Mismatched SLO

Last updated by | Peter Hewitt | Mar 6, 2023 at 6:56 AM PST

Mismatched SLO

Contents

Issue:

The issue occurs due to combination of a heavy write workload on the Primary and an imbalance between the service objective of Geo-Primary and Geo-Secondary (for eg., Primary P11 and Secondary P2) , where Geo-Secondary unable to keep up, and result in slowness Or potential unavailability.

Configuring Secondary to handle

Both primary and secondary databases are required to have the same service tier. It is also strongly recommended that the secondary database is created with the same backup storage redundancy and compute size (DTUs or vCores) as the primary. If the primary database is experiencing a heavy write workload, a secondary with lower compute size may not be able to keep up with it. That will cause redo lag on the secondary, and potential unavailability of the secondary. To mitigate these risks, active geo-replication will throttle the primary's transaction log rate if necessary to allow its secondaries to catch up.

Another consequence of an imbalanced secondary configuration is that after failover, application performance may suffer due to insufficient compute capacity of the new primary. In that case, it will be necessary to scale up database service objective to the necessary level, which may take significant time and compute resources, and will require a high availability failover at the end of the scale up process.

If you decide to create the secondary with lower compute size, the log IO percentage chart in Azure portal provides a good way to estimate the minimal compute size of the secondary that is required to sustain the replication load. For example, if your primary database is P6 (1000 DTU) and its log write percent is 50%, the secondary needs to be at least P4 (500 DTU). To retrieve historical log IO data, use the sys.resource_stats view. To retrieve recent log write data with higher granularity that better reflects short-term spikes in log rate, use sys.dm_db_resource_stats view.

How to check Mismatch SLO

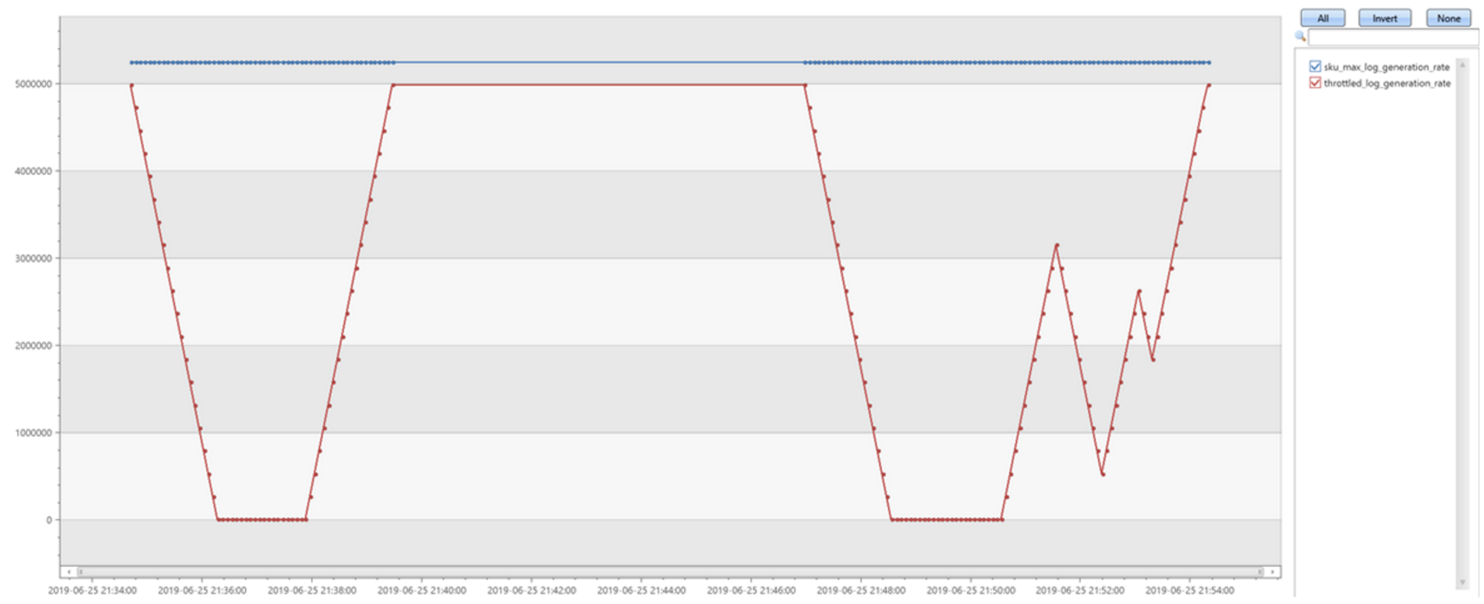
Kusto:

