Preparation

Set your max memory to 1GB, parallelism to defaults:

```
EXEC sys.sp_configure N'max server memory (MB)', N'1024';
EXEC sys.sp_configure N'cost threshold for parallelism', N'5';
EXEC sys.sp_configure N'max degree of parallelism', N'0';
```

Restart your SQL Server

Drop all nonclustered indexes on StackOverflow.dbo.Comments

Set up a new window for sp_BlitzFirst:

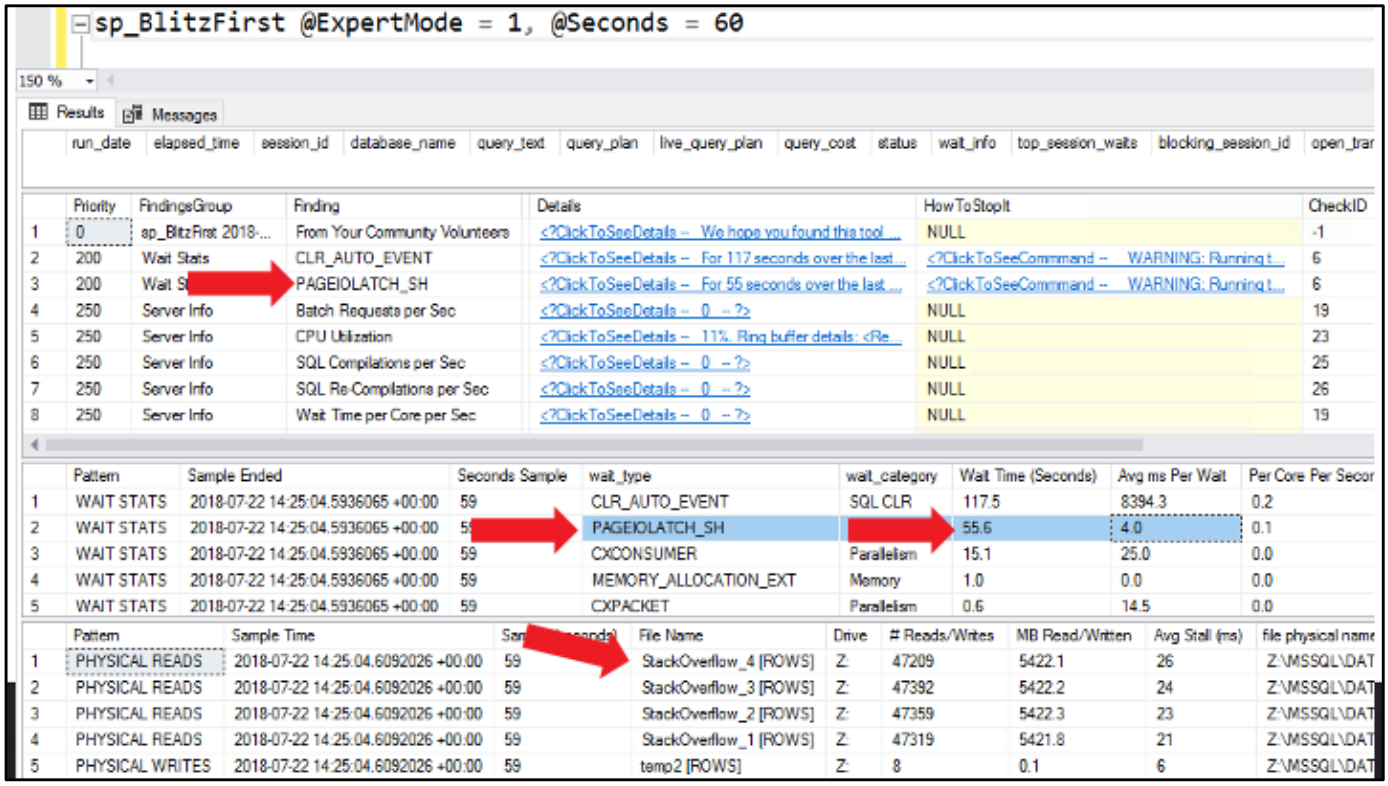
```
sp_BlitzFirst @ExpertMode = 1, @Seconds = 30
```

1.3 p2

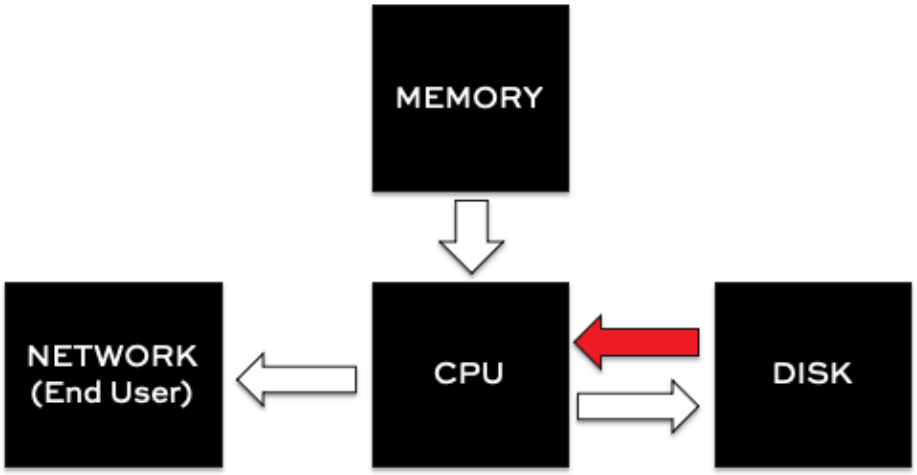
Simple Select

```
SELECT *
FROM StackOverflow.dbo.Comments
WHERE UserId = 26837
```

