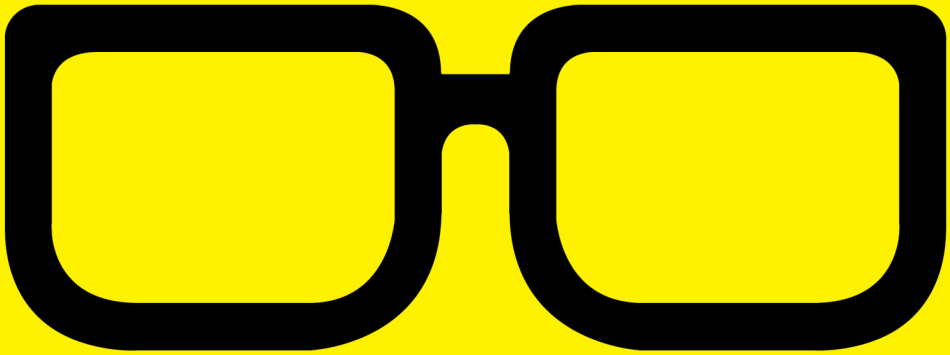
SPONSORED BY

IBM

GEEK GUIDE



Take Control of Growing Redis NoSQL Server Clusters

LINUX™ JOURNAL

Table of Contents

REUVEN M. LERNER is a Web developer, consultant, trainer and longtime columnist for *Linux Journal*. He recently completed his PhD in Learning Sciences from Northwestern University. You can read his blog, Twitter feed and newsletter at http://lerner.co.il. Reuven lives with his wife and three children in Modi'in, Israel.

GEEK GUIDES:
Mission-critical information for the most technical people on the planet.

About the Sponsor

IBM is a globally integrated technology and consulting company headquartered in Armonk, New York. With operations in more than 170 countries, IBM works to help solve problems and provide an edge for businesses, governments and non-profits.

Innovation is at the core of IBM's strategy. The company develops and sells software and systems hardware and a broad range of infrastructure, cloud and consulting services. As an example, IBM Power Systems are built with open technologies for mission-critical applications, offering servers designed for big data that are optimized, secure and adapt to changing business demands.

Today, IBM is focused on five growth initiatives: Cloud, Big Data and Analytics, Mobile, Social Business and Security. IBMers are working with customers around the world to apply the company's business consulting, technology and R&D expertise to enable systems of engagement that deliver dynamic insights for businesses and governments worldwide.

Take Control of Growing Redis NoSQL Server Clusters

REUVEN M. LERNER

Introduction

When I started to develop Web applications in the mid-1990s, I naïvely thought that if I needed to persist data across sessions, I could and should use a file. It quickly was explained to me that using the filesystem for such purposes was not a good idea, and that instead I should be using my server's relational database. Using a database made it possible for me to access the same data from a number of different servers. Moreover, using a relational database

I'm not the only one who is in love with Redis; it is the best possible combination of easy to use and very fast to execute, and it's full of functionality.

meant that I could store my data in an appropriate format, using a data type (for example, a number or a text string) that would express my intent more clearly.

Fast-forward 20 years, and you no longer have to convince developers that they should use a database. However, now the question isn't whether you should use a database, but rather what database technology you should use. The debate between SQL and what has become known as NoSQL continues to go on, with the SQL people pointing to established normals for ACID compliance, normalization and the many years of effort and knowledge that have gone into relational databases. NoSQL adherents claim that in the modern era of Web-scale applications, relational databases are passé, and that you should be using one of the flexible, schemaless, replicating databases.

I generally fall into the SQL category. I believe that relational databases work well, and that they're only getting better and more sophisticated. This doesn't mean NoSQL is always wrong, but I tend to be something of a NoSQL skeptic.

However, there is one major exception to my NoSQL skepticism, and that is Redis. I'm not the only one who is in love with Redis; it is the best possible combination of easy to

use and very fast to execute, and it's full of functionality. I've never come away from working with Redis disappointed.

How popular is Redis among businesses and developers? It's always hard to track the growth in popularity of an open-source product. Redis does appear to be growing in popularity, however, based on a few different metrics. The DB-Engines (http://db-engines.com) relative ranking of database popularity (by looking at mentions on Web sites, social networks and job boards) indicates that Redis is now at #10 in May 2015, up from #13 in May 2014. Redis Labs, a commercial supplier of Redis software, recently announced that it had 33% customer acquisition growth in the first quarter of 2015. Redis also is on GitHub, which provides some insights into the number of people interested in a project or contributing to it. At the time of this writing, 1,365 people follow the Redis project, and it has more than 170 contributors. By contrast, Memcached has 449 followers and 71 contributors.

