

- [Home](#)
- [About](#)
- [The Audiogalaxy Chronicles](#)
- [The Programmer's Toolbox](#)
- [Dev Diligence](#)
- [Java Programming Videos](#)

[« The Lawsuit, Shutdown, and Aftermath](#)
[Things That Are Important: Where Clauses »](#)

[« Older](#)

Loading

[Newer »](#)

» «

« »

Programmer's Toolbox Part 3: Consistent Hashing

Published on March 17, 2008 in [Toolbox](#). [45 Comments](#) Tags: [consistent hashing](#), [dynamo](#), [memcached](#), [partitioning](#).

Next up in the toolbox series is an idea so good it deserves an entire article all to itself: consistent hashing.

Let's say you're a hot startup and your database is starting to slow down. You decide to cache some results so that you can render web pages more quickly. If you want your cache to use multiple servers (scale horizontally, in the biz), you'll need some way of picking the right server for a particular key. If you only have 5 to 10 minutes allocated for this problem on your development schedule, you'll end up using what is known as the naïve solution: put your N server IPs in an array and pick one using `key % N`.

I kid, I kid — I know you don't have a development schedule. That's OK. You're a startup.

Anyway, this ultra simple solution has some nice characteristics and may be the right thing to do. But your first major problem with it is that as soon as you add a server and change N, most of your cache will become invalid. Your databases will wail and gnash their teeth as practically everything has to be pulled out of the DB and stuck back into the cache. If you've got a popular site, what this really means is that someone is going to have to wait until 3am to add servers because that is the only time you can handle having a busted cache. Poor Asia and Europe — always getting screwed by late night server administration.

You'll have a second problem if your cache is read-through or you have some sort of processing occurring alongside your cached data. What happens if one of your cache servers fails? Do you just fail the requests that should have used that server? Do you dynamically change N? In either case, I recommend you save the angriest tweets about

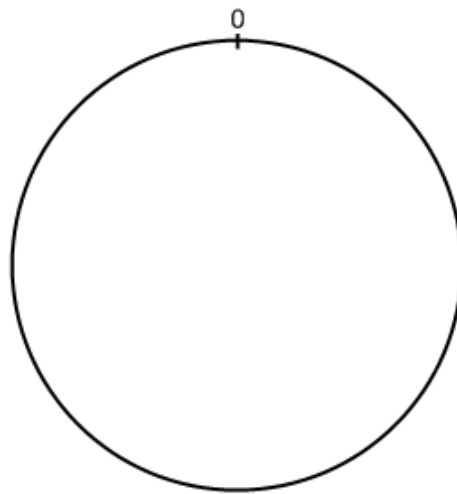
your site being down. One day you'll look back and laugh. One day.

As I said, though, that might be OK. You may be trying to crank this whole project out over the weekend and simply not have time for a better solution. That is how I wrote the caching layer for Audiogalaxy searches, and that turned out OK. The caching part, at least. But if I had known about it at the time, I would have started with a simple version of consistent hashing. It isn't that much more complicated to implement and it gives you a lot of flexibility down the road.

The technical aspects of consistent hashing have been well explained in other places, and you're crazy and negligent if you use this as your only reference. But, I'll try to do my best. Consistent hashing is a technique that lets you smoothly handle these problems:

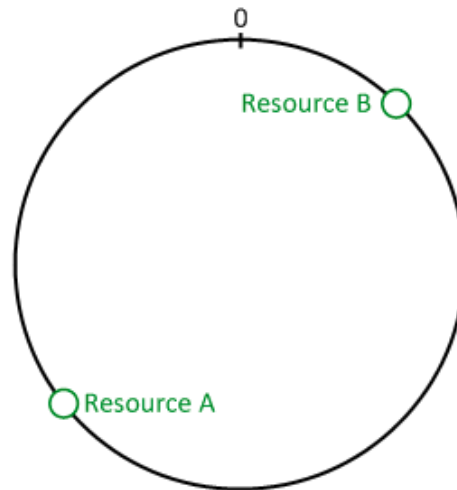
- Given a resource key and a list of servers, how do you find a primary, second, tertiary (and on down the line) server for the resource?
- If you have different size servers, how do you assign each of them an amount of work that corresponds to their capacity?
- How do you smoothly add capacity to the system without downtime? Specifically, this means solving two problems:
 - How do you avoid dumping 1/N of the total load on a new server as soon as you turn it on?
 - How do you avoid rehashing more existing keys than necessary?

