Capacity Planning

DAVID HIXSON AND KAVITA GULIANI

David Hixson is a technical project manager in Site Reliability Engineering at Google, where he has been for eight years. He currently spends his time predicting how social products at Google will grow and trying to make the reality better than the plan. He previously worked as a system administrator on high availability systems and has an MBA from Arizona State. https://plus.google.com/+DavidHixson, dhixson@google.com

Kavita Guliani is a technical writer for Technical Infrastructure and Site Reliability Engineering at Google, Mountain View. Before working at Google, Kavita worked for companies like Symantec, Cisco, and Lam Research Corporation. She holds a degree in English from Delhi University and studied technical writing at San Jose State University. kguliani@google.com

Capacity planning can be a torturous exercise in spreadsheets and meetings that drains the life out of junior engineers, provides little value to the company, leads to ongoing recriminations for everyone involved, and results in little planning or capacity.

Alternatively, it can be used to hone the understanding of the core services being offered, to work across the company to understand risks, and to make thoughtful choices for the business.

Let's focus on this second approach and talk about how three different parts of the company can play an active role in capacity planning. The more the players understand their part in the greater scheme, the better they can communicate and make tradeoffs that benefit the company.

Figure 1 represents three different perspectives. They may all be the same person thinking about the problem differently or they might be three vast organizations that rarely manage to get people into the same room. Even within a company, some products might need to be evaluated at different levels based upon their potential impact. The thought process is much more important than the job title associated with it.

What Is Capacity?

Before we get too far, let us define what capacity is. Very simply, capacity consists of the resources required to run your service or services in the context you have chosen to run them. The very core of this may be subject to debate or change, but the key things to predict are the resources you are constrained by. Depending on your scale or architecture, this could be gigs of RAM on a machine in your bedroom, cloud VMs, physical computers at your colo, bandwidth on a CDN, or megawatts of power.

As you increase in scale and complexity, you probably start to experience pressure in several dimensions, perhaps network capacity as well as storage or compute. And you may have different scaling limitations based upon your choices, either in terms of flexibility or timing around growth.

Engineering

Traditionally, the engineering organization would own the technical complexity and have the best understanding of how the system works right now, what choices were made to get here, and what things might be done in the future.

Depending on the organization, this might require cooperation across different teams, but the starting premise is that someone knows how things work and the resources required for the system to function today. This is far from trivial, but definitely a starting point when planning for the future.

Bottom-Up Capacity

The first step is to map out current system capacity. What resources does it use in order to get the work done? Identify all the large parts of the system: things that are material to your capacity-planning needs. Materiality will depend upon your organization, but there is a huge

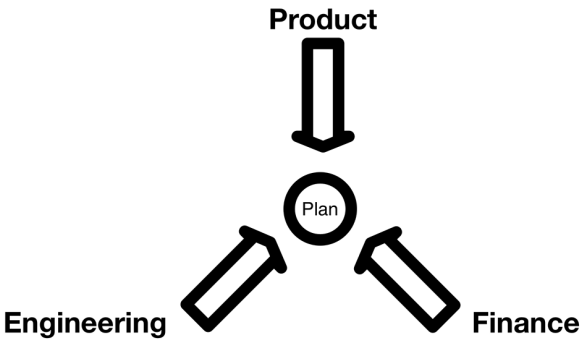

Figure 1: Three forces that impact capacity planning

value in simplicity, so if you can reduce the number of things you need to worry about, it will make everyone's life easier.

So, here we assume that you know how much you are using today in the resource dimensions that make sense for your service.

PRIMARY DRIVERS

The next step is to figure out why you are using the resources you are using today and what would make those numbers change. Metrics like "gigs of data uploaded by users today" are likely to impact your storage and bandwidth directly. Web queries per second (QPS) are likely to impact compute and, possibly, networking. Finding the fewest number of drivers (QPS, gigs uploaded, etc.) that capture the vast majority of the demand on your system is a challenge that should be reasonably thought-provoking.