How and why are businesses using Redis? In many cases, they're using it for the caching of user data, such as shopping carts or session information. But, Redis also can be used as a sophisticated counter for page views and social-network links. Additionally, it also can be used to identify trends in recent data, in case you're looking for fraud. You also can use it for publish-and-subscribe protocols, as a form of message queue that sits outside your application.

Redis is not the only key-value store, nor was it the first one to be released under an open-source license. For years, many developers used a similar system known as Memcached. However, as the name implies, Memcached

Redis is also amazingly fast. This has led to its use in many cases as a high-speed data cache, letting you store user profiles and shopping carts in something that's faster than a relational database.

stored data only in memory. If and when the Memcached server would fail or be rebooted, the contents would be lost. Redis, by contrast, regularly stores information to disk, such that even in the case of a power outage or other issue, little or no data actually will be lost.

In addition, Redis provides a rich set of data types, letting you store and manipulate not just text strings, but also lists of strings, sets of strings (in which each element is unique), sorted sets of strings and hash tables. If you want to use Redis to keep track of a user's current purchases, you can do that using a hash table, associated with the user's unique ID. If you want a set of the different IP addresses that have visited your server, or of the user IDs that have visited or even of the search terms that people have entered into your system, Redis' sets can handle those things with ease.

Redis is also amazingly fast. This has led to its use in many cases as a high-speed data cache, letting you store user profiles and shopping carts in something that's faster than a relational database. Several years ago, I worked on a project in the finance industry, where I needed to keep

track of currency prices. I used Redis to store the most recent values of the currencies we were tracking; this was a natural and easy mapping of the name-value pairs I needed to retrieve frequently from a source that was both reliable and faster than a database.

Redis is a client-server, key-value store. This means you can connect to it via a network protocol, and that it functions something like a hash table. Data is stored and accessed by a key, which must be unique. Thus, you can store the key-value pair:

```
a = 1
```

Or:

```
105 = reuven@lerner.co.il
```

This means that every key in Redis must be unique, but that's what you want in a key-value store. Redis does offer different databases, each of which is numbered, which you can specify when you connect to the server. However, you indicate the database to which you want to connect at connection time, not when you're reading or writing data.

That said, if you plan your key names a bit, you can ensure that there never will be collisions between the keys. For example, if your software is called "foobar", you might want to consider naming all of your keys starting with "foobar:". The colon (:) character is a legal part of a Redis key name, and it helps to create some degree of namespacing within a specific Redis instance.

Working with Redis

Most modern programming languages have Redis clients. A command-line client, known as redis-cli, lets you interact with Redis directly, rather than via another language. Within this interface, you use Redis commands, which are numerous but fairly simple to remember. For example, you can store the key-value pair a:b via redis-cli as:

```
set a b
```

You then can retrieve the value associated with "a" with:

```
get a
```

You even can modify the value with:

```
append a c
```

such that now, you see:

```
get a
```

returns

```
bc
```

You similarly can manipulate and work with lists, sets, sorted sets and hash tables. Each data type has its own commands. Those that have to do with lists generally begin with the letter L, while those having to do with sets begin

with S, hash tables begin with H, and sorted sets begin with Z. Operations in Redis are atomic, so you don't have to worry about what happens if you modify a set while someone else is reading it.

One of the nice things about the Redis data types is that they support the types of operations you would want. For example, you might want to have a counter of some sort. Well, Redis doesn't support a counter type per se, but if you store an integer (really, a string containing only digits) as a Redis scalar type, you can increment and decrement its value using the `incr` and `decr` commands. For example:

```
set a 10
incr a
```

returns:

```
11
```

And if you again say:

```
get a
```

you'll get:

```
11
```

Similarly, it's very useful to treat Redis' lists as stacks and queues. You can do this by combining the `lpush`, `rpush`, `lpop` and `rpop` commands. Indeed, you create a list by

invoking one of these commands; if the named list doesn't exist, Redis will create it. For example:

```
lpush numbers 1
lpush numbers 2
lpush numbers 1000
rpush numbers 555
```

Now you can ask for a list of numbers, from index 0 (the start) to index -1 (the end):

```
lrange numbers 0 -1
```