In a nutshell, here is how it works. Imagine a 64-bit space. For bonus points, visualize it as a ring, or a clock face. Sure, this will make it more complicated when you try to explain it to your boss, but bear with me:

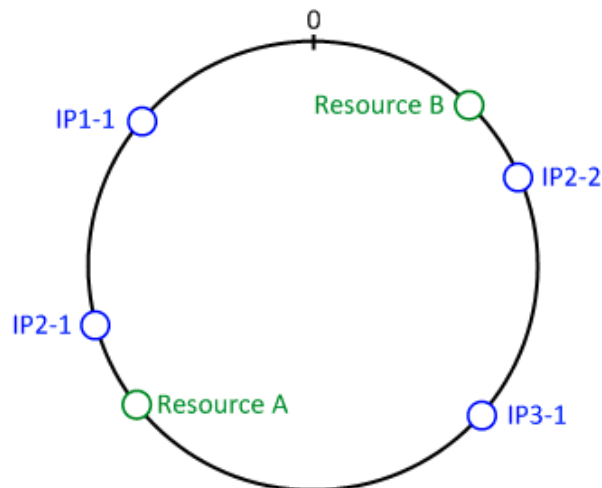


That part isn't very complicated.

Now imagine hashing resources into points on the circle. They could be URLs, GUIDs, integer IDs, or any arbitrary sequence of bytes. Just run them through a good hash function (eg, SHA1) and shave off everything but 8 bytes. Now, take those freshly minted 64-bit numbers and stick them onto the circle:



Finally, imagine your servers. Imagine that you take your first server and create a string by appending the number 1 to its IP. Let's call that string IP1-1. Next, imagine you have a second server that has twice as much memory as server 1. Start with server #2's IP, and create 2 strings from it by appending 1 for the first one and 2 for the second one. Call those strings IP2-1 and IP2-2. Finally, imagine you have a third server that is exactly the same as your first server, and create the string IP3-1. Now, take all those strings, hash them into 64-bit numbers, and stick them on the circle with your resources:



Can you see where this is headed? You have just solved the problem of which server to use for resource A. You start where resource A is and head clockwise on the ring until you hit a server. If that server is down, you go to the next one, and so on and so forth. In practice, you'll want to use more than 1 or 2 points for each server, but I'll leave those details as an exercise for you, dear reader.

Now, allow me to use bullet points to explain how cool this is:

- Assuming you've used a lot more than 1 point per server, when one server goes down, every other server will get a share of the new load. In the case above, imagine what happens when server #2 goes down. Resource A shifts to server #1, and resource B shifts to server #3 (Note that this won't help if all of your servers are already at 100% capacity. Call your VC and ask for more funding).
- You can tune the amount of load you send to each server based on that server's capacity. Imagine this spatially – more points for a server means it covers more of the ring and is more likely to get more resources assigned to it.

You could have a process try to tune this load dynamically, but be aware that you'll be stepping close to problems that control theory was built to solve. Control theory is more complicated than consistent hashing.

- If you store your server list in a database (2 columns: IP address and number of points), you can bring servers online slowly by gradually increasing the number of points they use. This is particularly important for services that are disk bound and need time for the kernel to fill up its caches. This is one way to deal with the datacenter variant of the [Thundering Herd Problem](#).