If you have different types of queries or page views, you may want to look at a "costed" metric as a way to normalize the work and make it easier to understand. For example, a database read may be extremely cheap, but a write may be very costly in terms of CPU consumed and disk I/O required. So if you measure your database just in terms of QPS, you may make poor assumptions about scalability. If you can assign different "costs" to different actions, you can normalize them and make them better predictors of your ability to scale. Ideally, this is automated and constantly recalculated. However, even a gross estimate is useful. You may also realize that something like "bandwidth" is a better estimator than QPS and want to use that instead. Understanding these "costed" metrics can provide value in explaining the system as well as predicting future usage.

Popular metrics for product reporting include 30-day active or seven-day active users, which almost definitely do not determine the resources required to support your product. Similarly, you need to aggregate the data finely enough to be able to provision your system for peaks in demand. A queries-per-day metric is unlikely to be helpful here. Instead, you want QPS over a short interval (~minutes, hopefully) so that you can identify the peaks and be prepared to survive them.

Once you uncover the likely drivers for growth within the system, start collecting data. You will need to understand how these change over time as well as how the system load changes. The combination of these is the key to your bottom-up plan.

THEORETICAL MINIMUM CAPACITY

Finding correlations (and, hopefully, causality) between your identified growth metrics and observed capacity is a bit of a holy grail. For a complex system, it can be surprisingly difficult to get the two of them to line up nicely. You may never get perfection, but a thought model around "theoretical minimum blow-up" is a good way to start looking at it, and it has a nice side effect that we will get to in a minute. The "blow-up" that we are looking for is the inflation of either data or work that is inherent in the design.

Step back from your measurements and drivers and think about what your system is really trying to accomplish. The simplest example might be backing up bytes for users. If a user gives you a byte of data and you promise to give it back upon request, you've got a really clear understanding of the product. At no point will you need to store less than that byte (probably).

So how many bytes of disk do you use to store that byte?

1 for the original byte * 1.2 for RAID5 * 1.3 for "overhead" (file system, metadata, caching, backups, operational slack), then * 2 the whole mess for our second site. So we used a total of 3.12 bytes to store the byte that user gave us.

The same kind of thing can be done for the CPU required to update your database. How many replicas, stored procedures, and other things have to happen? You almost invariably do a lot of work many times over in order to make a write. Reads probably have a different set of factors.

With this kind of model in mind, you can go down two very interesting paths:

- You might tie together your capacity drivers and observed growth in a more natural way. Using the disk example, it might be better to explain why you grow disk capacity 3x as fast as users are uploading bytes.
- You could identify a bunch of questions around how you engineered your product. If you now seek to drive all of your multiples to 1, you will go out of business shortly because it isn't about optimizing without thinking. Instead, you need to evaluate each of them and see whether they are doing what you intend. Is that 6-disk RAID configuration the one you wanted for availability? Do you need that second site, or should you have more than two? Should you be doubling your investment in caching, or is it no longer providing the value you expected?

You can also use this blow-up model to look at the engineering changes that you have planned for the future, and account for them more clearly. A wise engineer will also check in regularly to make sure that these blow-up factors and assumptions remain accurate. It may help you spot when your system is drifting away from how it was intended to operate.

Although it is possible to make this kind of model as complicated as you have time to work on it, the key thing is to pull out the large drivers of relevant capacity and to highlight the engineering tradeoffs. You get the largest benefit if you ignore all the little things, letting everyone involved focus their attention on the choices that matter.

PAST PREDICTS FUTURE

The best starting point for predicting the future is observing the past. It is far from perfect, but the alternatives all involve significantly more made-up numbers.

Using the theoretical minimum blow-up factors, extract out your historical growth, then project it into the future. Is it a curve, is it a line, is it some complicated pattern too deep for the human mind to comprehend? Maybe. But in the vast majority of cases, you can assume that it is a line that extends out since the last time you made a significant product change.

