System Design Document

CosmosDB RU Cost Calculation Improvement - Efficiency & Time

Version 2.0

January 19, 2019

Team Members: Joe Do, Ferdinand Tembo, Kevin Tran

https://docs.google.com/document/d/10_VA-ZX1XHbgyOCLrGOz1rzl38PAa6kZ7_1C6AFyq_s/edit?usp=sharing

Bellevue College - Computer Science Department

Alfred Nehme

Revisions Page

Version	Primary Author	Description of Version	Date Completed
1.0	Joe Do Ferdinand Tembo Kevin Tran	First Draft - Interpretation	11/07/2018
2.0	Joe Do Ferdinand Tembo Kevin Tran	Second Draft - Rough Interpretation	01/19/2019

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to describe the implementation of the Request Units (RU) calculator described in the software requirement specification. The RU calculator is designed to capture Azure Cosmos DB transactional throughput.

1.2 Scope

The main objective of the RU calculator is to calculate how many RUs are needed for the consumers. Azure Cosmos DB customer will provide a JSON document and the number of READ/CREATE/UPDATE/DELETE operations. Our primary scope is to calculate the required RU based on those inputs.

1.3 Definitions, acronyms, and abbreviations

Abbreviations	Description
CUD	Create, update, delete
CRUD	Create, read, update, delete
RU	Request Units
JSON	(JavaScript Object Notation) is a lightweight data-interchange format

Cosmos DB	Cosmos DB is Microsoft's proprietary globally-distributed, multi-model database service "for managing data at planet-scale"
GUI	Graphical User Interface
Indexing Policy	A structural rule in a database that strategizes in optimizing database commands and the database resources (CPU usage, I/O, ram) that they use.

1.4 References

[1] "Azure Cosmos DB Pricing." *Pricing - Azure Cosmos DB | Microsoft Azure*, azure.microsoft.com/en-us/pricing/details/cosmos-db/.

[2] "Azure Cosmos DB: NoSQL Capabilities Everyone Should Know About." *Microsoft Azure Cloud Computing Platform & Services*, azure.microsoft.com/en-us/resources/videos/build-2017-azure-cosmos-db-nosql-capabilities-everyone-should-know-about/.

[3] "NoSQL Databases." *Basho*, basho.com/resources/nosql-databases/.

[4] Howell, Jason W, et al. "Indexing in Azure Cosmos DB." Microsoft Azure, Microsoft, 9 Nov. 2018, docs.microsoft.com/en-us/azure/cosmos-db/index-overview.

2. System Overview

In a client-server architecture, we have Azure Cosmos DB as the server side and a web client as the client side. The web client will take in user inputs including one or more JSON files and the number of database operations. The web client then makes a request to Azure Cosmos DB. Based on the response header [x-ms-request-charge] received from Azure Cosmos DB, the web client will be able to display the appropriate Request Units to the end user(s).

In the server side, Azure Cosmos DB stores documents based on the destination database and collection. Users can access their database(s) and collection(s) if they have created it or if they have been invited to the database account. Database accounts have permissions ranges of limited permissions and full ownership permission. Further backend information about that architecture is limited.