You can execute below query on geo-primary to check if log generation is being throttled

```
MonFabricThrottle
| where AppName == "<dbappname>" and database_name =~ "dbname"
| where event == "hadr_db_log_throttle" and throttling_reason == "MismatchedSlo"
| where TIMESTAMP > ago(1h)
```

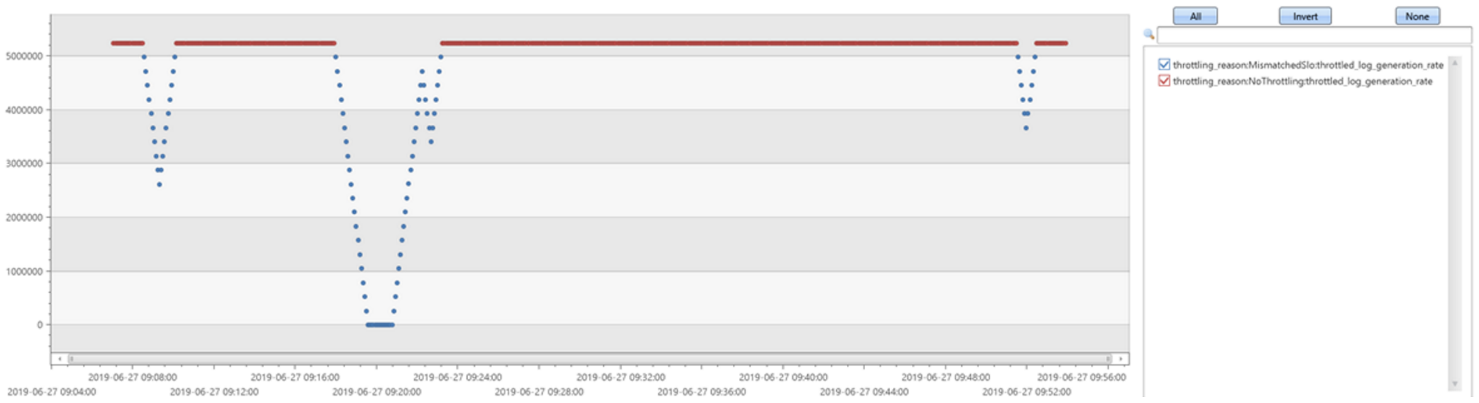
On geo-primary to check the history of throttling on log generation rate due to Mismatched SLO:

```
MonFabricThrottle
| where AppName == "<dbappname>" and database_name =~ "dbname"
| where event == "hadr_db_log_throttle" and throttling_reason == "MismatchedSlo"
| where TIMESTAMP > ago(1h)
| extend throttled_log_generation_rate=iff(throttled_log_generation_rate == -1, sku_max_log_generation_rate, t
| project TIMESTAMP, throttled_log_generation_rate, sku_max_log_generation_rate
| render timechart
```



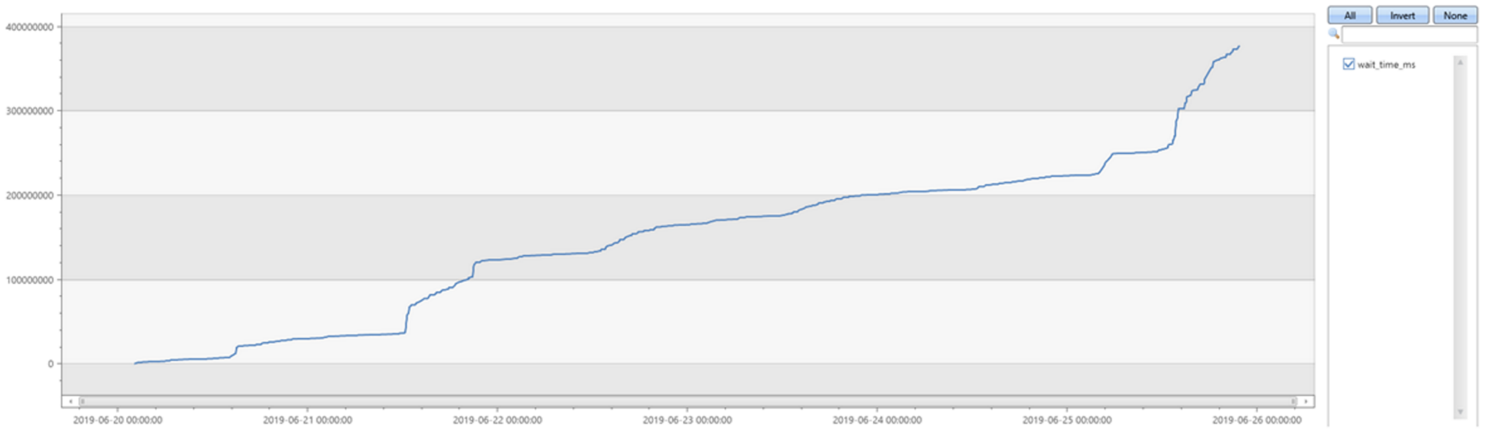
Check if there are multiple reasons for throttling:

```
MonFabricThrottle
| where AppName == "bd7ffc0e23ae" and database_name =~ "6dbdb391-92e6-4491-af1c-a6de87c23b58"
| where event == "hadr_db_log_throttle"// and throttling_reason == "MismatchedSlo"
| where TIMESTAMP > ago(1h)
| extend throttled_log_generation_rate=iff(throttled_log_generation_rate == -1, sku_max_log_generation_rate, t
| project TIMESTAMP, throttled_log_generation_rate, throttling_reason
| render scatterchart
```