Growth is frequently broken into two categories: organic growth and inorganic growth.

Organic growth comes from the natural adoption and usage of your product by customers. It may change over time, but it should change comparatively gradually and not as a step function. Examples of organic growth in an image-serving system might include uploading more photos, resulting in more bytes that need to be stored, and having more people view the photos resulting in more network load and larger serving capacity.

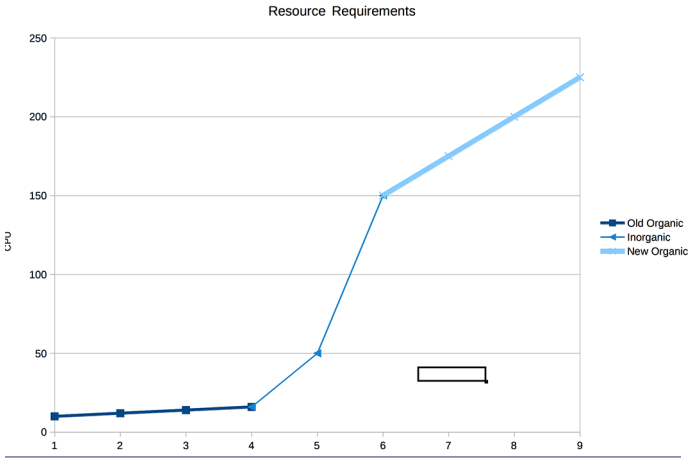

Figure 2: Examples of organic and inorganic growth over time

Inorganic growth refers to those step changes, likely the result of a feature launch, a marketing campaign, or business-driven change to how your product is being used (e.g., acquired another company and redirected traffic, bundled with another product, changed pricing, etc.)

Then you can layer on expectations about future inorganic changes. Perhaps it is the next advertising campaign or an upcoming change in the business strategy, or just a change of the background color for your app to a really soothing shade of teal. Estimate the impact and layer that into your plans, but keep them itemized because that gives you room to learn as you repeat the process and a place to begin discussions with the product and finance representatives. You should also consider how quickly you start to treat past launches as part of the organic line. As you start to observe the actual behavior, you will have a more accurate understanding of the impact on your service than before you made the change.

Assumptions

The final responsibility of engineering in the capacity-planning process is to highlight the design assumptions and any risks that are being taken with the product. It is very difficult to ship a product. It is even more difficult to get it out the door if you don't have any fundamental assumptions about how it will be used.

Did you build in caching at any level? Does it assume that traffic will be distributed in a certain way? Does the system break down if your assumptions become incorrect? Can you survive with 90% of your traffic going to a single Web page? What if it is evenly distributed over your entire corpus?

Did you assume a readers-to-writers ratio for your system? What if it suddenly flips and all of your traffic surges towards the more expensive of the two?

Did you assume that load would be distributed evenly over the day? What if you get a traffic spike from a media event or successful advertising?

One great thing about these assumptions is that clearly stating answers to them provides a warning for your product managers. However, if we take actions to invalidate these assumptions, we first need to do the engineering to survive it or consciously accept the risk it entails.

Risks

Through this analysis, we have identified a couple areas of risk that should be mitigated or accepted as part of the planning process. Any risk that is mitigated by capacity planning is an explicit tradeoff against money (or alternate uses for the resources aka opportunity cost). So the goal should never be to eliminate or even reduce risk. The goal should be to drive the system to the appropriate level of risk for the lowest cost.

Each of the theoretical minimum blow-up factors makes for a good place to start itemizing the places where you are spending money to avoid risk. What does that second datacenter buy you? How about your disk configuration or even the vendor for your hardware or cloud service? What problem are you avoiding, and how much are you avoiding it by? These problems and costs will be extremely specific to the service and might involve reliability, durability, performance, or even developer velocity. Or they could easily include all of them.