1.3 p3



PAGEIOLATCH:
Reading the data file from disk



1.3 p5

Data you have to read

/ How fast you read

Minimum query runtime

1.3 p6

Reading millions of 8KB pages

```
SELECT * FROM dbo.Comments WHERE UserId = 26837;
```

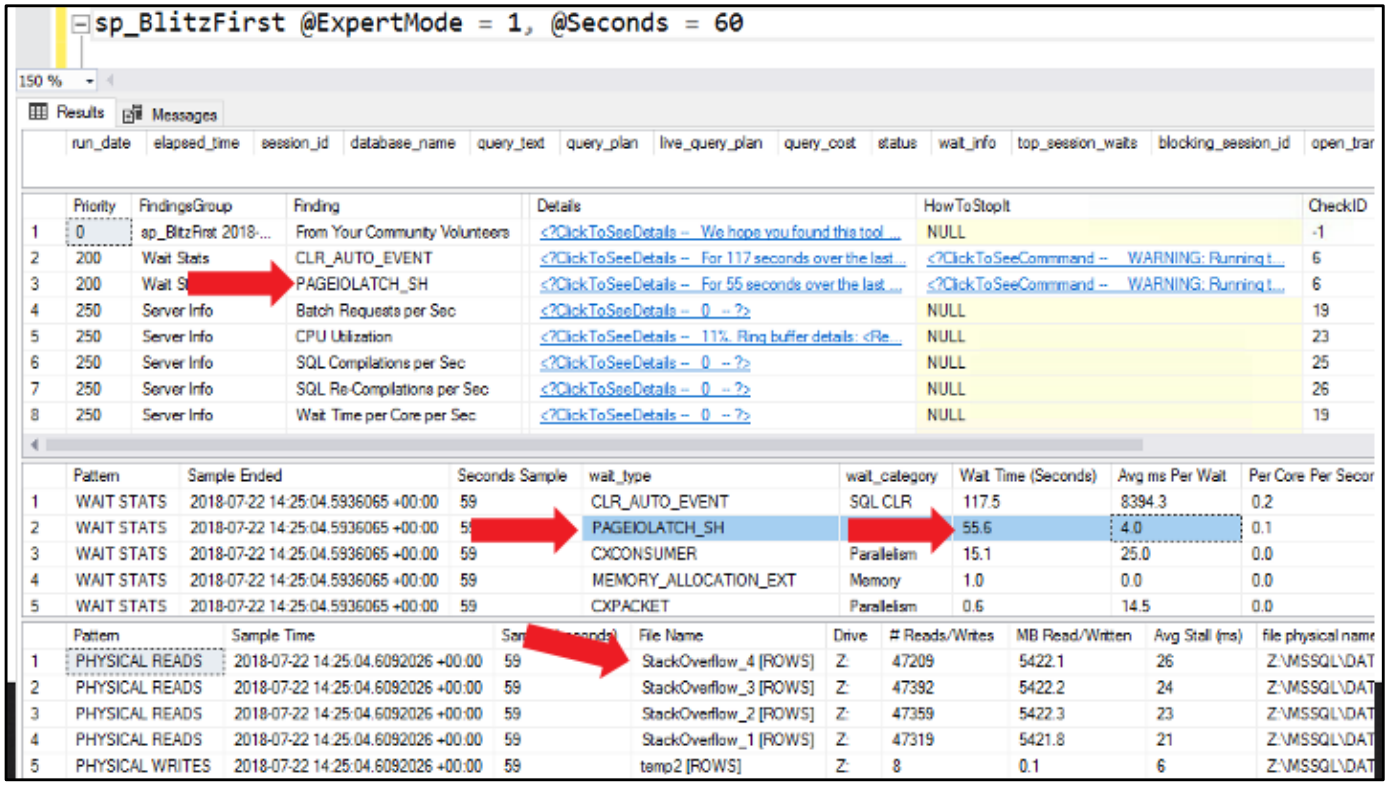
150 %

Results | Messages

```
(150 rows affected)
Table 'Comments'. Scan count 9, logical reads 2765260,
physical reads 13, read-ahead reads 2762692,
lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
```

2,765,260 pages * 8KB each / 1,024 = 21.6GB

1.3 p7



So how do we tune this?

