Section 3: Week 7: Cloud Migration via DSL

Nate Bachmeier

TIM-8110: Programming Languages and Algorithms

May 19, 2019

North Central University

# Cloud Migration via DSL Outline

Contoso is a provider of online student analytic services with a vast collection of micro services hosted in their private data center. One of the tenants of micro service design states that each component should ‘share-nothing’ including data stores. This reduces the blast radius and improves the resiliency of the over-all system.

They need a mechanism to efficiently transition their product lines from private data centers into the public cloud. These heterogenous private stores can introduce challenges for that migration as they need to become hydrated. While significant literature exists for trivial ‘lift and shift’ paradigms of an individual store there is less research on rehydrating the data network.

# Background

Many businesses like Contoso are actively working to transition their proprietary systems into the public cloud. This enables them to reduce infrastructure costs and improve the agility to provide new features to their customer base. These capabilities are well documented through highly optimized ‘pay per use’ pricing model and instant access to a virtually unlimited amount of resources.

For many existing service providers, the journey to the cloud can be complex as it requires moving their proprietary systems into new environments. There are two supersets of issues (1) getting the infrastructure to the cloud; and (2) migrating the customer data into the new cloud stores.

Business stakeholders often take a conservative position during cloud migrations and be hesitant to filling the switch across all customers. This leads to scenarios such as creating a new deployment instance in the cloud and then hydrate specific customers history. Transitioning small handfuls of customers per data migration iteration reduces the blast radius and provides better assurances of business continuity in exchange for more time to complete the project (Ferdiana & Putra, 2018).

Due to the longevity of the migration project the system state can end up in an arbitrary flux. This can occur from new features being are added, and internal implementation details changing. The data migration process has two choices to mitigate this issue (a) create and maintain bulk load interface on each data store; or (b) replay the traffic through the steady state interface; or (c) a combination of both choices.

For systems with large number of heterogenous private data store and a high degree of data connectivity between them; it could be prohibitively expensive to (a) create bulk load interfaces. Consider the scenario where events cascade such as ‘service A’ emits N events by ‘service B,’ which in turn emits M events to ‘service C.’ To correctly model this would essentially duplicate the large amounts of business logic and be error prone.

Another argument for businesses to desire choice (b) is that improvements to the data migration scenario are improvements to the general customer. This can be seen in terms of general correctness, backward compatibility assurances, performance and scalability scenarios.

There are potentially specific micro services where scenario (a) is desired in combination with (b). For instance, a business might have archives of physics models each requiring days of compute time. They might choose to maintain an out of band process for moving the models while replaying the metadata through steady state ingestion pipeline.

Within the context of this proposal it is assumed that the business must choose option (b) or (c) due to option (a) being prohibitively expensive to build and maintain.

# Problem Statement

Cloud Migration Strategies tend to center around (1) getting the infrastructure; and (2) getting the data into the new environment. For enterprise customers with large numbers of micro services there are challenges to successfully accomplishing both.

Patterns around moving the infrastructure is a well-studied problem, thanks in part to virtualization and containerization technologies (Cefaratti & Lin, 2018) (Ahmad, Naveed, & Noda, 2018). A lesser studied scenario is the data migration and methods for overcoming their inherent challenges.

Consider the scenario where a bulk migration of 10 million records traverses a component with a 0.01% failure rate. This will reproduce 1,000 instances of that defect which can cause inaccurate results for the customer. If all issues within the product code can be resolved, there will still be issues due to the underlying system being built on commoditized hardware. An expectation of failure needs to exist within the data migration scenario.

Another challenge is optimizing the throughput that these migrations should be performed. If the ingestion rate is too slow it will never complete, yet too fast and it cause scalability issues within the infrastructure. Those scalability issues will increase the failure rate and result invalid results.

If the system was set to scale infinitely then it could become to costly to perform the data migration. This would lessen the competitive gain of moving to the cloud and place an unjust financial burden on the business. While that might be acceptable for a short period it has already been stated that the entire enterprise migration could last years.

# Goals

## Max Flow Model

These challenges could be restated as there is a need to find the ‘min-cost max-flow’ through the distributed architecture. There is also a need to model which junctions lead to bottlenecks versus are over provisioned. If ‘service X’ is bound by the scale of ‘service Y’ then either X needs to be reduced for cost savings or Y increased to provide additional load.

