Performance antipatterns for cloud applications

06/05/2017 • 2 minutes to read • Contributors

A *performance antipattern* is a common practice that is likely to cause scalability problems when an application is under pressure.

Here is a common scenario: An application behaves well during performance testing. It's released to production, and begins to handle real workloads. At that point, it starts to perform poorly—rejecting user requests, stalling, or throwing exceptions. The development team is then faced with two questions:

- Why didn't this behavior show up during testing?
- How do we fix it?

The answer to the first question is straightforward. It's difficult to simulate real users in a test environment, along with their behavior patterns and the volumes of work they might perform. The only completely sure way to understand how a system behaves under load is to observe it in production. To be clear, we aren't suggesting that you should skip performance testing. Performance tests are crucial for getting baseline performance metrics. But you must be prepared to observe and correct performance issues when they arise in the live system.

The answer to the second question, how to fix the problem, is less straightforward. Any number of factors might contribute, and sometimes the problem only manifests under certain circumstances. Instrumentation and logging are key to finding the root cause, but you also have to know what to look for.

Based on our engagements with Microsoft Azure customers, we've identified some of the most common performance issues that customers see in production. For each antipattern, we describe why the antipattern typically occurs, symptoms of the antipattern, and techniques for resolving the problem. We also provide sample code that illustrates both the antipattern and a suggested solution.

Some of these antipatterns may seem obvious when you read the descriptions, but they occur more often than you might think. Sometimes an application inherits a design that worked on-premises, but doesn't scale in the cloud. Or an application might start with a very clean design, but as new features are added, one or more of these antipatterns creeps in. Regardless, this guide will help you to identify and fix these antipatterns.

Here is the list of the antipatterns that we've identified:

Antipattern	Description
Busy Database	Offloading too much processing to a data store.
Busy Front End	Moving resource-intensive tasks onto background threads.
Chatty I/O	Continually sending many small network requests.
Extraneous Fetching	Retrieving more data than is needed, resulting in unnecessary I/O.
Improper Instantiation	Repeatedly creating and destroying objects that are designed to be shared and reused.
Monolithic Persistence	Using the same data store for data with very different usage patterns.
No Caching	Failing to cache data.

Antipattern	Description
Synchronous I/O	Blocking the calling thread while I/O completes.