Redis responds:

```
1) "1000"
2) "2"
3) "1"
4) "555"
```

And of course, you can sort the list:

```
sort numbers
```

which returns:

```
1) "1"
2) "2"
3) "555"
4) "1000"
```

Sets are like lists, except that they're unordered and the elements are unique. For example, let's say that you want to add a bunch of users to your "users" set, using the `sadd` command:

```
sadd users reuven
sadd users root
sadd users atara
sadd users shikma
asdd users amotz
```

And, add three people to your "administrators" set:

```
sadd administrators reuven
sadd administrators root
sadd administrators joeshmoe
```

You now can ask Redis to tell you if anyone is listed as an administrator who isn't also a user:

```
sdiff administrators users
```

You also can ask Redis to find the intersection of these two sets:

```
sinter administrators users
```

I have become a real fan of sets in the last few years, in all of the programming languages that I use. The fact that they exist in Redis only extends my affection for them and the number

For tracking users, IP addresses and other such pieces of data, sets are pretty much unparalleled in their simplicity and utility.

of places in which I can and will use them to track things. For tracking users, IP addresses and other such pieces of data, sets are pretty much unparalleled in their simplicity and utility. If your data also needs to be sorted, as well as unique, you might well want to use the "sorted set" data type provided by Redis.

Finally, given that Redis is a key-value store (aka a hash table), it might seem odd that one of the Redis data types is itself a hash. Then again, this is also the most natural thing in the world for Redis to provide—and it's extremely useful, as well.

For example, you can say:

```
hset person first_name Reuven
hset person last_name Lerner
hset person email reuven@lerner.co.il
```

Not surprisingly, you then can use the `hget` command to retrieve one of these fields:

```
hget person email
```

and you get:

```
"reuven@lerner.co.il"
```

The `hexists` command takes a hash name and a key name, returning 1 if the key exists and 0 if it doesn't. If you try to retrieve a non-existing key, Redis (as usual) will return a "nil" value, rather than raising an exception or otherwise having problems with it.

There is much more to say about all of Redis's data types. In some ways, the fact that they provide such rich functionality makes Redis into something more than a plain-old key-value store. Although you can't say that Redis is a document or object database, it's doing more than just storing a single, simple value with each key.

Single-Server Redis

Installing, configuring and running Redis is simpler than you might imagine. In many cases, you can just download, install and run Redis within a few minutes.

On my Linux box running Ubuntu, for example, I was able to install the Redis server and redis-cli program (among other command-line tools) with the following command:

```
sudo apt-get install redis-server redis-tools
```

Once installed, I was able to start it with:

```
sudo service redis-server start
```

The redis-server configuration in the Ubuntu package is configured to be "demonized", meaning that the server runs in the background when you start it up. This is the behavior that you normally would like, if you're running a

server; although foreground execution can be convenient when debugging, it's otherwise pretty much unwarranted.

If you're running a simple Redis server, there's not more to things than that. There is a configuration file, and you might need to modify it, but I've always found the configuration to be a simple and straightforward process. The configuration file is in a typical UNIX-style format, with a long set of setting names followed by the appropriate values, each on its own line, and with hash marks (#) indicating comment lines.

The configuration consists of several well-documented sections. First is the basic configuration and networking, telling Redis on which port it should listen and the IP address(es) to which it should pay attention.

The next part of the configuration tells Redis how it should handle saving its current data set (which is in RAM) to disk. Older versions of Redis would save data to disk every two seconds, which was good enough for many purposes, but did mean that up to two seconds' worth of data might be lost.

An alternative system, known as AOF (append-only file), instead of writing the complete data set every two seconds, writes a constant stream of data to the disk, indicating what changes were made to the in-memory data set. Thus, when the system is restarted, it's possible for Redis to replay the AOF and get back to the state it was in before the shutdown occurred. The Redis documentation suggests using both persistence systems together for maximum safety.

In the Redis configuration file, you can indicate not

How should you tell Redis to back up your data? And, should you use RDB, AOF or both?

only where the files should go, but also the rules that govern when the snapshots should be put on disk. The "save" configuration directive takes two numbers for how often saves should be made, giving Redis an indication of how long it should wait to write information to disk, based on how many keys have changed. Thus, the default lines in my configuration file:

```
save 900 1
save 300 10
save 60 10000
```

mean that Redis should snapshot to disk after 900 seconds if one key has changed, after 300 seconds if ten keys have changed and after 60 seconds if 10,000 keys have changed. In other words, Redis will ratchet up the frequency of dumping itself to disk when the number of changed keys changes.