The other area of risk mitigation is around future growth: the thing that we are traditionally trying to estimate in capacity planning. You should plan at least as far ahead as the order time for your resources. This could be five years if you erect your own buildings, or 10 minutes if you run a small service in a cloud and have good automation and a credit card on file. Within that horizon, you must understand the risk of spending too much money, or of running out of capacity to handle your growth. That tolerance should define how aggressively you provision your service for growth.

PLAN B

Where would you be without a solid backup plan? Living in fear of a "success disaster" and failing to get a good night's sleep. Success disaster can occur when your product becomes so popular that the number of users who show up overwhelms your ability to actually provide the service they seek. It is great to be wanted, but this can squander your one opportunity to make a good impression.

As part of this process, always take time to understand what would happen when your demand for capacity exceeds what is available. What if the service fails and there is no way to save it? You might want to highlight that clearly in the process. On the other hand, if you can come up with some ideas for either graceful failure modes or improvements to efficiency and estimate the time required to implement them, then you can sleep more soundly. You just need to be able to build and deploy those solutions quickly enough to help. I'd suggest setting some alert thresholds to signal when you really should start to panic.

Product

The job of the product managers in the capacity-planning process is simple: Create a product that users love so much that it completely obliterates the planning and, after a brief period of panic, brings tears of joy to the finance team and the rest of the company. No pressure.

But that isn't the part of the job we are focusing on right now. This is about communication of the practical costs and risks that the engineers either have built or plan to build into the product, the needs of the users, and making sure those are aligned.

Alignment with Engineering

PROVIDE INFORMATION TO ENGINEERING

Start with the information you can provide to engineering. The big items are growth estimates, user behavior changes, and real product requirements. Attempt to understand how users will use the service in the future, with as much lead time as your capacity planning requires and in the growth dimensions used by the engineering model. These estimates will form the basis for future growth if you want to predict anything more complicated than an extrapolation from the past. You can pull the numbers out of thin air or dive deeply into the metrics of similar products, or survey your customers—whatever will provide you with numbers you can confidently use to drive planning far enough into the future that it makes a difference.

Second, you should help identify changes in user behavior, particularly when they conflict with any assumptions that have been made in the product design. Did you plan on building a system for sharing photos publicly, but people are using it to back up their receipts and keeping everything private? If so, you probably want to rethink that caching strategy. These kinds of changes can be critical to the success of your product but also need to be accounted for in the planning.

Third, product requirements should come from someone representing the customer. Is availability critical? Two nines or five? How fast does the system need to be? And if your answer is "all the nines" and "instant," then you probably need to rethink how you see the product. The goal should be to identify at least a minimum level of quality (i.e., the minimum level before it slows adoption), or better yet, a range of requirements that can be tied to how customers will feel about the product. For many products, it is possible to run experiments, making changes to the performance of the system, and observing the behavior of users in order to get firm numbers around what the real requirements are. For example, you can increase latency artificially or decrease it by moving the user to a less loaded copy of your infrastructure and measure differences in how they use the product.

An example would be latency requirements and establishing their impact on customers. We should understand how users perceive our service based on different levels of responsiveness. This might let us learn that anything faster than 250 ms is indistinguishable to the user and that anything slower than 750 ms conveys a sense of low product quality. This would let us target between 250 and 750 ms as an ideal range for our planning.

You can use this kind of information to drive engineering and finance decisions, potentially making your product much less expensive to operate or much easier to develop and deploy. The earlier you can create and refine these numbers and feed them into the design process, the more potential you have to build the product you need at the lowest cost.

RECEIVE INFORMATION FROM ENGINEERING

The engineering assumptions about the product should provide extremely valuable information about how it is being designed and deployed. In a negative sense, it should highlight the parts of the product that are either expensive or particularly risky if the consumer behavior changes dramatically. So it is important to keep these in mind either while marketing or while designing upcoming features because these are likely to increase the risk in the system.

