ALLAN THRAEN | 6 years ago | PDF |

Tips and Tricks Azure .NET Development

AZURE STORAGE PERFORMANCE SHOWDOWN (POST 3)

This is the 3rd post in my Azure Storage Performance comparison. So far we've examined the typical scenario of storing/retrieving data that most dynamic websites of today deal with. In this post, we'll take a closer look at Update and Delete - and finally review the financial aspects.

Update & Delete

Let's start where we left of in the earlier posts. By now we've completed half of the CRUD (Create, Read, Update, Delete) test and the last two parts are left. Again, we're working with a dataset of 1 million StackExchange questions in various forms of storage.

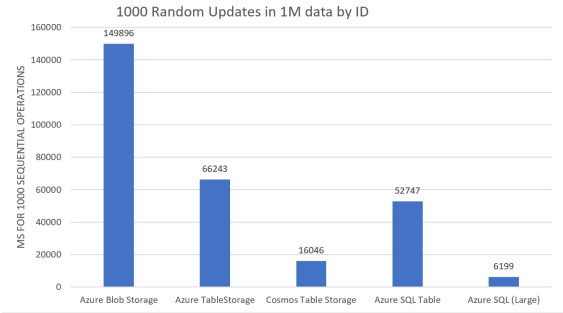
The update scenario I have chosen is again trying to be as realistic as possible: An answer has been added, so we'll increase the "AnswerCount" by 1. Again, we'll do this 1000 times with randomly picked data in the dataset, and we'll do it in the most optimal (but this time sequential and one-by-one) way for each data store.

As per usual I asked my fellow tweeps what they'd expect:

> Tweeps! Final poll for my @Azure storage performance blog post series: Which service would you expect to be fastest at doing 1000 sequential random updates in 1M dataset?
>
> — Allan Thraen (@athraen) September 25, 2018

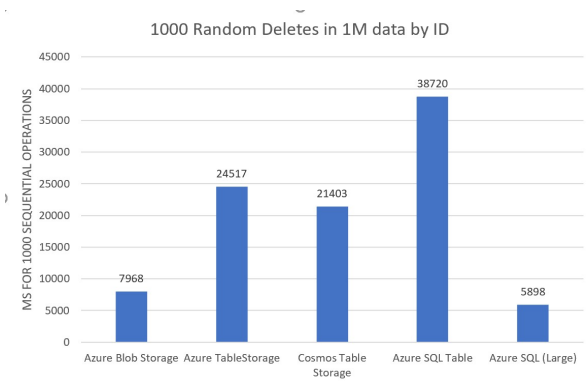
One thing to note when we examine the results is that when it comes to Blobs, Table Storage & Cosmos, I had to get the object first, then add 1 and then save it back - where as with SQL Azure I could get away with a simple "UPDATE test SET AnswerCount = AnswerCount + 1 WHERE Id = @Id" statement. There might be a similar smart way to do it with Cosmos (I've noticed there is some kind of stored procedures) but since I'm using the Table API to Cosmos, I didn't see any obvious approach other than using the same as I did with regular Table Storage.

Let's check out the results:



Not surprising we have Blob storage on the slow end with 150 ms per operation on average. But remember it has to fetch a json file, deserialize it, add, serialize it and re-upload the file to the server. Table storage comes in second place at 66 ms. Not bad considering it goes through a fairly similar flow. Meanwhile I'm surprised both that the S0 SQL server is so slow at it - but then again, it's been slow all along - and I think it's worth keeping in mind that if you go with SQL you want to put enough power behind your operations to not spend your life waiting for the requests to complete :-) When it comes to the tweet, I think we can declare Cosmos once again the clear winner (as the S3 SQL server wasn't an option).

Now, considering deleting an object. The setup is the same as we know it. 1000 randomly picked ID's should be deleted from the 1 million records dataset.



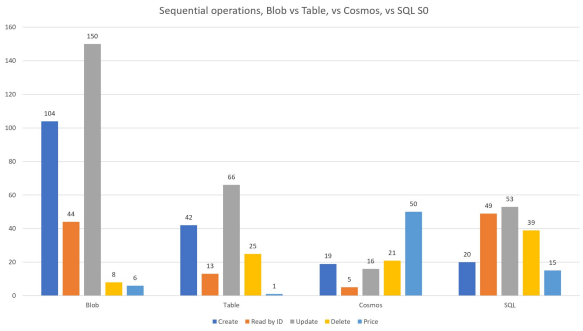
Wow! Can it be? Do we really have situation where Blob storage is the winner? Well - I think the answer is fairly straightforward here: Since the ID is the filename, deleting it is simply a single call to Blob storage with the command to delete a certain blob. It might not even be deleted straight away, maybe just marked for deletion and it'll then happen soon after. Who knows. Both TableStorage and Cosmos are challenged by the fact that the Table API's DeleteOperation takes a full Entity, which means that we first have to fetch the full entity like this:

```
TableOperation retrieveOperation = TableOperation.Retrieve<Question>(q.PartitionKey, q.RowKey);
var res = table.Execute(retrieveOperation);
TableOperation del = TableOperation.Delete(res.Result as Question);
table.Execute(del);
```

Now, why SQL Azure S0 is so slow, I have no idea - but we can at least appreciate that it's slightly faster at deleting than it is at updating...