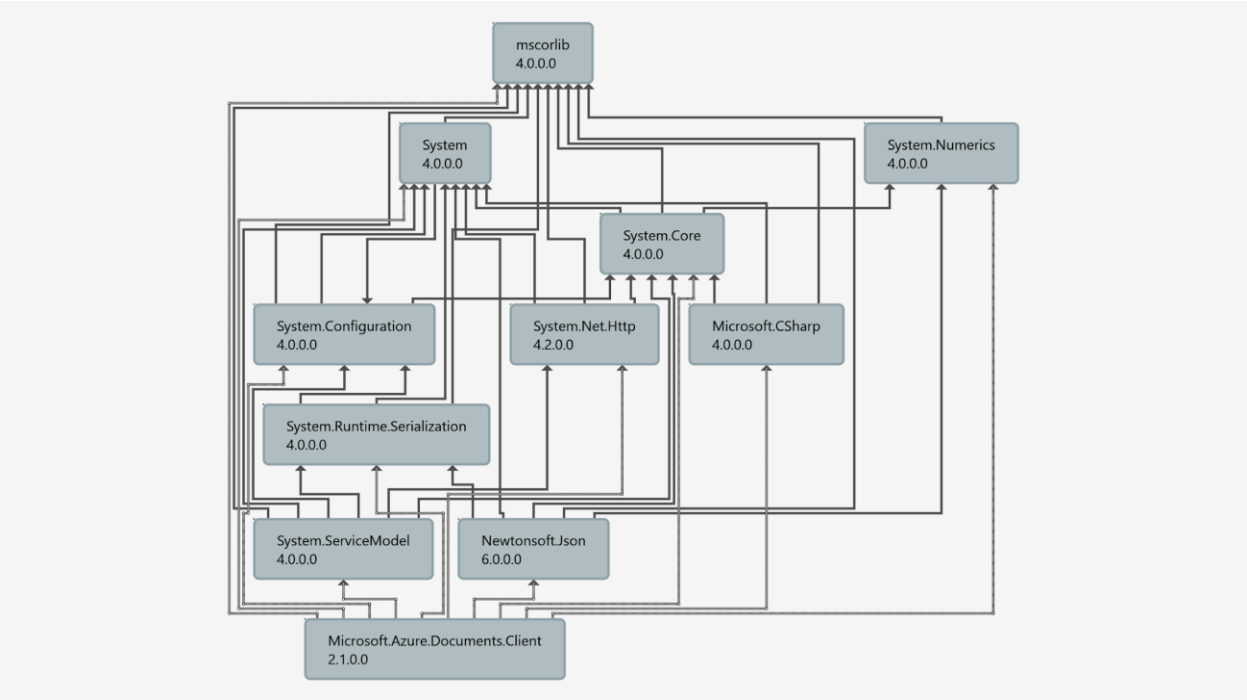
3. System Software Architecture

Please note that this section will not stipulate the final and actual product that we will be working on. We are currently on the testing phase to identify product improvements. The System Software Architecture that is currently used in testing and problem investigation will be stipulated only in this section at this time.

3.1 Decomposition Description

The current decomposition for our testing software is that a Microsoft Azure Document Client package and the Newtonsoft JSON package is attached to a C# project that contains references to the .NET library. Whenever we test and create a query request, it utilizes all of the packages in order to provide us with the request unit amount after it has been processed.

3.2 Dependency Description



The Microsoft Azure Client requires System-related C# packages and a Newtonsoft JSON package in order for it to run correctly in our testing custom client. The website is currently unknown.

3.3 Interface Description

There are no human or physical interfaces in any part of our system. We only have module interfaces at this time. In terms of testing on the website, we have a web browser and various web page services from the RU estimator website that would provide us with intuitive secondary RU-amount checking.

