it a secret that I think this quote has given many an engineer (both dev and ops) reasoning to make dumb decisions. I actually agree with the idea behind the quote, but I also think that it's another one of those balancing acts. Knuth (or Hoare) also said "We should forget about small efficiencies, say about 97% of the time…" Another good question might be: which 3% are you going to pay attention to?

4. We plan to throw hardware at the problem.

Ok. When does that hardware need to come, how much of it, and what will you do when you realize more hardware won't fix a particular problem? The now classic limitations of the single-write-master, many-read-slaves database architecture is a great example of this. Experienced devs and ops people consider the possibility that hardware can't solve all problems.

Note: these questions were adapted from How Important Is Being Scalable?.

Now that you have some free time again,

what are you going to do with your life?

Eat. Sleep. Pay attention to my family. •

Related Articles

- Capacity Management for Web Operations by John Allspaw