How should you tell Redis to back up your data? And, should you use RDB, AOF or both? If the data is important to you, I'd suggest going with the whole thing, storing data using both methods for maximum safety. However, if you're just using Redis as a cache, and if all data exists elsewhere, the fact that Redis provides persistence is a nice

bonus, rather than something critical, and you can be a bit more lax—possibly by using only the RDB snapshots and not employing the AOF.

Perhaps the most important thing to realize when configuring Redis is that for a large number of cases, the system requires little or no tuning. You install it, modify the configuration a bit, and you're pretty much set to go. Certainly, this has been one of the reasons why I have so enjoyed working with Redis for many years.

Multi-Server Redis and Replication

However, as applications have grown, so have the data needs of those applications. Redis can handle a very large number of clients, almost certainly because it's a key-value store, rather than a full-fledged database. Even so, Redis also has its limits. Unlike some NoSQL databases, support for sharding is new and, thus, still (at the time of this writing) somewhat untrusted. Without sharding, the entire data set exists in a single computer's memory. This means that the single computer in question can become the bottleneck if too many clients are trying to read or write data.

One solution to this problem is the use of master-slave replication. The idea behind such replication in Redis, as in other master-slave database configurations, is that one computer is designated as the "master", meaning where the changes actually take place. In such a configuration, only the master computer will accept write requests. The assumption is that reads constitute the majority of queries, and thus, they can be spread across one or more slave machines, balancing the load. (Actually, you can configure

Redis such that it accepts write requests, but given that writes to the slave aren't permanent, the uses cases for this appear to be fairly rare, and it's quite possible that this functionality will go away in the future.)

Master-slave replication in Redis is easy to set up. First, you need to have two computers running Redis or (if you just want to test things) a computer with two instances of Redis, each running in a different directory and on a different port. On the slave, you add the configuration directive:

```
slaveof IP PORT
```

where `IP` and `PORT` should be set to the IP address and port of the master Redis server. If you have configured your master Redis server to require a password for authentication, you'll need to add the password to your client's connection invocation.

Once you've got this set up, master-slave replication should work fairly well. You can read from or write to the master, or you can read from the slave. You can set up multiple slave servers, and you even can tell Redis that the master should accept write requests only if the number of active slaves is higher than a certain minimum (min-slaves-to-write) threshold.

Master-slave replication in Redis would thus appear to be a good solution for a large number of organizations. However, as organizations are scaling up, they're sometimes finding that master-slave doesn't quite do the job well enough. In such cases, they're moving toward Redis clusters. In master-slave replication, as you've seen,

each of the computers (the master and all slaves) have a complete copy of the data set. In a Redis cluster, the data is "sharded", meaning that it's split automatically among a number of servers. If you have six Redis servers, no one server will contain all of your data; it will be contained across a number of them. This not only means greater availability of your data, but it also helps ensure that if one or more of the cluster's nodes are unavailable, the data still will be reachable via one or more of the remaining servers.

Redis is known for both its performance and its reliability. Clustering changes this somewhat, in that it makes Redis a distributed system, with some of the problems inherent in such a system. For example, it's possible that a write query, already acknowledged to a Redis client, will have its data invalidated, because another Redis node received a competing write that won out. It's possible that Redis nodes will go up and down, or that data will be slightly different and inconsistent across different nodes.

To some degree, many of the questions about Redis clustering still are unanswered, because the cluster software was only released into production on April 1, 2015, as part of Redis 3.0.0. How reliable is it? How easily does it handle failures? What's involved in scaling a cluster up to 1,000 nodes or so?

I say these things not to take away from the amazing progress that Redis has made through the years, nor to detract people from trying out Redis clusters. That said, this is a new technology—and any new technology, particularly one that adds an element of distributed computing, is likely to require at least a short shaking-out period.

Now, you might be wondering what a chip architecture has to do with Redis. The answer lies in the CAPI (coherent accelerator processor interface) that is available in the POWER8 architecture.

IBM's CAPI Technology

Another new innovation that might be of interest to large-scale Redis users has come from a somewhat unlikely source, IBM. IBM has long been developing and promoting its Power architecture, with the POWER8 being the latest in this series.

Now, you might be wondering what a chip architecture has to do with Redis. The answer lies in the CAPI (coherent accelerator processor interface) that is available in the POWER8 architecture.

The basic idea is as follows. Redis normally stores all of its data in memory. This means if you have a very large data set, you're going to need to have a large investment in RAM, which is quite expensive. With the release of Redis 3.0, you have an alternative, which is to create a Redis cluster. However, this means having multiple machines, which also can be expensive.