Data you have to read
/ How fast you read
Minimum query runtime

1.3 p9

How to reduce PAGEIOLATCH

1. Tune indexes, looking at missing indexes:
   `sp_BlitzIndex @GetAllDatabases = 1`
2. Tune queries:
   `sp_BlitzCache @SortOrder = 'reads'`
3. Add more memory
4. Make storage faster

What index could you make?

```
/* Our query: */
SELECT *
FROM dbo.Comments
WHERE UserId = 26837
```

Fixing it with indexes

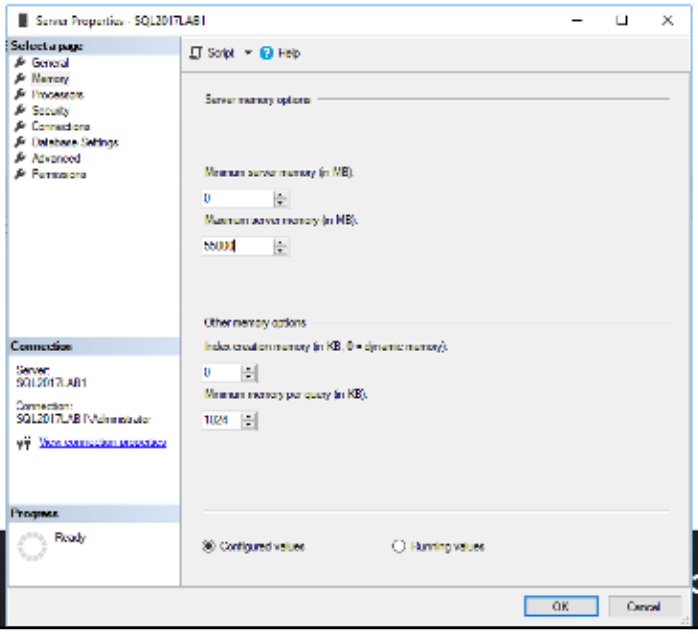
```
SELECT * FROM dbo.Comments
WHERE UserId = 26837

/* Our index: */
CREATE INDEX IX_UserId ON
dbo.Comments (UserId);
```

Before you do that, add memory.

Otherwise, trust me, the index build takes forever.

My default: leave 4GB or 10% free for the OS, whichever is greater.



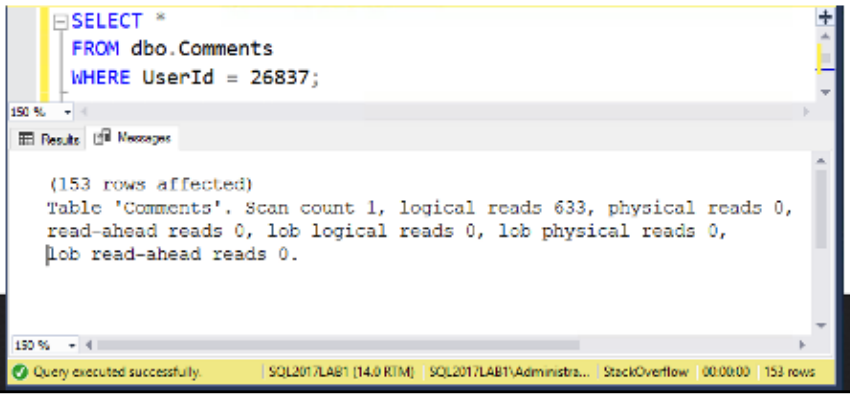
With the index, we read less pages

Under 1K pages down from millions, and finishes in a second

Frees up our memory for other queries

Reduces the workload on our slow storage

We don't have to cover the whole query – just one column helps



1.3 p14

With the index, we read less pages

	Pattern	Sample Ended	Seconds Sample	wait_type	wait_category	Wait Time (Seconds)	Per Core Per Second	Signal Wait Time (Seconds)	Percent Signal Waits	Number of Waits	Avg ms Per Wait
1	WAIT STATS	2018-02-13 22:33:30.8702564 +00:00	29	ASYNC_NETWORK_IO	Network IO	0.4	0.0	0.0	0.0	22	18.9
2	WAIT STATS	2018-02-13 22:33:30.8702564 +00:00	29	MEMORY_ALLOCATION_EXT	Memory	0.0	0.0	0.0	0.0	5351	0.0
3	WAIT STATS	2018-02-13 22:33:30.8702564 +00:00	29	PAGELATCH_SH	Buffer Latch	0.0	0.0	0.0	0.0	19	0.5
4	WAIT STATS	2018-02-13 22:33:30.8702564 +00:00	29	PREEMPTIVE_OS_WRITEFILE	Preemptive	0.0	0.0	0.0	0.0	1	1.0

	Pattern	Sample Time	Sample (seconds)	File Name	Drive	# Reads/Writes	MB Read/Written	Avg Stall (ms)	file physical name
1	PHYSICAL READS	2018-02-13 22:33:30.8858857 +00:00	29	tempdev [ROWS]	Z:	3	0.2	0	Z:\MSSQL\DATA\tempdev.mdf
2	PHYSICAL READS	2018-02-13 22:33:30.8858857 +00:00	29	MSDBData [ROWS]	D:	8	0.5	0	D:\MSSQL14.MSSQLSERVER\MSSQL\DATA\MSDBData.mdf
3	PHYSICAL READS	2018-02-13 22:33:30.8858857 +00:00	29	StackOverflow_1 [ROWS]		38	2.4	0	Z:\MSSQL\DATA\StackOverflow_1.mdf

Waits down under 1 second.

The quality of the storage simply no longer matters.

Storage stalls disappear because less work means faster average response time per task.