The primary goal of this study would be to identify the features which are most critical to the maximum flow model. Consider a component which can only process N records per second. In cloud environments the ability to add or remove virtually unlimited resources might rephrase this scenario into ‘N records per V virtual cores per M gigabytes of memory.’

## Min Error Rate

Another requirement is to maximize the flow of records while minimizing the number of failures. Writing error free distributed source code is an extremely complex problem. A domain specific language (DSL) could be created to describe the distributed compensation policy. This would ensure consistency of the rules and remediation behaviors.

If the system is capable of self-healing, then fewer concerns exist around availability which can lead to even higher net flow rates.

## Define New Patterns

Multi-casting data to several unreliable receivers is not a unique problem and is shared by several sciences. These challenges are well-studied in certain electrical and customer broadcast communication systems. A deeper investigation needs to exist to determine how those learnings can be applied to this use case.

# Relevance and Significance

Businesses of all shapes and sizes are actively migrating their workloads onto the public cloud. If they can determine bottlenecks in the hydration process, then it is possible to prioritize changes that lead to the maximum return. This can expedite their journey and the lead time before they begin reaping the benefits of that transition.

While this is the primary scenario that research is seeking to understand, there are other scenarios that would also benefit. For example, auto scaling legacy services can introduce its own set of challenges. Today businesses often rely on simple metrics such as CPU rates to signal an increase or decrease in service instances. This is inefficient as it relies on degradation quality as the notification mechanism for an individual service. Instead a holistic flow model could identify the increase in load and adjust multiple services to meet the change in traffic pattern.

Another shared area of all distributed micro services is reliably transmitting their notifications between each other. For many scenarios the cost of duplicating the message is significantly less than the cost of missing the event entirely. It would not be efficient to merely send three copies of the same traffic, however there are potentially encoding schemes that allow for the introduction of parity bits or auto remediation. Such self-healing solutions would increase the fault tolerance of the entire system and provide a more positive user experience to all consumers.

# Literature Review

There are four core aspects of literature that was reviewed (1) cloud migration challenges and strategies; (2) maximum flow and graph analysis; (3); reliable distributed systems; and (4) event patterns.

## Cloud Migration Challenges and Strategies

In *A Review of Cloud Migration Strategies in the Developing Countries*; Ferdiana and Putra performed an extensive survey of challenges that face cloud adoption. The issues facing developing countries can also be a proxy for small enterprise environments (<=1B$). This is due to their limit access to funding and resources. Two of the most frequent challenges identified through is (1) data migration issues; and (2) conservative business cultures.

In *Strategy and procedures for Migration to the Cloud Computing*; Ahmad, Naveed, and Hoda define the six R’s to migration as: Re-host, Re-platform, Re-purchase, Re-factor, Retain, and Retire. They continue with an investigation into the ‘state-of-art’ implementations of these ideas. There also define the levels of cloud maturity that businesses transcend on their journey. While significant attention is provided to the infrastructure, little is provided for complex data migrations.

In *Exploring Data Center Migration: A Case Study*; Cefaratti and Lin provide a postmortem to a successful enterprise migration. Their use case impacted 80,000 users as their workload transitioned from a private data center to public cloud. Thematically planning and effective communication were provided as primary drivers of success. A strong focus on infrastructure is also provided with little consideration for the complex data migration.

## Maximum Flow and Graph Algorithms

In *Efficient Maximum Flow Algorithms*; Goldberg and Tarjan provide a broad overview of scenarios that can be addressed with ‘max flow’ algorithms. They describe optimizations that exist in various concrete examples. Many of these examples resonated with the challenges of data hydration scenarios.

In *Modeling and Optimizing of Renewable-Energy Sharing*; Benda, Chu, Quek, and Buckley describe the challenges of harvesting and distributing power between a mesh of base stations. Their analysis showed that using a ‘min-cost max-flow’ over ‘max-flow’ resulted in a 30% savings. The problem structure is like the hydration and suggests it would be advantageous to look for a similar pattern.

In *A Study on Rapid Incremental Max Flow in Dynamic Networks*; Wang and Ling provide algorithms on top of classical ‘max-flow’ problems which enables them to operate on dynamic networks. These capabilities would enable an auto scale controller to adapt changes more quickly even as the number of dimensions increased. In the context of the migration scenario that would also mean the optimization point could be continuously adapting in real time.