The ideal solution would be to have a single machine with cheaper RAM. That's not going to happen in the near future. We do have Flash memory, which might not be as fast as RAM, but it's certainly faster than hard disks; however, the big problem there is the latency. The usual

way a computer and storage communicate isn't fast enough to give you the Redis performance you're looking for.

IBM's CAPI provides an interface that makes it possible to offload processing to an external subsystem. This offloaded processing can be used to complement actual processing of computation-intensive algorithms or for high-speed storage. In this particular case, CAPI makes it possible for the CPU to speak directly with the external Flash storage system, reducing latency and, thus, making Flash more comparable to RAM than to a typical storage system in its responsiveness. Perhaps Flash won't be as fast as RAM, but you're looking at pretty-fast/not-too-expensive Flash, which is often a reasonable trade-off.

According to the white paper "Data Engine for NoSQL—IBM Power Systems Edition" distributed by IBM engineers, their system provides a special API for NoSQL. Now, I have to admit I was a bit skeptical about a "NoSQL API", given the wide variety of different types of NoSQL databases out there. What I take from their description is that they have provided a special API that supports key-value stores such as Redis. The API doesn't currently support other NoSQL databases, such as MongoDB. However, in my conversations with IBM staff, it was clear that they are interested in pursuing this route, and that we probably can expect to see additional NoSQL offerings in the future.

Now, these new features are fairly hardware-specific, and the out-of-the-box Redis installation doesn't know how to take advantage of them. So, IBM has partnered with Redis Labs, a private company that offers high-capacity Redis hosting and software, to create a special version of Redis that can work with this hardware. In a conversation I had with Yiftach Shoolman, founder and CTO at Redis Labs, he

But perhaps the most interesting part of this, at least to me, is the fact that the system as provided leaves the mix of RAM and Flash up to the user.

said that from the programmer's perspective, there is no difference between the API that Redis Labs' implementation offers and that of the standard Redis.

The basic idea, then, is that by using IBM's hardware in conjunction with Redis Labs software, you'll be able to have a high-speed, large-capacity Redis instance. But perhaps the most interesting part of this, at least to me, is the fact that the system as provided leaves the mix of RAM and Flash up to the user. You can configure IBM's system to use lots of RAM and little Flash, for a very high-speed, but expensive, system. By contrast, you can use lots of Flash and very little RAM. The ultimate balance is left up to you, set via a Web-based control panel that tells the POWER8 system how to configure and balance the mix of RAM and Flash. The paper provided by IBM estimates that when about 80% of the storage is in Flash, users still will benefit from very good performance. Because the control panel makes it easy and straightforward to change the mix of RAM and Flash, I expect that many customers, or potential customers, of IBM's storage solution will do a number of tests, trying a variety of different combinations before finding the sweet spot between performance and price.

Using Redis with CAPI

For the purposes of this Geek Guide, IBM provided me with access to a set of systems using the POWER8 CAPI interface and Redis. Specifically, I was given access to three different machines:

- A virtual Linux machine running Red Hat Enterprise 6.6. This machine was running PHP and Apache, with a Web application that could be used to monitor and configure the Redis Enterprise server.

- A POWER8 system running Ubuntu 14.10, with 20 cores and 256GB of memory. It was on this box that Redis Enterprise Edition was running (more on that below). This box had the CAPI adapter to which the Flash storage unit was attached.

- The third box was an IBM FlashSystem 840. Although it had its own IP address and could be monitored via a Web application, it is a storage system, and thus doesn't run an operating system that is directly of interest here.

The above configuration is a small, if reasonable, facsimile of the configuration that you would expect organizations to use if and when they want to use IBM's Flash solution. Of course, anyone who is interested in using this high-powered Redis solution likely is going to have more than a single Web server. Given that Redis provides a server, the only difference between this sample configuration and the one that you would use for yourself is the number of servers. Each one would be outfitted with an appropriate Redis client, connected to the Flash server.

Remember that the version running on the IBM system is

using a system from Redis Labs, not the usual open-source Redis edition. The behavior should be identical from the client's perspective; the difference is in the configuration. However, the Redis Labs system is running a variant of Redis 2.8.12, which means that the Redis 3.0.0 features you might want will not be available.

Whereas a normal Redis system, as described above, will have a single process running, the Redis Labs system uses a number of separate processes. Each Redis instance has its own configuration file, which it references when it starts up. However, you're not supposed to change those configuration files directly. Rather, you're supposed to use a Web-based GUI, which then uses a back-end API to modify the configuration files for you.