1.3 p15

How to reduce PAGEIOLATCH

1. ~~Tune indexes, looking at missing indexes:~~
   ~~`sp_BlitzIndex @GetAllDatabases = 1`~~
2. Tune queries:
   `sp_BlitzCache @SortOrder = 'reads'`
3. Add more memory
4. Make storage faster

But if we can't tune indexes

I'd rather tune indexes first.

The right indexes make *lots* of queries faster.

Tuning individual queries can take a lot longer.

1.3 p17

Can we “fix” this query?

```
SELECT * FROM dbo.Comments
WHERE UserId = 26837
```

1. Don’t select columns you don’t need
2. Don’t select rows you don’t need (paginate)
3. Cache in the application layer

We cover these in Mastering Query Tuning.

How to reduce PAGEIOLATCH

1. ~~Tune indexes, looking at missing indexes:~~
   ~~`sp_BlitzIndex @GetAllDatabases = 1`~~
2. ~~Tune queries:~~
   ~~`sp_BlitzCache @SortOrder = 'reads'`~~
3. Add more memory
4. Make storage faster

But if we can't fix those...

1.3 p19

Dr. Phil says,

you don't solve money problems with money.

1.3 p20

Dr. Brent says,

you don't solve storage problems with storage.

1.3 p21

Dr. Brent says,

you solve storage problems with memory.

1.3 p22

Systems Performance by Brendan Gregg

1 CPU cycle	0.3 ns
Level 1 cache access	0.9 ns
Level 2 cache access	2.8 ns
Level 3 cache access	12.9 ns
Main memory access	120 ns
Solid-state disk I/O	50-150 µs
Rotational disk I/O	1-10 ms
Internet: SF to NYC	40 ms
Internet: SF to UK	81 ms
Internet: SF to Australia	183 ms

1.3 p23

Systems Performance by Brendan Gregg

1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access	120 ns	6 min
Solid-state disk I/O	50-150 µs	2-6 days
Rotational disk I/O	1-10 ms	1-12 months
Internet: SF to NYC	40 ms	4 years
Internet: SF to UK	81 ms	8 years
Internet: SF to Australia	183 ms	19 years

1.3 p24

How much memory do we need?

dbo.Comments table: about 22GB

But that's just this one table, and not:

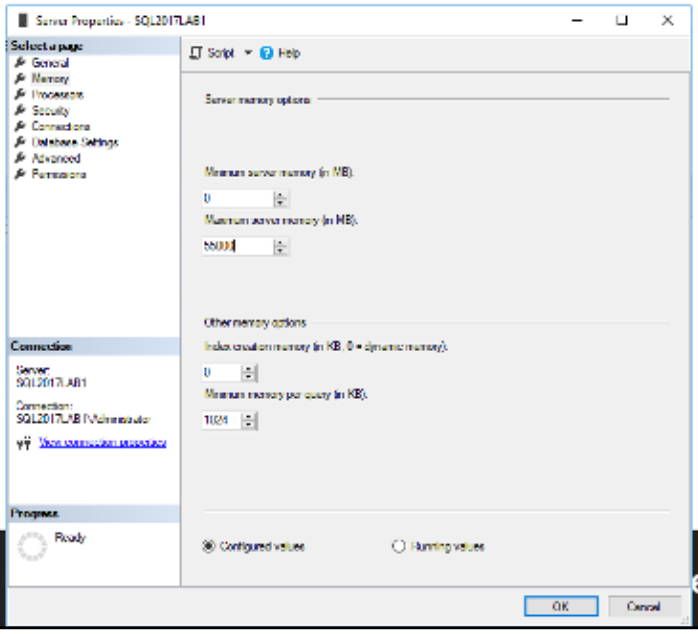
- Other tables we query
- Caching execution plans
- Query workspace memory for sorts, joins, etc.

1.3 p25

Take off the handcuffs

If you temporarily restricted memory, crank it back up to high.