In *An Algorithm for Comparing Similarity Between Two Trees*; Xu provides a mathematical model that can determine the distance between two arbitrary tree structures. This could be paired with Wang and Ling’s algorithm to further optimize the frequency that changes are required to the holistic flow.

## Reliability Patterns

In *Reliability of Multicast under Random Linear Network Coding*; Tsimbalo, Tassi, and Piechocki describe the challenges encountered with transmitting television signal. They provide an enhancement traditional application level correction (AL-FEC), by providing a notion of parity encoding. Consider a scenario where a sender has access to 3 channels and needs to post messages A and B. If they send one message per channel encoded as (A), (A+B), (B); then the consumer can miss 1/3rd of the messages without data loss.

In *Reliability and Availability Issues in Distributed Component Object Model*; Wang, Damani, and Lee propose a system collection of aspects that can be injected into existing COM servers. These capabilities include automated state checkpointing which allows the stateful object to periodically die without impacting the larger system. This idea aligns with the objective of the system to self-heal.

In *Scalable Eventually Consistent Counters Over Unreliable Networks*; Almedia and Baquero propose a scheme for incrementing a distributed counter such that no numbers are skipped or repeated twice. This is accomplished through Conflict-free Replicated Data Types (CRDT), which can be expanded to any scenario that needs assurances all elements in a set are processed exactly once. Consider the scenario where hydration has a message to refund the customer 1000$, it would be advantageous to guarantee this message is not repeated.

In *A Comprehensive Survey on Distributed Transactions*; Bharati and Attar performed an extensive review of methods used for ensuring consistency across a record set. These include techniques for heterogenous data store technologies and ensuring broad system consistency. This is directly applicable to the hydration scenario as it spans multiple systems.

## Event Patterns

In *SagaMAS: a framework for distributed transactions in microservice architecture*; Limon et. al combine the ideas of the Saga Pattern with Multiagent Systems. A Saga is defined as a sequence of local transactions which become eventually consistent. This occurs through either an orchestrator (eg. central service) or choreographer (eg. log monitor). They construct a tree structure with each node being the tuple (id, name, data, action name, compensation action). The tree is continuously pruned by either raising the action or compensation behavior, until all child tasks are successful.

In *the Prometheus Methodology*; Winikoff and Padgham provides design requirements for multi-agent system which need to coexist. The lectures suggest building the system in terms of goals and aligning the agents to the goals results in better systems. This is the approach that *SagaMAS* used and makes multiple references to this effort.

# Approach

Based on the guidance of the literature review there are two systems which need to co-exist. The first is an auto scaling solution which uses a ‘min-cost max-flow’ model to ensure that the resources are efficiently sized based on a collection of dimensions. The second is the ability to detect hydration failures and apply compensations as a mechanism for self-healing.

If the system can dynamically scale based on set of metrics, then it should process the records in the most resource efficient manner. From the model it would also be possible to identify bottlenecks in the distributed call graph which makes up the micro services. This can lead to prioritized improvements that provide the maximum return on investment. Even with a perfectly error free system there will still be failures which the system must be able to correct without manual intervention.

## Milestones

1. The first objective is to collect the information required to describe the data flow and associate metrics as nodes and edges. This could be accomplished through a small Domain Specific Language (DSL) that models the inputs and outputs of each service. Once these data points are centrally available the ‘max flow model’ can be started and initial parameters configured.
2. An open source tool would be created to transform existing Infrastructure as Code systems into the flow DSL. This would target a 1 or 2 platforms and allow data collection from more participants, allowing for a wider survey of workloads.
3. A reporting portal would be created to display the which resources are incorrectly provisioned. This would enable the participants of the study to gain direct value, as their environments are cheaper to operate. Though an economic incentive it is expected more participation will occur.
4. The flow model will be extended from making recommendations to implementing those choices. For instance, if it determines that a bottleneck exists then the system will compensate by modifying related resource sizing. The correct size can be inferred based on flow graph.
5. The second requirement is the ability to automatically recover from failures. These will be described through a second ‘Saga DSL’ which describes the compensation actions for failures scenarios. The implementation would align with ideas from Limon
6. The Saga DSL will describe steps for registering tasks with the distributed transaction manager (DTM). Amazon, Microsoft, and Google already expose platform native methods for routing unhandled exceptions to a custom endpoint. This functionality will be leveraged to ensure the DTM is properly notified even if process has crashed.
7. It is expected that the Saga implementation would result in duplicates where at exactly-once semantics are required. This could be addressed though a