Section 3: Week 7: Reliable Cloud Data Ingestion

Nate Bachmeier

TIM-8110: Programming Languages and Algorithms

May 19, 2019

North Central University

# Reliable Cloud Data Ingestion

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 hydrating the data network nor reliable data ingesting at the system level.

# Background

Many businesses like Contoso are actively working to transition their proprietary systems into the public cloud. This reduces infrastructure costs and improve the agility to create new features for their customer base. These strengths are well documented through highly optimized ‘pay per use’ pricing models and instant access to virtually unlimited resources.

For many enterprise businesses 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 are hesitant to ‘flip the switch’ across all customers. This deployment decisions such as creating a (1) dedicated deployment instance in the cloud; and (2) then hydrating the history of certain customers. Transitioning a small handful of customers per migration iteration reduces the blast radius and simplifies the communication across stakeholders 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 occurs from (1) new features being added, (2) internal implementation details changing, and (3) unused features removed.

The data migration process has three choices to mitigate these issues: (a) create and maintain bulk load interface for each data store; (b) replay the historic traffic through the steady state interface; or (c) a combination of (a) and (b).

If the system has (1) many heterogenous private data store; and (2) a high degree of data connectivity between them; then it could be prohibitively expensive to choose option (a). 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 externally model this could duplicate large amounts of business logic and be error prone.

Another reason for businesses to desire (b) is that improvements to the data migration scenario are improvements shared by the general customer. Shared fixes can include general correctness, backward compatibility assurances, performance and scalability scenarios.

There are potentially micro services where scenario (a) is desired in combination with (b). A business might have archives of physics simulations, where each required several days to render. In this example 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. Another key advantage of choosing scenario (b) is that enhancements to the migration scenario can extend to generic data ingestion scenarios. Thus, improving multiple dimensions of the system without duplicating efforts.

# 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 regarding the movement of infrastructure are 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 the defect which can impact results across several customers. 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 and ingestion scenarios.

Another challenge is optimizing the throughput rates that ingestions should target. If the rate is too slow it will never complete, yet too fast and it causes scalability issues across the infrastructure. These scalability issues will increase the failure rates and cause invalid results.

If the system was set to scale infinitely then it would become too costly to operate. This would lessen the competitive gain from 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 an entire enterprise migration could last years.

# Goals

## Max Flow Model

These challenges could be restated as there is a need to solve the ‘min-cost max-flow’ (MCMF) across the distributed architecture. There is also a need to model which junctions lead to bottlenecks or are over provisioned. For instance, if ‘service X’ is bound by the scale of ‘service Y’ then either X needs to be reduced for cost savings purposes.

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 unlimited resources changes the dimensionality into ‘N records per V virtual cores per M gigabytes of memory.’

Once these optimization points are identified then a system can be created to provide recommendations to the operations team or apply them automatically. This would result in economic gains of the service in terms of reduced labor overhead and fewer billable cloud resources.

## Min Error Rate

Another objective 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 would exist regarding availability and the net flow rate could be increased higher. It would also reduce the number of customers reported support requests as the holistic system would be more reliable.

## 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 consumer 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 lessen the lead time before they begin reaping the benefits of that transformation.

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 instance counts.

This can be inefficient as it *reacts* to degradation of an *individual* service as the notification mechanism to begin scaling. Instead a holistic flow model could *preemptive* determinethe increase in load will result in *multiple* services needing to scale modifications.

All distributed micro services rely on notifications to be reliably transmitted between one another. For many scenarios the cost of processing duplicate messages is less than the cost of missing the event entirely. That is not to say that all traffic should be redundantly sent, however there are potentially encoding schemes that allow for the introduction of parity bits and auto remediation. These self-healing solutions would increase the fault tolerance of the entire system and provide a more positive user experience to all consumers of the service.

# Literature Review

There are four core literature scenarios that were reviewed (1) cloud migration challenges and strategies; (2) maximum flow and graph analysis; (3); reliable distributed systems; and (4) event broadcasting 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 are (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. The journal also defines levels of cloud maturity which businesses transcend across their journey. While significant attention is provided to the infrastructure, little stated around data migrations.