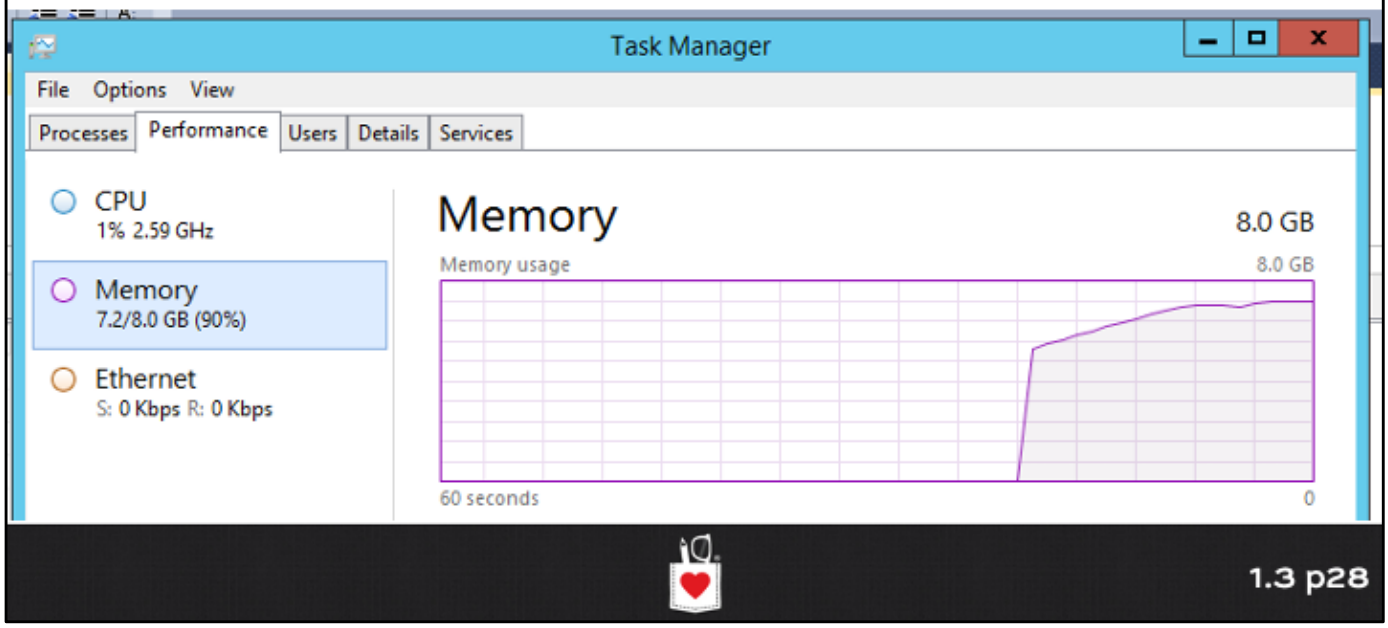
In our lab, this lets us cache the entire table in memory.



Then run the query a few times.

```
/* Our query: */
SELECT *
FROM dbo.Comments
WHERE UserId = 26837
GO 5
```

SQL starts using more memory



The art of fixing it with RAM

You don't have to cache the whole table in memory.

But the more you cache, the less storage has to work.

1.3 p29

How much RAM to throw at it

Physical boxes:

- 2012-2014 Standard: put 96GB RAM in the box
- 2016+ Standard: put 144GB in the box
- Enterprise Edition: 256GB

Virtual machines: use the Tequila Technique

Cloud: dictated by your # of cores

How to reduce PAGEIOLATCH

1. ~~Tune indexes, looking at missing indexes:~~
   ~~`sp_BlitzIndex @GetAllDatabases = 1`~~
2. ~~Tune queries:~~
   ~~`sp_BlitzCache @SortOrder = 'reads'`~~
3. ~~Add more memory~~
4. Make storage faster

But if we can't fix those...

Let's do the math.

1.3 p32

Reading millions of 8KB pages

```
SELECT * FROM dbo.Comments WHERE UserId = 26837;
```

150 %

Results | Messages

```
(150 rows affected)
Table 'Comments'. Scan count 9, logical reads 2765260,
physical reads 13, read-ahead reads 2762692,
lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
```

2,765,260 pages * 8KB each / 1,024 = 21.6GB

1.3 p33

If the users want it in 2 seconds...

How much data do we need to read?	21,600 MB
How long will the query take?	2 seconds
Divide the two to get throughput required:	10,800 MB per second

1.3 p34

Larger example, data warehouse

How much data do we need to read?	250,000 MB (250GB)
How many seconds can we take?	30 seconds
Divide the two to get throughput required:	8,333 MB per second