Here I go again with the control theory — you *could* do this automatically. But adding capacity usually happens so rarely that just having somebody sitting there watching top and running SQL updates is probably fine. Of course, EC2 changes everything, so maybe you'll be hitting the books after all.

- If you are really clever, when everything is running smoothly you can go ahead and pay the cost of storing items on both their primary and secondary cache servers. That way, when one server goes down, you've probably got a backup cache ready to go.

Pretty cool, eh?

I want to hammer on point #4 for a minute. If you are building a big system, you really need to consider what happens when machines fail. If the answer is “we crush the databases,” congratulations: you will get to observe a cascading failure. I love this stuff, so hearing about cascading failures makes me smile. But it won't have the same effect on your users.


Finally, you may not know this, but you use consistent hashing every time you put something in your cart at Amazon.com. Their [massively scalable data store, Dynamo](#), uses this technique. Or if you use Last.fm, you've used a great combination: consistent hashing + memcached. They were kind enough to [release their changes](#), so if you are using memcached, you can just use their code without dealing with these messy details. But keep in mind that there are more applications to this idea than just simple caching. Consistent hashing is a powerful idea for anyone building services that have to scale across a group of computers.

A few more links:


- [Another blog post about consistent hashing](#)
- [The original paper](#)

45 Responses to "Programmer's Toolbox Part 3: Consistent Hashing"

[Feed for this Entry](#) [Trackback Address](#)


-  [Peter T - webshop](#)
[March 17, 2008 at 8:59 pm](#)

Would sticky sessions enabled on a load balancer help with the caching issue "If you want your cache to use multiple servers (scale horizontally, in the biz), you'll need some way of picking the right server for a particular key."? Good article by the way...thanks.

-  [localhost](#)
[March 18, 2008 at 12:45 am](#)

Interesting and informative article, thanks!


- [Working Notes on Consistent Hashing - Laughing Meme](#)
Pingback on [Mar 19th, 2008 at 8:51 am](#)
- [Consistent Hashing at Gea-Suan Lin's BLOG](#)
Pingback on [Mar 21st, 2008 at 12:43 am](#)

-  [Al](#)
[March 23, 2008 at 3:07 am](#)


Peter,

Sticking a users session to a server, web, application or other, isn't going to help in determining what particular server within your cache cluster has the key you want.


Al.

- [Link Box « handthrow](#)
Pingback on [Mar 26th, 2008 at 6:51 am](#)
- [Photo Matt » Consistent Hashing](#)
Pingback on [Mar 30th, 2008 at 4:52 pm](#)
- [How To Split Randomly But Unevenly - PHP Code For Load UNBalancing \(Utopia Mechanicus\)](#)
Pingback on [Apr 3rd, 2008 at 1:52 am](#)
- [How To Share The Load \(Off-Topic\) \(ActiveBlogging\)](#)
Pingback on [Apr 3rd, 2008 at 4:11 am](#)
-  [David](#)
[April 7, 2008 at 11:24 am](#)

Thanks for sharing important logic like this that most people don't think of until they are in a big mess.


-  [Mike Zintel](#)
[April 15, 2008 at 9:34 pm](#)

MD5 isn't secure.

-  [Paul Annesley](#)
[April 21, 2008 at 5:20 pm](#)

Thanks for great the article Tom.

Your clear explanation and illustrations inspired me to write an open source implementation for PHP, as I couldn't see anything decent around that fit the bill. I've put it on Google Code at <http://code.google.com/p/flexihash/> in case anybody needs it.