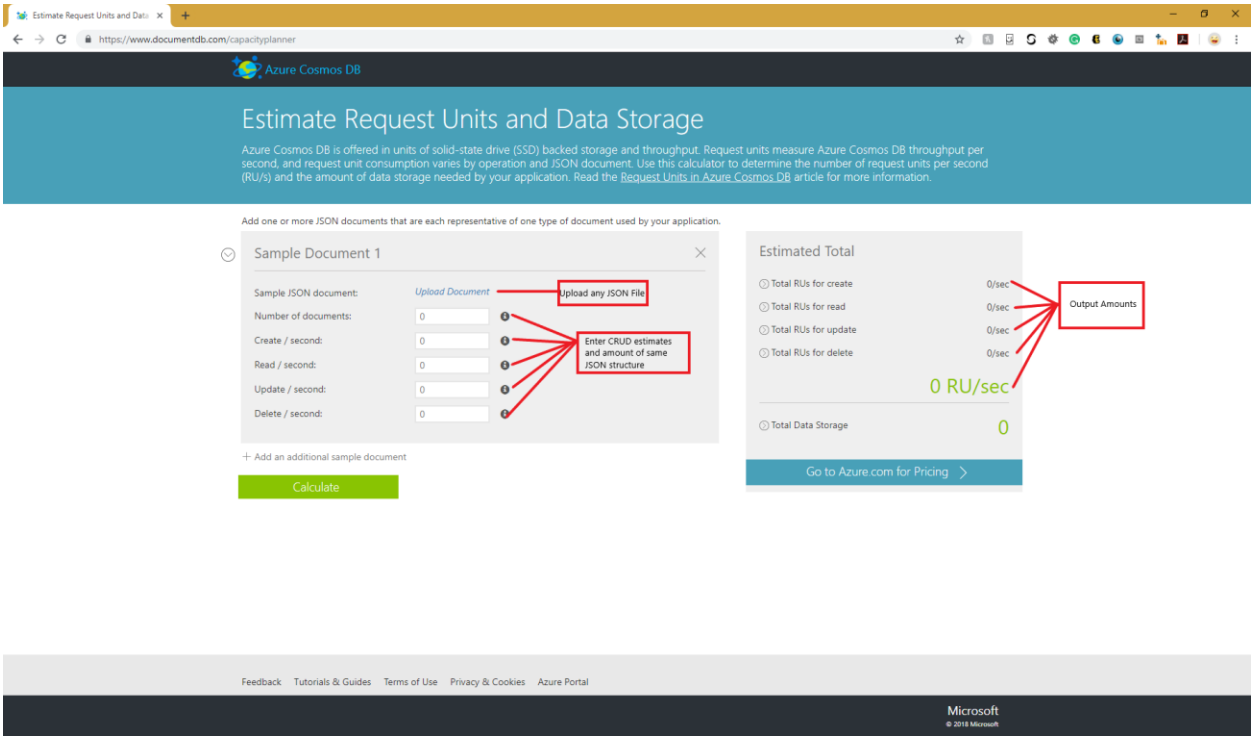
3.4 Module Interfaces

At this time the testing JSON files of any design/complexity will work as test cases within our system. That is created by a separate software package system. There are software interfaces that exist between CosmosDB and us developers/users within the testing phase. Those are .NET and the Azure CosmosDB API. The CosmosDB API utilizes some classes and functions available in .NET in order to build a meaningful connection between the

CosmosDB service in the cloud and to our computers where we give it commands and receive output from CosmosDB. We do not know of the specificity of the module interfaces implemented in the RU-amount website.

3.5 User Interfaces (GUI)

There is currently no GUI for testing from the console but there is a GUI that is used in the RU-calculator website.



We predict that the final and actual product that we will work on will involve a GUI that is already implemented.

4. Detailed Design

4.1 Module Detailed Design

In our second part of the investigation phase of the project, we are comparing RU output to collection index policy and outputs of stored procedures with various argument inputs.

For collection index policy, we tested against a controlled collection (automatic dynamic indexing), which organizes the indexes of uploaded documents synchronously per every change (CUD of CRUD).

Then there is a non-indexing mode where no index is created on the uploaded documents. Thus, all document items will be stored/accessed index-less in a key-value type of behavior based on a trait like its ID.

4.2 Data Detailed Design

The types of JSON data used in our tests will be of the two as follows:

Family JSON Test Document		
	_childrenSize	Int size
(Ideally) Primary Key	ID	String id
	_lastName	
	_family	(Array input of below)
	_family.Parents	(Array of two parents)
	_family.Parents.FirstName	String
	_family.Parents.LastName	String
	Children	Array of children associated to parents
	Children.FamilyName	String (similar to last name)
	Children.FirstName	String
	Children.Gender	String of an ‘m’ or ‘f’ character
	Children.Grade	double
	Children.Pets	Pets array
	Children.Pets.GivenNam	String

	e	

Package JSON Test Layout		
(Ideally) Primary Key	Id	String
	Records	Array of n size in the below format
	Records.ReceiverLastName	String
	Records.ShippingAddress	String
	Records.TrackingNumber	Int
	Records.NumberOfItems	Int
	Records.SenderFirstName	String
	Records._SenderLastName	String
	Records._SenderAddress	String

4.3 RTM

Yellow = not applicable at this time.

Req.#	Requirement	Design specification	Program Module	Test Specification	Test Case(s) Number	Successful test Verification	Modification of Requirement	Remarks
3.1	External Interfaces	Utilize Azure Cosmos DB NuGet library		Connect front-end to Azure	1	Verified	N/A	For testing phase, install is only required
3.2	Functional Requirements							
3.3	Performance Requirements	(Perceive		(Perce				

		d) Improve RU output accuracy		ived) Run tests between old and new engine.				
3.4	Logical Database Requirements	Documents have to have a unique trait like ID, etc.						
3.5	Design Constraints	UDF: Has to be defined as a string Stored Procedure cannot be duplicated by its ID; Also can only pass in one parameter on the server-side						
3.6	Software System Attributes							