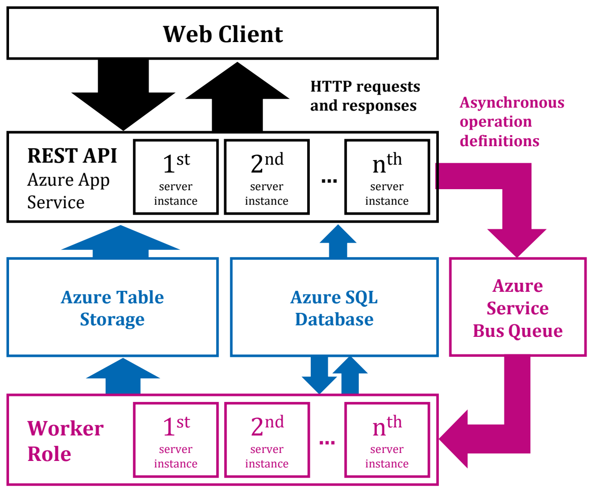
Architectural Tactics for the Design of Efficient PaaS Cloud Applications

David Gešvindr Barbora Bühnová Tomáš Pitner

Introduction

Application deployment in the cloud is not itself a guarantee of high performance, availability, and other quality attributes, which may come as a surprise to many software engineers who detract from the importance of proper architecture design of a cloud application. An operation of applications designed for on-premises environment in a PaaS cloud is costly and inefficient as they often rely on expensive and poorly scalable services (eg. commonly used relational databases).

Efficient PaaS cloud applications require specific software architecture design, which combines high variety of PaaS cloud services, not commonly used in an on-premises environment.



Software architecture of a PaaS cloud application which combines multiple PaaS cloud services to provide high scalability of both read and write operations using **Materialized View** and **Asynchronous Messaging** tactics.

Problem Definition

When designing a software architecture, it is a common good practice to use existing design patterns, that are however not specifically designed for cloud applications so they do not take into account a rich set of services and features of the PaaS cloud.

Objectives

Our objective is to examine the impact of current state of-the-art architectural tactics and design patterns available for on premise applications on PaaS cloud applications.

Based on the measurable impacts we will create a list of identified tactics, design patterns and their combinations advisable for the design of PaaS cloud applications.

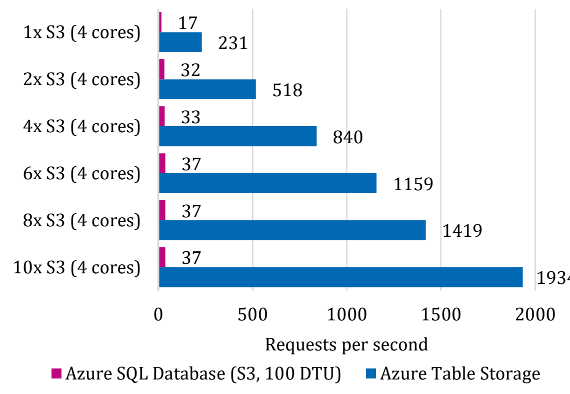
Storage and Data Access Tactics

In the PaaS cloud there is no single storage service that outperforms others in terms of high scalability, low costs and complex querying and integrity enforcement support.

Materialized View

Prepopulated views over data from primary data storage are generated to highly scalable, inexpensive storage services (usually NoSQL).

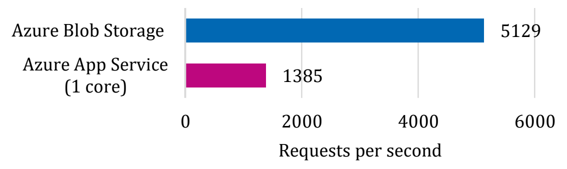
- Data is always written first to a primary storage which guarantees data integrity.
- After modification of primary data copy, derived copies of data must be updated.
- Majority of client read requests is served by highly scalable storage services where are stored additional copies of data in a form exactly matching client needs so that data can be retrieved by a single query without any need for an additional processing by the application server.



Comparison of read throughput with and without materialized view

Static Content Hosting

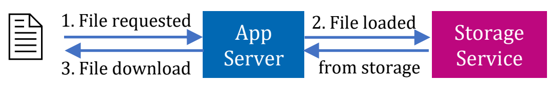
Static content should be distributed using specialized storage services which scale automatically with user demands and not by more expensive application servers with reserved performance.



Azure Blob Storage:	$0.024 per GB/month + $0.00036 per 10,000 transactions
Azure App Service:	$55.80/month (includes max 10 GB storage)

Valet Key Tactic

To distribute secured static content a user needs to be authenticated and authorized by the application which loads and transfers the file using server resources.



Valet Key Tactic uses a short-term access token generated by the storage service and passed to the client by the application server so that the file can be downloaded directly from the storage.

Messaging and Data Processing Tactics

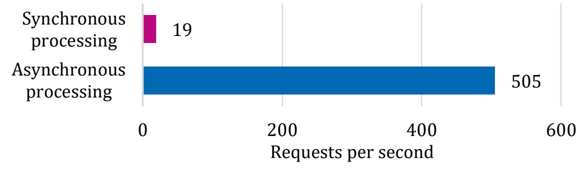
Web applications usually handle requests synchronously, which limits scalability but simplifies SW architecture. Alternate approach is to take an advantage of asynchronous request processing.

Asynchronous Messaging

Requests received by the application server are not immediately processed, instead of that they are stored in a highly scalable queue service and confirmed to clients, therefore especially long running operations are not terminated due to client time-out.

Independently hosted worker processes continuously load messages from the queue and process them.

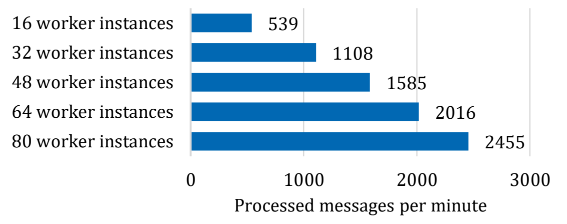
As the client does not receive operation output as a part of the response, is advisable to establish additional notification channel.



Comparison of write throughput of a REST API with applied materialized view tactic, which significantly increases complexity of data modifications due to its requirement to update derived copies of data. Use of asynchronous processing for data updates is strongly recommended.

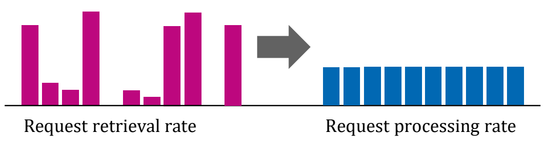
Competing Consumers

Multiple consumers (workers) can load messages simultaneously from the queue which increases scalability and elasticity as the number of workers can quickly change.



Load Leveling

When requests are received irregularly in bursts, highly scalable queue service is capable to persistently store them so that workers can process them at a constant rate.



References:
GEŠVINDR, David and Barbora BÜHNOVÁ. Architectural Tactics for the Design of Efficient PaaS Cloud Applications. In 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA). IEEE, 2016.