That's all fine and great - but let's get down to business...

If we look across all of these options there are several factors to take into account. One thing is how they perform when you call them once - but scaling both in terms of data-volume and in terms of simultaneous calls is also very interesting. And for some of us, young starving coders and data-addicts, price also plays a significant role in picking the right storage.



Here, we look at the different CRUD features along side a price point. The price is the approx. amount of USD I estimate it would cost per month to hold around 1 million of my data records with 1 million read and 1 million write operations. Cosmos is calculated at 800/1000 RU and SQL is the small S0 with only 10 DTUs. In all cases I've reserved around 5 Gb for storage (although the data is probably only about half that). Here are all my numbers:

Test	Azure Blob Storage	Azure TableStorage	Cosmos Table Storage	Azure SQL Table	Azure SQL (Large)
Create 100k, 1 thread (sequential)	10399472	4236882	1882995	2032763	n/a
ms per operation	*103,99472*	*42,36882*	*18,82995*	*20,32763*	*n/a*
Create 100k, 20 threads (parallel)	5246389	2123452	178591	2384546	n/a
ms per operation	*52,46389*	*21,23452*	*1,78591*	*23,84546*	*n/a*
Create 1M, 20 threads (parallel)	51775772	20375302	1021991	n/a	3210180
ms per operation	*51,775772*	*20,375302*	*1,021991*	*n/a*	*3,27018*
1000 Random Reads from 1M data by ID. 1 thread	43927	12832	5076	49976	1825
ms per operation	*43,927*	*12,832*	*5,076*	*49,976*	*1,825*
1000 Random Reads from 1M data by ID. 20 threads	17605	7429	12421	30657	627
ms per operation	*17,605*	*7,429*	*12,421*	*30,657*	*0,627*
1000 Random Reads from 1M data by ID. 40 threads	16512	9785	10637	27326	639
ms per operation	*16,512*	*9,785*	*10,637*	*27,326*	*0,639*
100 Random Reads from 1M data (non-indexed field)	n/a	813356	3631	(timeout)	334592
ms per operation	*n/a*	*8133,56*	*36,31*	*n/a*	*3345,92*
1000 Random Updates in 1M data by ID	*149896*	*66243*	*16046*	*52747*	*5199*
ms per operation	*149,896*	*66,243*	*16,046*	*52,747*	*6,199*
1000 Random Deletes in 1M data by ID	*7968*	*24517*	*21403*	*38720*	*5898*
ms per operation	*7,968*	*24,517*	*21,403*	*38,72*	*5,898*
Setup	**Hot (LRS)**	**Hot (LRS)**	**RU at 800 tuned 10.000 R/U for parallel creation, then back down**	**S0, 10 DTU, 5Gb**	**S3, 100 DTU, 20 GB**
Approx. Max Monthly Cost (USD)	*~65*	*Less than $1*	*50$ / 600$ (large)*	*$15*	*$150*

Let's summarize a bit on these options:

1. Blob Storage: Actually, I don't think the timings are bad all things considered. And it can certainly handle the stress when you scale it up. However, the serializing and deserialization is sort of killing the mood so I think I'll keep my Blobs for the binary data they are so well suited for!.
2. Honestly, this is still my personal favorite of the 4. Scaling datavolume doesn't seem to affect it at all. And according to MSFT it can easily handle 20.000 operations per second. Sure, the update/delete's took 2 roundtrips and hence were a bit slower - but honestly, when are you in a rush to delete stuff? Write and Read speeds usually matter the most - and at 42ms / 13 ms that's pretty damn impressive. Oh - and the best part: 1 month running this database would cost less than 1 USD!
3. Cosmos is certainly the new kid on the block. And it did deliver! Like with table storage the datavolume seems not to matter at all. And when it comes to simultaneous calls it can handle pretty much whatever you can throw at it. BUT - since it charges by the number of Request Units it'll handle (meaning the scale) you increase your RU limit when doing more requests. And those are expensive. The calculated cost when I switched it up to 10.000 RU for one of the tests (1M indexing) is close to 600 USD / month!
4. Good ol' SQL Server Aheem..I mean SQL Azure. Again, you get what you pay for. There is a low entry with the S0 (and some even cheaper basic ones which are pretty worthless), but even though it's not Blob provider bad, I'm not that impressed. Of course - the moment we start throwing money at it, we get incredible performance as we see in the numbers on the S3 (Large) instance. 1,8 ms for a read - yeah, that's fast! However, when we look at some of the numbers with multiple threads it's worth remembering that SQL probably gets some of that power from it's connection pooling - something the others lack.

I hope you have enjoyed this little Performance Series. I'll leave it at this for now, but I do have an idea for a little follow-up, that I'll probably post soon!

Make sure to drop me a comment here if you liked it!

Tips and Tricks | Azure | .NET Development

Azure Storage Performance Showdown
Azure Storage Performance Showdown (Post 2)
Storage Performance Aftermath - ElasticSearch Joins the Fight

CodeArt ApS
Teknikerbyen 5, 2830 Virum, Denmark
Email: info@codeart.dk
Phone: +45 26 13 66 96

CVR: 39680688