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

## 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 the 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’ (MCMF) over ‘max-flow’ (MF) model resulted in a 30% savings and an equivalent output. The problem structure is like the hydration scenarios, which suggests it could be advantageous use 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’ (MF), which efficiently update across dynamic networks. If the MF can be cheaply recalculated, then it allows for auto scalers to be more responsive and adapt with higher precision.

In *An Algorithm for Comparing Similarity Between Two Trees*; Xu provides a model for can determining the distance between two arbitrary trees. This could be paired with Wang and Ling’s algorithm to further optimize the frequency that changes are applied the auto scaled system.

## 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 over traditional Application Level Correction (AL-FEC), through a notion of parity encoding. Consider the 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 a consumer subscribed to all three channels could 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 collection of aspects that can be injected into existing DCOM services. This adds capabilities such as automated state checkpointing, which can enable stateful objects to periodically die without impacting the broader 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. The capability could be leveraged by a Distributed Transaction Manager (DTM) to be certain a transaction needs compensation.

In *A Comprehensive Survey on Distributed Transactions*; Bharati and Attar performed an extensive review of methods available for ensuring consistency across a replicated record set. Their survey includes techniques for heterogenous data stores which is directly applicable to the data ingestion and self-healing scenarios.

## Event Patterns

In *SagaMAS: a framework for distributed transactions in microservice architecture*; Limon et. al combine the ideas of the Saga Pattern with Multi-Agent 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). Saga MAS construct a tree structure with each node containing the tuple (id, name, data, action name, compensation action). The tree is continuously pruned by either invoking the actions of a leaf until the operation is successful.

In *the Prometheus Methodology*; Winikoff and Padgham provides design requirements for the co-existence of multi-agent system. The lectures suggest best results are achieved by (1) building the system in terms of goals; and (2) aligning the agent’s functionality with the goals. This is the approach that *SagaMAS* referenced extensively and proposed in their work.

# Approach

## Summary of 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 MCMF 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 should also be possible to identify bottlenecks in the distributed call graph of 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.

## Max Flow Model

The first objective is to build a model that encompasses the required information to determine the MCMF flow. Using a MCMF function instead of MF has the potential to provide a more efficient solution (Benda, Chu, Sun, Quek, & Buckley, 2018). The optimization target would be increasing the number of records per second through the entire system. There are several potential system aspects to minimize such as (1) total cost; (2) total errors; or (3) a domain specific metric.

After the model can reliably tell the desired state of the system, it needs to compare against the actual state. If the deviation exceeds a threshold then the system needs to compensate through initiating auto scale requests. As the scale requests are invoked call backs can notify the model of incremental changes without recomputing the entire graph (Wang, Y; Ling, J, 2018). For huge enterprise environments this will become critical as there can be thousands of highly connected components.

It would also be beneficial to expose a domain specific language (DSL) to express the service connectivity, configuration and thresholds. This would simplify the end user experience and limit their direct interaction with the system. Many enterprises already have this information described through Infrastructure as Code (IaC) languages. It could be advantageous to include a translator for 1 or 2 of the most common IaC languages (eg. Teraform and Puppet).

The final deliverable for this goal would be to deliver a dashboard that shows the operations team how much the auto scale system has reduced their cloud bill. Any additional measurements of success could also be rendered in simple graphical form.

## Min Error Rate

Implementing a distributed transaction management protocol based on Sagas, could improve the reliability of the broader system (Limon, X, et.al, 2018). This can be accomplished by having components emitting schematized intents into log files. Those logs could then be mined by a decoupled service which constructs a tree of desired actions and compensations.

If the system discovers success confirmation messages are missing, then it can automatically apply the compensations to self-heal failures to the system. An investigation is required to determine the correct pattern acknowledging a transaction completed. As reported by Bharati and Attar, there are numerous variations with slight performance and consistency considerations.