1.3 p35

Benchmark your current storage

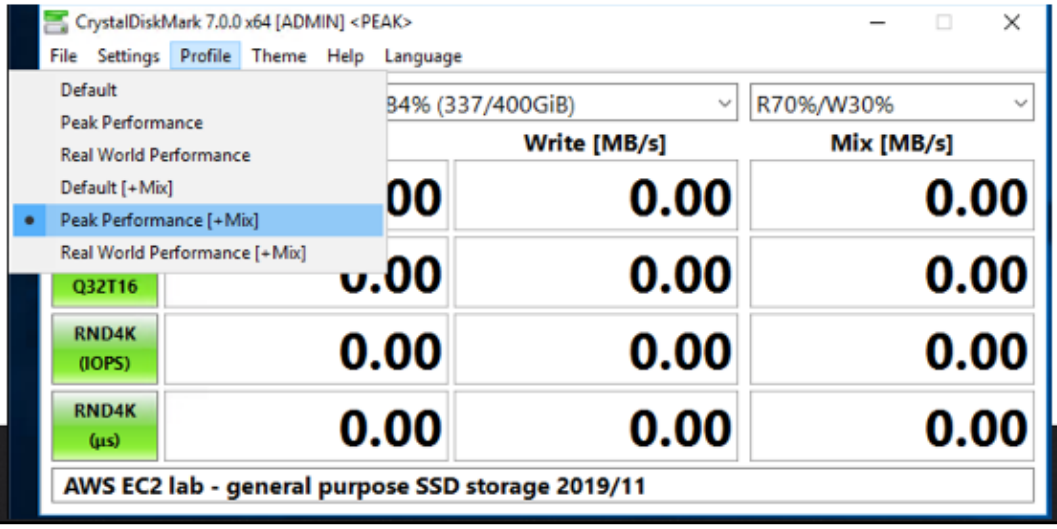
CrystalDiskMark: BrentOzar.com/go/cdm

Get the zip edition without ads, save it to a share

1.3 p36

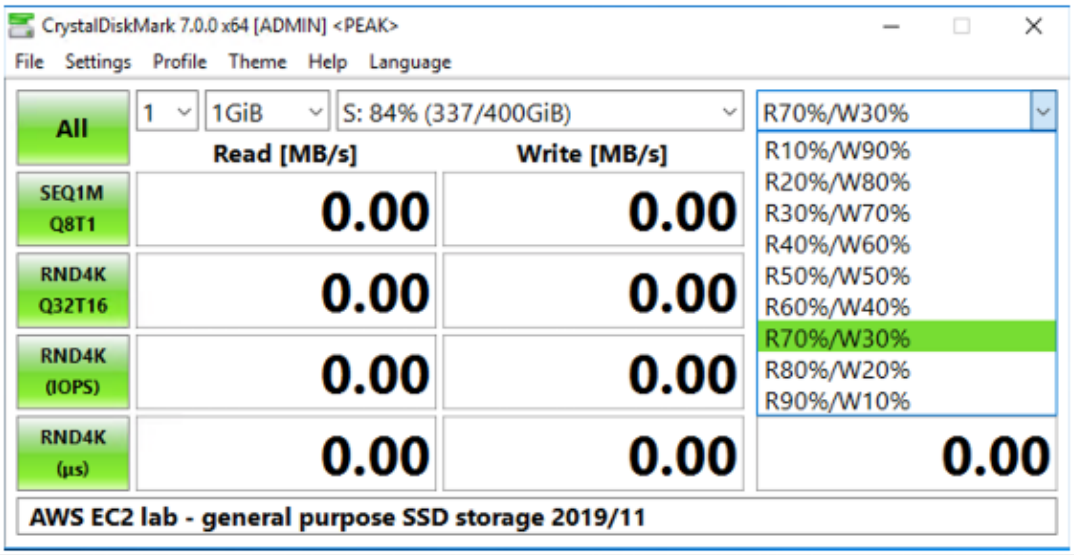
Setting it up

Click Profile, Peak Performance + Mix

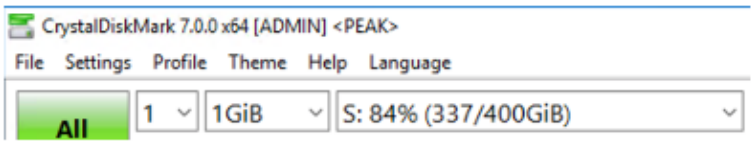


1.3 p37

Bonus points: choose your mix



Settings across the top

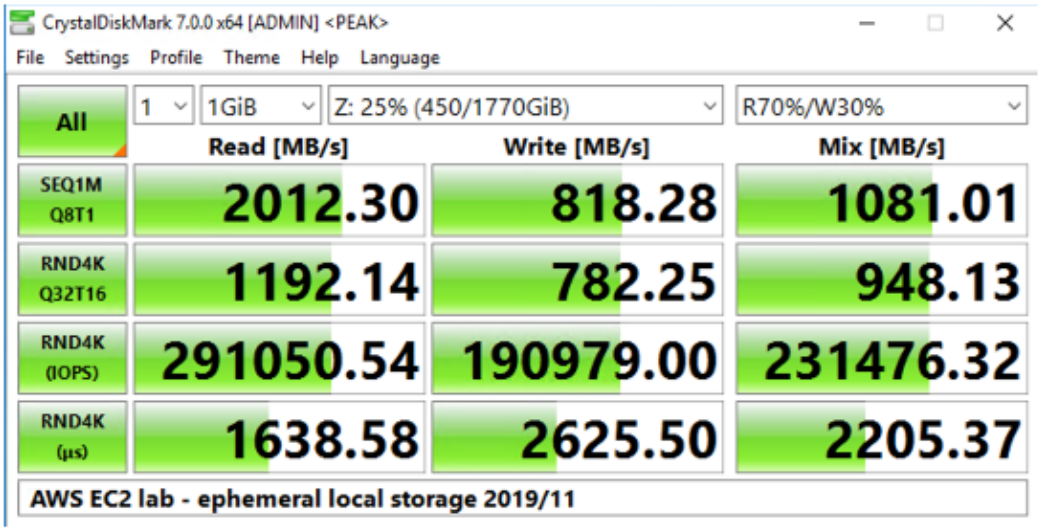


1 = number of tests (1 = quick test, 9 = many tests)