On a more positive note, you may discover functionality that is particularly inexpensive or trivial to implement in the product, and this may help you come up with features for the future. Or you might gain a better understanding of how your competitors may have designed their infrastructure, letting you focus on features that will be difficult for them to quickly emulate.

Finance

"Finance" is shorthand for people responsible for keeping the business funded and growing. At the end of the day, this is everyone's responsibility, but most companies have people that focus on these types of things to the exclusion of developing new products or talking with users. Most importantly, this is the role that **looks across the entire company** and not just your product.

Asking Hard Questions

So the defining characteristic of Finance is actually that of scope, and with that comes the ability to make tradeoffs across multiple products and across time.

Working through these questions in a small well-funded company with a single product is a difficult task. As the size and complexity of the products offered by the company increase, doing holistic capacity planning becomes increasingly difficult. The goal should be to gain the advantages of scale and risk pooling to offset this increased challenge.

TIMELINES

Start by filling out a small table (Table 1). We can assume that we have a good resource model in place and this has been done before, but invite the engineering and product people to help generate these numbers.

Sum up those dates, and if any part of this can only be done at specific times of the month, quarter, or year, add that in as well. Specifically, if your organization has budgets that are only flexible at quarterly or annual boundaries, it leads to a substantial increase in your lead time. This lead time is critical for anyone doing capacity-planning to be familiar with.

One of the most valuable things that a company can do to drive down capacity-planning risk and cost is to shorten that cycle. So over time, each step should be evaluated to see whether it provides value to offset the cost of the additional delay.

Lead Time	Topic	Description
	Generate planning numbers	Create the numbers that drive the process
	Estimate resources	Turn growth estimates into specific resource requests
	Request resources	Ask for budget and equipment
	Approve resources	Complete budget and ordering process
	Provision resources	Deliver and set up
	Ready to serve	Provide service to users

Table 1: Timeline for resource delivery

CONFIDENCE AND PRECISION

For each growth estimate that you receive from each product, you need to understand more than just the bottom-line number around the resources required. The first thing to do is ignore the precision. Precise numbers are easy to generate since they just require the multiplication of two or three made-up numbers and very little rounding, but they trick almost everyone into thinking that they are "better" than a person who just writes down a 10 and moves on with their life.

Dig deeper. Check out the confidence associated with both the capacity model as well as the growth predictions that went into turning that model into a future-growth forecast. Check out the sensitivity that those estimates have to their time horizon and how far into the future people are being asked to forecast. There is a very natural tendency to overstate the potential upside of a launch. The people involved are likely very excited about the changes, and that may make it difficult for them to remain objective. Challenge these assumptions and make sure they aren't unnecessarily keeping you from spending resources seeking out other opportunities.

Finally, look across the products and see whether they appear to be correlated with each other in terms of growth and cost. You may be able to collapse the "upside" of several products together and plan on having only some of them succeed. This is a very specific way to trade increased risk for the organization against lowered cost. It assumes that the resources you are under-planning are fungible across products and that someone is in a position to resolve conflicts if your demand outstrips your resources because of this choice.

Alternatively, there may be two forms of synergy that make this particularly dangerous. The first is technical coupling, where the success of one product forces work on the other products, so they aren't actually independent. The second is that if one product is successful and is able to pull along the other products

indirectly, then growth may be correlated, again increasing the risk of a bundled approach to planning. Brand recognition, news coverage, cross-promotion between products, or any other ideas you have can tie together the growth rates of various products. In cases where you desire more traffic, these are great problems to have, but you must consider the potential in your planning.

PORTFOLIO RISK MANAGEMENT

The risks of each product and their growth scenarios must be understood in the context of the larger portfolio of the company. These risks come in two general forms: the specific product risks and the organizational risks.

The specific product risks should largely be as explained by the engineers. What happens if growth exceeds the capacity that is planned for the service? Do we have legal liabilities or customer dissatisfaction that will be particularly harmful to the company? Options may be available to mitigate these risks if it is not possible to provide the required resources, but they should be fairly explicit.

