SOLVING FOR NOISY NEIGHBOUR PROBLEMS IN DATABASES USING AWS

REBURA
A Tenth Revolution Group Company

A noisy neighbour is a tenant whose activity negatively affects the performance and resources available to other tenants in a SaaS environment. The root cause of this issue is shared components—be that compute, storage, licenses, etc. Depending on the architecture model (silo, pool or bridged), the risk of noisy neighbours changes. In a truly siloed model there is no risk as each tenant is given a dedicated environment with no (direct or indirect) dependence on other tenants. Shared environments pose the greatest risk of noisy neighbours as resources are shared between tenants, making the performance for one tenant directly dependent on the activity of the other tenants.

Noisy neighbours are a broad topic in SaaS, relevant to just about every layer of a service's stack. This article focuses on how the problem manifests itself in databases and what solutions can be approached.

Databases here refer to relational and non-relational databases. In particular, this article will look at RDS-managed databases (e.g., RDS for PostgreSQL) and DynamoDB.

Four approaches are presented:

1. Absorb
2. Limit and Throttle
3. Optimise
4. Offload

In the first 3 we assume that a completely shared (pooled) database is implemented. Given this assumption, the root cause of a noisy neighbour becomes the activity of the noisy tenant combined with limited resources. The 4th incorporates a bridged architectural pattern.

Each approach has many designs and strategies for its implementation. AWS-centric strategies are mentioned but the approaches are agnostic of any service provider.

1. ABSORB

It was mentioned previously that part of the cause of a noisy neighbour is limited resources. This is true insofar as a resource (or its capacity) is static. A common method of handling a system under variable load is automated scaling—effectively removing capacity limits (to a certain extent).

Increasing system capacity under increased load can be thought of as the system absorbing the effect of the noisy tenant.

Database performance can scale with increased load along 2 dimensions—vertical and horizontal. Put simply, vertical scaling means upgrading hardware (increasing CPU, memory) and horizontal means adding more servers/instances.

Amazon RDS provides a few avenues for scaling. For read-heavy workloads, read replicas are the traditional approach—an example of horizontal scaling. Vertical scaling is achieved by opting for instances with higher specifications. Aurora Serverless V2 is a more dynamic approach to scaling along the vertical and horizontal dimensions, with the benefit of reduced operational overhead.

DynamoDB offers autoscaling functionality through scaling policies on tables and global secondary indexes. Write and read capacity can be scaled differently and provisioned capacity is supported.

Scaling resources when a tenant imposes disproportionate load on the system has the advantage that tenants (including the offender) are none-the-wiser, however, it means that the SaaS provider generally absorbs the cost as well. It is also true that resources cannot scale infinitely in the real world and limits (be those cost, performance, capacity) will inevitably be reached.

SOLVING FOR NOISY NEIGHBOUR PROBLEMS IN DATABASES USING AWS

REBURA
A Tenth Revolution Group Company

2. LIMIT AND THROTTLE

Another component to the cause of noisy neighbours is the activity of a noisy tenant. Without absorbing the load by auto-scaling, our attention can be placed on addressing the activity of the tenant.

Limits can be imposed on tenants through usage quotas or rate limits. This is commonly done at the API level. Amazon API Gateway provides the usage plan feature which allows SaaS providers to place rate and quota limitations on requests. These are configurable per tenant, enabling limits on a per-tenant basis. As this feature sits at the API level, the SaaS provider must determine some relationship between API usage and database load. The data needed for this is available through CloudWatch metrics, database log streams and RDS Performance Insights. AWS X-Ray can be implemented to provide a coherent view of an API request through to database query.

Limits can be related to tenant tiers in the SaaS pricing model, giving tenants visibility of their allowances and the SaaS provider a way of scaling platform costs with revenue.

3. OPTIMISE

Up until now we have taken 2 components (tenant activity and limited resources) of the root cause of noisy neighbours and alternately held one constant and solved for the other. This section presents an approach to noisy neighbour which addresses neither but can help a SaaS provider minimise the likelihood of a noisy neighbour event occurring. This involves any way of optimising the existing system. Generally this means optimising the data model, or the code which performs queries on the database.

A good practice in any case is to optimise database operations with respect to the limiting factors - commonly CPU and memory. This can include restructuring of data, indexing, or query optimisation. Query optimisation involves considerable monitoring on the database. Amazon RDS Performance Insights enables a SaaS provider to assess database load, allowing the SaaS provider to identify resource-intensive queries. These queries can then be investigated further and appropriately optimised. Global secondary indexes in DynamoDB can be an effective way of reducing heavier scan operations.

Caching is an approach with a similar effect to query optimisation when applied to resourceintensive queries. This is most appropriate when queries to the database are repeated often. Caching can often be performed at a higher level (e.g. REST API) than the database as well, eliminating any load on the database for certain access patterns.

4. OFFLOAD

The final approach in this article follows a similar pattern to the first in that the noisy tenant's load is absorbed by the SaaS platform. However, this is not achieved by a traditional scaling technique. Instead the architecture of the SaaS platform changes from a pooled to a bridged model.

The noisy tenant is offloaded to a dedicated database instance, thereby relieving the shared database of the load. Similar to the auto-scaling approach, this has a large cost implication and is best suited to a tiered pricing model, which takes into account the increased operating costs of a dedicated database.

CONCLUSION

Noisy neighbour problems will always be a concern when operating a shared, multi-tenant environment. There is no one-size-fits-all solution, and SaaS providers must judge their design and approach not only on a technical ground, but also commercial. The strategies discussed here have ramifications for pricing models, quality of service, operating cost and technical complexity.