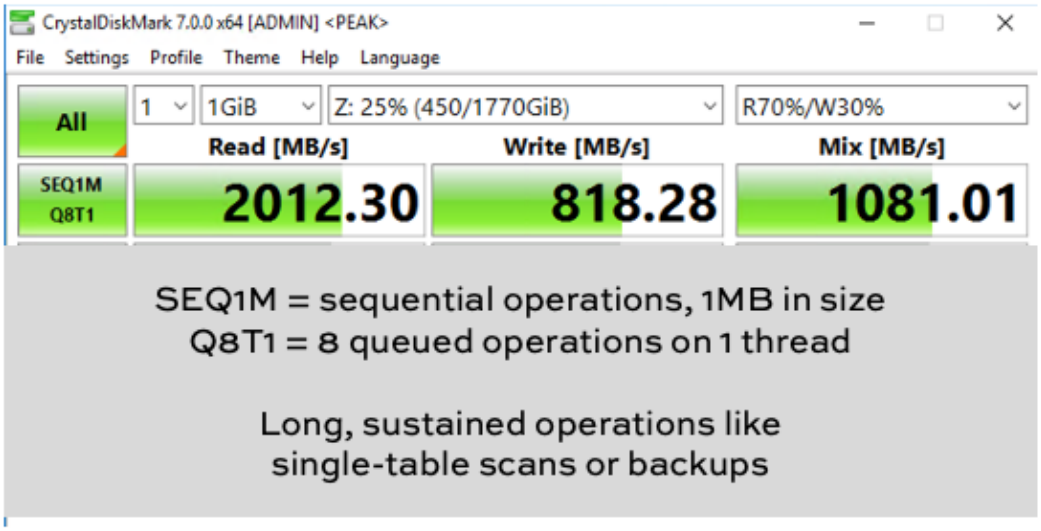
1GiB = test file size (1GiB = quick test, 64GiB = in-depth test that won't just get cached)

S: = the drive letter or folder that you want to test

Interpreting the results



Interpreting the results



SEQ1M = sequential operations, 1MB in size
Q8T1 = 8 queued operations on 1 thread

Long, sustained operations like
single-table scans or backups

1.3 p41

Interpreting the results

RND 4K= random operations, tiny 4KB size
Q32T16 = 32 queued operations on 16 threads: busy!

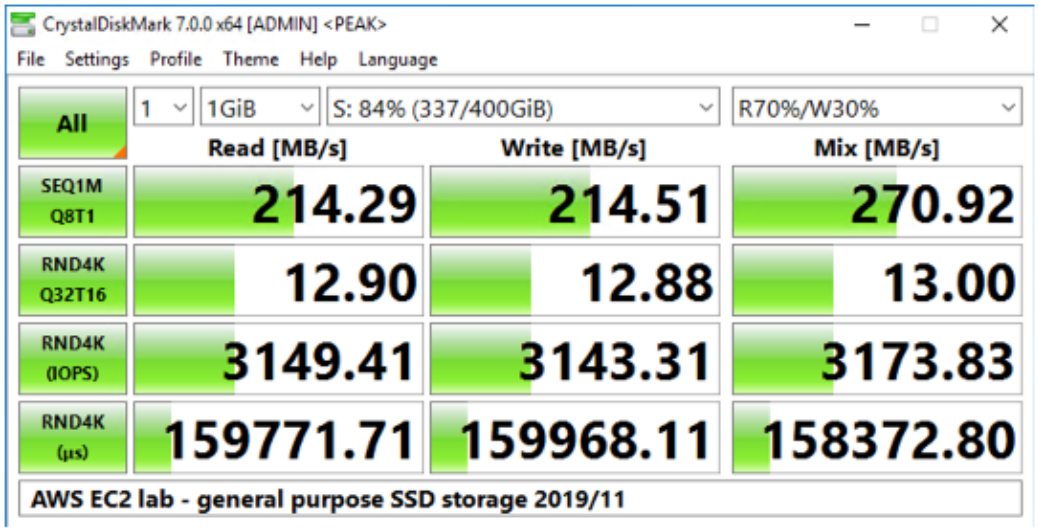
Kinda like an active, busy OLTP server or TempDB

RND4K Q32T16	1192.14	782.25	948.13
RND4K (IOPS)	291050.54	190979.00	231476.32
RND4K (µs)	1638.58	2625.50	2205.37

AWS EC2 lab - ephemeral local storage 2019/11

1.3 p42

Slower storage



1.3 p43

Click File, Copy, and paste into Notepad

```
-----------------------------------------------------------------------
CrystalDiskMark 7.0.0 x64 (C) 2007-2019 hiyohiyo
                           Crystal Dew World: https://crystalmark.info/
-----------------------------------------------------------------------
* MB/s = 1,000,000 bytes/s [SATA/600 = 600,000,000 bytes/s]
* KB = 1000 bytes, KiB = 1024 bytes

[Read]
Sequential 1MiB (Q=  8, T= 1):   214.294 MB/s [    204.4 IOPS] < 38947.38 us>
   Random 4KiB (Q= 32, T=16):    12.900 MB/s [   3149.4 IOPS] <159771.71 us>

[Write]
Sequential 1MiB (Q=  8, T= 1):   214.515 MB/s [    204.6 IOPS] < 38829.04 us>
   Random 4KiB (Q= 32, T=16):    12.875 MB/s [   3143.3 IOPS] <159968.11 us>

[Mix] Read 70%/Write 30%
Sequential 1MiB (Q=  8, T= 1):   270.915 MB/s [    258.4 IOPS] < 30766.38 us>
   Random 4KiB (Q= 32, T=16):    13.000 MB/s [   3173.8 IOPS] <158372.80 us>

Profile: Peak
   Test: 1 GiB (x1) [Interval: 5 sec] <DefaultAffinity=DISABLED>
   Date: 2019/11/23 19:34:10
     OS: Windows Server 2016 Datacenter (Full installation) [10.0 Build 14393] (x64)
Comment: AWS EC2 lab - general purpose SSD storage 2019/11
```

1.3 p44

If the users want it in 2 seconds...