-  [Steven Gravell](#)
[May 27, 2008 at 12:33 pm](#)

Oh, there's a java version in there too along with the libketama C library that is used for the PHP extension

- [Flexihash » Blog Archive » Flexihash - Consistent Hashing for PHP](#)
Pingback on [Jun 24th, 2008 at 6:17 am](#)
- [User Primary » Blog Archive » Friday Links](#)
Pingback on [Oct 3rd, 2008 at 8:06 am](#)
- [Live Mesh : Behind Live Mesh: The Pub-Sub System](#)
Pingback on [Oct 8th, 2008 at 7:34 pm](#)
- [Memcached and Mogile Form MemcacheMegaZord! | FewBar.com - Make it good](#)
Pingback on [Dec 14th, 2008 at 9:21 am](#)
- [-= Linkage 2007.02.18 =-](#)
Pingback on [Jan 26th, 2009 at 7:37 am](#)
- [Bookmarks for April 15th through April 16th • Passing Curiosity](#)
Pingback on [Apr 15th, 2009 at 10:23 pm](#)
- [Shane K Johnson » Blog Archive » How I learned to say 'No' to SQL](#)
Pingback on [Sep 30th, 2009 at 6:23 am](#)
- [Consistent Hashing?? | ??????????](#)
Pingback on [Oct 2nd, 2009 at 12:27 am](#)
- [Sean Neakums \(sneakums\) 's status on Sunday, 18-Oct-09 09:25:42 UTC - Identi.ca](#)
Pingback on [Oct 18th, 2009 at 1:25 am](#)
- [OCTO talks ! » Consistent Hashing ou l'art de distribuer les données](#)
Pingback on [Nov 6th, 2009 at 10:09 am](#)
- [Episode 2: A brief introduction to NoSQL databases - Hacker Medley](#)
Pingback on [Jan 20th, 2010 at 5:29 pm](#)
- [diego sevilla's weblog » Más de bases de datos NoSQL: Consistent Hashing](#)
Pingback on [Mar 1st, 2010 at 4:57 am](#)
-  [Ross](#)
[March 26, 2010 at 3:44 am](#)

Hej Tom,


I got half excited about your article... hits most of the squares... However your circle shows the servers nicely equadistant on the 360 but if we leave their placement to the hash function then they could actually all appear in a very accute part of the circle so that the services or resources could mostly be directed unneavenly to one server....

Would it not be best to divide the available resources equally in the pie? (and one could take into account weighting on this also)

If in the case that one server fails, then instead of applying roundrobin, would it not be best to select the best server... i.e. the server with the most currently available processing capability (obtained from server size and current load).

Your comments would be well appreciated.

Slainte //Ross

-  [Tom](#)
[June 7, 2010 at 3:49 pm](#)

@Ross — each server should be hashed into many points on the circle. That way probability takes care of spreading them out equally.

-  [Siddhartah](#)
[July 14, 2010 at 8:38 am](#)

Good Article Tom,

For newbies, if you include active-standby also in this article, then removal use-case will make more sense. I mean when one server has to be removed, what will happen to the data it contained. So either (a) the standby server is assigned the same IP (say using Virtual IP) or (b) a new server which was having the original data of the failed node will be added to the hashing and failed node IP will be removed.

Many thanks Tom

-  [Onkar](#)
[July 22, 2010 at 3:59 am](#)

Very informative article. Nicely written. Cool stuff.

Thanks.

- [Optimiser la répartition de charge avec le "Consistent Hashing"](#)
Pingback on [Oct 3rd, 2010 at 12:35 pm](#)
- [Consistent Hashing » ~magog/public](#)
Pingback on [Nov 13th, 2010 at 4:48 pm](#)

-  PJ
[January 20, 2012 at 12:57 am](#)

Fantastic article!

- [Ars: Storing Big Data // RLASKEY: words](#)
Pingback on [Jan 27th, 2012 at 10:49 am](#)

-  [luc1e](#)
[March 24, 2012 at 8:43 am](#)

Interesting, useful, well explained, and well written. Thanks 😊

-  Brent
[March 24, 2012 at 9:00 am](#)

Great article. I ran a few tests that started with 3 servers, each weighted with 3000-5000 points handling 2000 resources over several million random resource requests. Half way through the test, I dynamically added another, much more powerful server (50000 points). The hashing worked great. The requests were distributed according to weight over the first half of the test and weighted heavily on the last server for the second half of the test.