There could be challenges with expressing the distributed transaction semantics within the micro service code itself. It is also undesirable to make modifications each time the configuration changes. To mitigate these issues a DLS could be defined to instrument common scenarios at build time, or express distributed component dependencies.

Based on the configuration of the transaction a compensation could ‘at-least-once’ or ‘at-most-once’ semantics. Using a CRDT structure would allow the DTM to honor those requirements even though it is also a distributed asynchronous entity (Almeida & Baquero, 2013).

The success of this deliverable could be a simple KPI that states the number of self-healed operations. Based on the intent tree structure other metrics could be extracted such as failures that were not correctly recovered.

## Define New Patterns

Introducing parity information into redundant messages is a unique idea that could be very helpful in certain lossy scenarios (Tsimbalo, Tassi, & Piechocki, 2017).

This could be implemented as an extension to the Http Client which keeps a sliding window of recent messages. These messages could be sent first in their original form and then later in a RAID like combination.

Services which support the encoding could then decode the messages and replay any that were missing. To determine if the message was not present Almedia and Baquero’s CRDT solution could be used here as well.

Another deliverable of this objective is to determine what additional ideas are present in general communication systems that are applicable to the data ingestion scenarios. There is a lot of potential for reusable patterns.

To measure the success of this feature performance metrics should be published to identify the number of times this helped versus became a no-op. Using standardized performance benchmarks could then be used for a non-bias scoring.

# References

Ahmad, N., Naveed, Q., & Noda, N. (2018). Strategy and procedures for Migration to the Cloud Computing. *2018 IEEE 5th International Conference on Engineering Technologies & Applied Sciences, 22- 23 Nov 2018, Bangkok Thailand*.

Almeida, P., & Baquero, C. (2013). Scalable Eventually Consistent Counters over Unreliable Networks.

Benda, D., Chu, X., Sun, S., Quek, T., & Buckley, A. (2018). Modeling and Optimization of Renewable-Energy Sharing among Base Stations as a Minimum-Cost-Maximum-Flow Problem. *2018 IEEE 87th Vehicular Technology Conference (VTC Spring) Vehicular Technology Conference (VTC Spring), 2018 IEEE 87th. :1-5 Jun, 2018*.

Bharati, R., & Attar, V. (2018). A Comprehensive Survey on Distributed Transactions based Data Partitioning. *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*.

Cefaratti, M., & Lin, H. (2018). Exploring Data Center Migration: A Case Study. *JOURNAL OF INFORMATION SYSTEMS Vol. 32, No. 1 Spring 2018*, 1-17.

Ferdiana, R., & Putra, G. (2018). A Review of Cloud Migration Strategies in the Developing Country. *2018 4th International Conference on Science and Technology (ICST), Yogyakarta, Indonesia*.

Goldberg, A., & Tarjan, R. (2014). Efficient Maximum Flow Algorithms. *COMMUNICATIONS OF THE ACM* , 82-89.

Limon, X, et.al. (2018). SagaMAS: a software framework for distributed transactions in the microservice architecture. *2018 6th International Conference in Software Engineering Research and Innovation*.

Tsimbalo, E., Tassi, A., & Piechocki, R. (2017). *Reliability of Multicast under Random Linear Network Coding.* Retrieved from Cornell University: https://arxiv.org/abs/1709.05477v3

Wang, Y., Damani, O., & Lee, W. (1997). Reliability and Availability Issues In Distributed Component Object Model (DCOM) . *0-7803-4290-9/97/$10.00 @ 1997 IEEE*.

Wang, Y; Ling, J. (2018). A Study on Rapid Incremental Maximum Flow Algorithm in Dynmaic Network. *2018 1st International Cognitive Cities Conference (IC3)*.

Winikoff, M. (2004, April 2). *The Prometheus Methodology.* Retrieved from RMIT University: https://www.cs.rmit.edu.au/agents

Xu, H. (2014, April). *An Algorithm for Comparing Similarity Between Two Trees.* Retrieved from Graduate School of Duke University.