Wait stats due to mismatched SLO throttling:

MonDmCloudDatabaseWaitStats | where AppName == "bd7ffc0e23ae" and database_name =~ "AnaproVendas" | where wait_type contains "HADR_THROTTLE_LOG_RATE_MISMATCHED_SLO" | project TIMESTAMP, wait_time_ms | render timechart



Other issue that could cause this mismatched SLO

In some cases you could see in the Kusto Table MonSQLSystemHealth the error messages: NotifyLogPoolMgr in DB: xyz shrinks logpool under memory pressure for xyz1 times with decrease percentage: x. This means Log if logpool gets shrunk because of memory pressure due to a heavy log in the primary or secondary is not redoing in a timely maner the changes.

```
MonSQLSystemHealth
    | where PreciseTimeStamp between(datetime('2023-02-02 08:00:00')..datetime('2023-02-02 09:00:26'))
    | where LogicalServerName =~ 'servername'
    | where message !contains 'Backup'
    | where message !contains 'FSTR:'
    | where message !contains 'UpdateHadronTruncationLsn'
    | where message !contains 'VersionCleaner'
    | where message !contains '[INFO]'
    | project TIMESTAMP, message
```

From our customer side our customer could see the following error message: During redoing of a logged operation in database 'xzy-08d8-4bca-8c3e-0b35e9a4c4c7' (page (0:0) if any), an error occurred at log record ID (20724:13040:1). Typically, the specific failure is previously logged as an error in the operating system error log. Restore the database from a full backup, or repair the database.

From customer end they can also check using DMVsr :

Transaction log rate throttling on the primary due to lower compute size on a secondary is reported using the HADR_THROTTLE_LOG_RATE_MISMATCHED_SLO wait type, visible in the sys.dm_exec_requests and sys.dm_os_wait_stats database views.

Mitigation

This behavior is on by default for all databases and it will be the expected behavior going forward. If the customer is using this mismatched SLO configuration in production, then they should know that this is not a recommended configuration.

You can find more info in the below link and an example RCA to provide to the customer.

Note : If customer have a scenario where geo secondary databases developed high redo queue, which needs to be cleared up before update slo on a database can continue. Please advise customer to always scale up geo secondary first and after that scale geo primary.

RCA Template

We have noticed that you have GeoReplication setup from Server/DatabaseName (Region1) -> Server/DatabaseName (Region2). However the service objective for both databases is different. The Primary is <SLO here> and the secondary is <secondary SLO>.

Due to the combination of a heavy write workload on the Primary and this imbalance between the service objective or the GeoPrimary and GeoSecondary, the GeoSecondary is unable to keep up and falling behind. Over time, the replication lag between the GeoPrimary and GeoSecondary will increase, thereby increasing the risk of substantial data loss during a DR failover.

Therefore, we strongly recommend upgrading your GeoSecondary to the same service-objective as the GeoPrimary. The configurable SLO on GeoPrimary and GeoSecondary is more suited for customers who have a higher read workload, but their write load fits into the GeoSecondary's service objective.

Please refer to the guidance on [Configurable performance level of the secondary database]

How good have you found this content?