The only outlier is if you have, say, 3 resources instead of 2000. In that case, server 1 will never be used unless it has exponentially more weight than the other servers.

-  [John](#)
[March 24, 2012 at 9:08 am](#)

A coworker of mine blogged about how to expand on consistent hashing to include the concept of availability zones.

<http://www.tlohq.com/p/building-consistent-hashing-ring.html>

-  [Cameron Purdy](#)
[March 27, 2012 at 7:08 am](#)

Wow. Amazon really screwed the pooch with Dynamo by pretending that the “hash wheel” was a solution to anything (unless the problem that they were trying to solve was “How to create needlessly complex systems.”)

Now tech-lemmings everywhere are preaching this ludicrously bad design as the second coming of Microsoft Bob.

Look, if a design solves a problem poorly, and degrades drastically on just about every type of failure condition, then the design sucks. Logic alone is enough to cause this design to be discarded.

-  Greg Holt
[March 31, 2012 at 9:45 am](#)

Cameron, it'd be cool if you'd state your case with some pointers to facts, research, or something helpful. You've essentially just said "You're stupid." which isn't horribly useful to helping us become smarter.

-  Bob
[April 2, 2012 at 8:50 am](#)

Thanks for the great article. FYI there's Riak by Basho which implements this structure.

-  [zellux](#)
[April 3, 2012 at 4:27 am](#)

Thanks for the great article!


But I still do not understand how to make the caching server balanced. As shown in the third figure, when IP2-1 and IP2-2 are added, since their positions are based on hash values and thus randomized, is it possible that the second server will only occupy a small fraction of the whole interval, instead of 1/3?

-  Kevin
[April 6, 2012 at 4:27 pm](#)


@zellux

In reality, you would extend this idea by assigning thousands or more keys per server (IP2-1 ... IP2-5000, for instance). If you do this for every server and using a uniform hash function, you'll have balanced load distribution.

- [Partitioning Tables to Scale Writes | Brent Ozar PLF](#)
Pingback on [Apr 10th, 2012 at 5:00 am](#)
- [Round the Campfire - Issue 1, 2010](#)
Pingback on [Apr 13th, 2012 at 3:46 am](#)

-  Jason Huang
[April 24, 2012 at 3:43 pm](#)

God I love this article!

-  Yogi Sharma
[April 29, 2012 at 9:58 pm](#)

Really liked it. I had learned about the idea of consistent hashing a long time ago, and this article, especially the pictures, brought all the memories back, and in a few

minutes, I could freshen up my memory. Thanks for writing a good article.

Leave a Reply

Name (required)

Mail (will not be published) (required)

Website

Submit

[« The Lawsuit, Shutdown, and Aftermath Things That Are Important: Where Clauses »](#)

Recent Posts

- [In San Francisco Next Week](#)
- [Android Market Search Bug](#)
- [Introducing the Audiogalaxy API](#)
- [Where Are the AB Testing Frameworks?](#)
- [Two and a Half Months of Twitter](#)

Tags

[/dev/epoll](#) [Assertions](#) [Audiogalaxy](#) [Berkeley DB](#) [Bloom Filters](#) [consistent hashing](#) [Dev Diligence](#) [dynamo](#)
[FolderShare](#) [Google](#) [History](#) [ideas](#) [intolerance](#) [memcached](#) [Message Passing](#) [MS Manners](#) [MySQL](#)
[partitioning](#) [prizes](#) [productivity](#) [RIAA](#) [Skip Lists](#) [Soft Deletes](#) [Startup Lessons](#) [startups](#) [STS](#) [Toolbox](#) [Unrolled Linked Lists](#)
[uptime](#) [VEB Trees](#)

Powered by [WordPress](#) and [K2](#)

[Entries Feed](#) and [Comments Feed](#)

19 queries. 0.301 seconds.