How much data do we need to read?	21,600 MB
How long will the query take?	2 seconds
Divide the two to get throughput required:	10,800 MB per second

1.3 p45

This explains our performance

How much data do we need to read?	21,600 MB
But if the query is taking...	11 seconds
Then our throughput is only around...	1,963 MB per second
If the users want it to finish in...	5 seconds
Then our storage needs to read this fast...	21,600 MB / 5 seconds **4,320 MB/sec**

1.3 p46

This simple formula drives everything.

1.3 p47

Data you have to read

/ How fast you read

Minimum query runtime

1.3 p48

Red herrings

Buffer pool extensions (until maybe SQL Server 2019)

In-memory OLTP (Hekaton)

SQL 2012's non-clustered columnstore indexes

SQL 2014's clustered columnstore indexes for OLTP
(For more on columnstore: columnstore.net)

Fast disks connected over shared network storage

1.3 p49

Simple SAN path (incl. cloud)

1.3 p50

Path Speeds

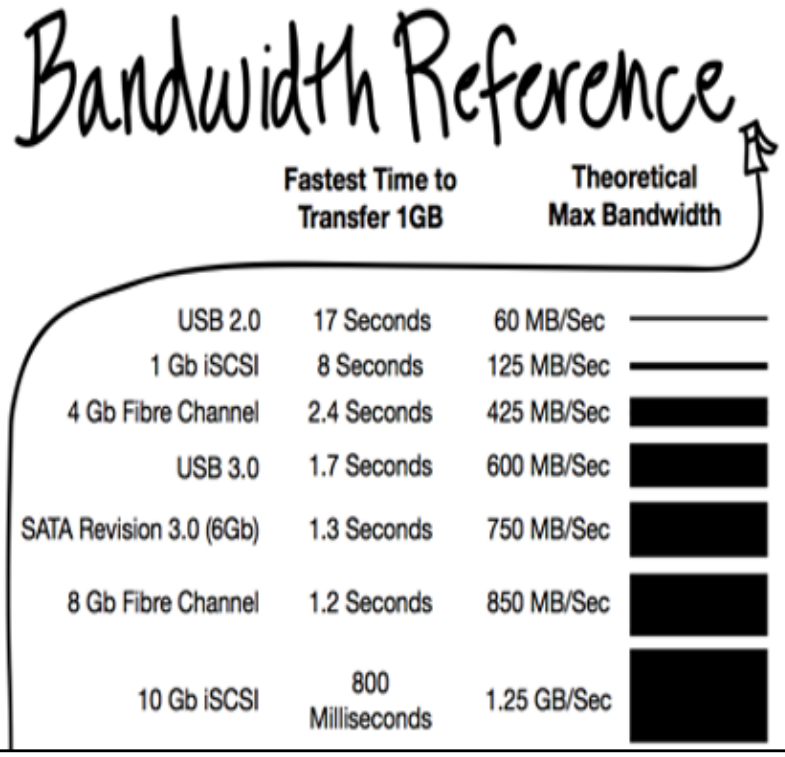
1Gb iSCSI

2Gb Fiber Channel

4Gb Fiber Channel

8Gb Fiber Channel

10Gb iSCSI

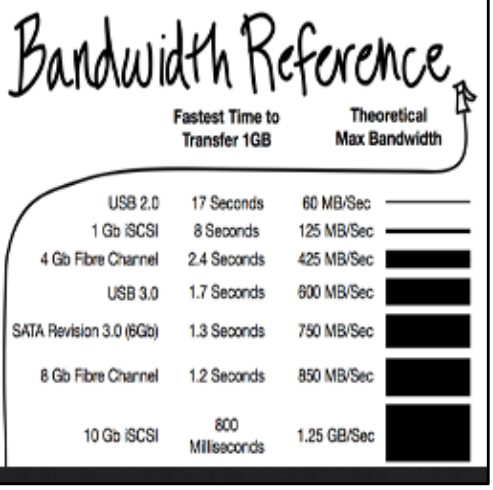


SQL Server handles more than you think

Maximum Consumption Rate (MCR):
How fast each core can consume data from storage

Starting point: 200MB/sec
of sequential reads per core

- 2 quad cores: 1,600MB/sec
- 4 quad cores: 3,200MB/sec
- 4 ten cores: 8,000MB/sec



But your VMs get a lot less

Because we pack a lot of clowns in this car.

1.3 p53

How to reduce PAGEIOLATCH

1. Tune indexes, looking at missing indexes:
   `sp_BlitzIndex @GetAllDatabases = 1`
2. Tune queries:
   `sp_BlitzCache @SortOrder = 'reads'`
3. Add more memory
4. Make storage faster

Setting up for the lab

1. Restart the SQL Server service (clears stats)
2. Restore your StackOverflow database
3. Copy & run the setup script:
   BrentOzar.com/go/serverlab1
4. Start SQLQueryStress:
   1. File Explorer, D:\Labs, run SQLQueryStress.exe
   2. Click File, Open, D:\Labs\ServerLab1.json
   3. Click Go

1.3 p55