The second class is more difficult. Identify the risks that cross product boundaries and that may be less obvious to the specific product teams. If you have a company with multiple products that have dependencies on each other, this is where you need to look for those and highlight them specifically. Make sure that failure (or success) of one product doesn't do anything surprising and harmful to other products. This is when help from leaders within engineering will be very helpful to identify linkages and make them explicit. It may provide a sort of transitive priority or mutual dependencies between different products that need to be evaluated.

SUPPLY CHAIN AND LOGISTICS

Very specific to the table at the beginning of this section, it is important to understand what can be done to drive down the time required to go from having the desire to fund a product to having the ability to make that product functional with the resources in hand. With a cloud-provisioning model and a small-scale relative to your cloud provider, this may be trivial. But as your resource requirements increase in size or complexity, this may be about shaving months or even years off the system.

The rule of thumb here should be that **if you know what you are going to do in the future, you probably shouldn't get hung up on the paperwork.** This is much more difficult than it sounds and will likely present a challenge for any company that tries to implement it, but the goal is simple: shrink the time horizon between taking estimates and providing capacity. If the resources used by different products are fungible, you can pool them and manage their provisioning much more quickly than their full lead time.

Alternatively, if you spend the money before the customer arrives and you have reasonable fungibility between resources, you may be able to greatly reduce the time from request to provisioning, by ordering the resources in advance and scheduling the capacity plans to arrive in time for provisioning rather than for ordering the resources. An example of this would be building out datacenter space based on past growth trends for the company, but not deciding which product you would fund until right before the machines landed.

Fungibility is obviously very helpful: letting tradeoffs be delayed until the last moment, having resources shifted as necessary between different products, or keeping pooled resources available to manage risks on short notice.

RELIABILITY AND OTHER METRICS

The final valuable questions center around the metrics that each product is attempting to achieve. Understanding what these mean and the choices made by engineering to achieve them, as well as the value provided to the customer, is critical to providing the "best" experience possible at the lowest cost.

Reliability is the easiest example, since it is fairly straightforward to buy reliability with increasingly large piles of money as you request more "nines" of availability. However, in most cases, it actually has diminishing returns to your users. Take the time to dive into this for the big products, find commonalities around how you are reaching your targets, and look for the things that cost the most. The cost could be in buying Tier-4 vs. Tier-1 datacenter space, fault-tolerant hardware, licensed software solutions, or through engineering and operational complexity that slows down your rate of development. Don't underestimate the cost or value of having reliability designed into your software stack and your operational practices. It may be a much more effective investment than hardware. Having common solutions across the company and regular investigations into each of these choices can provide opportunities to improve products and save money at the same time.

Prioritization

The most difficult task that will come up at the company level is that of prioritization. In most organizations, it will almost certainly be impossible to fund the capacity requested of each product at its most optimistic growth rate without any improvements in efficiency. And over any reasonable period of time, it probably isn't a wise investment either. On the other hand, in case of small companies or startups, the cost of resources is probably small relative to other expenses. As a result, the limiting factors around prioritization won't be around capacity planning but engineering time or management attention.

What is important is having a full understanding of the risks of underfunding each product relative to its actual growth. This

risk is likely an "opportunity cost" in many cases as well as some more "real" costs in the places where customers are negatively impacted by the underfunded products. Prioritization, then, is about making clear choices between products so that they can operate with certainty, and doing it in a clear and timely manner. Timeliness is particularly critical in systems that have long cycle times since time spent in analysis is actually costly in terms of the accuracy of estimates and planning that fed into the process.

Conclusion

More than just drawing graphs of how services will grow in the future, capacity planning should ideally be a process that pulls together different parts of the organization to determine how resources should be allocated to maximize their benefit to the company. Out of this will flow improvements in engineering, product, and process in a virtuous cycle.