On the system I used, there were two different types of Redis servers actually running. One was called "redis-server", and the other was called "redis-server-big", reflecting whether my Redis instance was running against RAM or Flash. I could use the Web-based control panel to add or remove Redis server instances running against each of the types of memory.

The Web control panel that Redis Labs provides makes it easier to monitor and configure a Redis installation, especially when you start to work with multi-machine clusters. It centralizes the configuration functionality, allowing you to add and remove Redis server instances according to your load-balancing needs. At the same time, as someone who is used to working with text-based configuration files, it took some time to understand the mapping between those files and the configuration control panels used by the Redis Labs enterprise cluster. Adding and removing Redis instances couldn't have been simpler though.

Moreover, changing the balance between RAM and Flash

The Redis Labs control panel also provides you with monitoring capabilities.

storage was as simple as going into the configuration's control panel for the Flash system and moving a slider. It's thus possible to experiment with a number of different potential configurations, trying them out by moving the slider and seeing how the system behaves.

The RAM/Flash split configuration means that you're configuring two different Redis instances, one for RAM and one for Flash. Redis clients connect to a single port, and they aren't aware of which back end they are using or where the storage sits. Each of the configured Redis systems (RAM and Flash) can be set separately to use either of the Redis persistence models (AOF or a snapshot). In the case of snapshot persistence, you must tell Redis how often to dump the contents of the Redis database to disk.

The Redis Labs control panel also provides you with monitoring capabilities. You can see the number of operations/second that each of the two nodes (RAM and Flash) are performing, and you can break that down over time to see the latest few minutes, day or even year. In this way, you can keep your fingers on the pulse of your Redis machine and understand whether you need to rebalance the RAM vs. Flash ratio.

Is It Appropriate for Your Needs?

I must admit that I'm more of a software guy than a hardware guy. But, in speaking with some of the IBM engineers who

were involved in creating the new Power-backed Redis system, I was impressed by what I heard. The idea of basically plugging the I/O system in to a lower-latency, higher-bandwidth communication channel means you are no longer restricted to using RAM for high-speed I/O.

I asked the IBM engineers if the open-source version of Redis will be able to take advantage of this architecture, or if databases, either SQL or NoSQL, will do so in the future either. They indicated that Redis, and the commercial Redis Labs implementation, is one of the first things they've done with their CAPI connection, and that other things might well be coming down the pike later on. Redis Labs does support a version of Memcached for the Power systems, if you're interested in using it rather than Redis, although I'm not sure what advantages there would be in doing so. But for now, the commercial Redis Labs implementation is one of the first ways that people will be able to take advantage of this technology.

This raises the question of whether this is worth considering. The answer, from everything I've read and seen, is that it's definitely worth consideration, but the trade-offs will come down to a business decision. If your Redis installation is large enough that you need to find a high-speed solution, and if RAM is simply too expensive for you to consider, then you have to decide between a Redis cluster and IBM's offering. The trade-off there is between buying and managing a number of servers, and buying the IBM product and the Redis Labs commercial license—but a single server, rather than a number of them. According to information provided by IBM, the POWER-based Redis store provides greater ROI as the size of the storage increases, reaching as much as 3x savings for a 40TB

Flash storage system. IBM's products also mean that you replace a large number of servers and storage systems with a single server and a single storage unit, making it easier to maintain as well as consume less rack space and electricity. If you're looking at multiple, large storage boxes for your Redis system, IBM's new products definitely are worth a look and comparison.

Both the Redis cluster and IBM's offering are new on the market and haven't been used by many organizations yet. It will be interesting to see how each of them fares. I'm optimistic that some of the largest Redis installations will find this solution from IBM and Redis Labs to be one that saves time and money.

Conclusion

Redis has long been a popular, stable, feature-rich and high-speed key-value store. I'm not surprised to hear that it's the most popular NoSQL solution currently in use. As I described here, its data types provide a wide variety of functionality, and the fact that it provides both in-memory speed and persistent, on-disk storage is an unbeatable combination. However, as Redis has grown in popularity, and as companies have used it for larger and larger data sets, it has hit a bit of a wall. Master-slave still assumes that all of the data can reside in the memory of a single computer. Redis cluster is new, and it requires the configuration and maintenance of multiple machines. This new offering from IBM and Redis Labs promises to give large-scale Redis installations a smaller footprint, easier management and easier configuration, thanks to a combination of the RAM/Flash mix and a useful Web-based GUI. If you are currently managing a large Redis installation and worry about how you will continue to scale it up, you would be wise to consider this new product from IBM.■