Week 7: Improving Availability and Performance of Cloud Systems

Nate Bachmeier

TIM-7140:Software Engineering

April 11th, 2021

Northcentral University

**Improving Availability and Performance of Cloud Systems**

Applications move to the cloud to gain access to elasticity, instantaneous provisioning, sophisticated security, and cost controls, among other reasons. These new capabilities shift the modern architecture away from monolithic designs toward micro-service systems. While many applications are modernizing, their high-availability approach is not evolving (Verbitski, et al., 2018). These traditional strategies include several tried-and-true methodologies, such as state checkpointing and fail-over clustering (Zhao, 2014). Fundamentally, these strategies make assumptions regarding the physical hardware constraints. However, the cloud’s virtualization enables bending some of these rules. Engineering teams need to reassess cloud-native patterns as a mechanism to improve their systems’ performance, reliability, and economics.

# Background

Distributed systems are the most complicated computing environments because of their parallel and asynchronous nature. Many implementations also make false assumptions regarding the network’s reliability, security, homogeneousness, latency, bandwidth, and transport costs (van Steen & Tanenbaum, 2019, p. 986). When systems introduce one of those fallacies into the design, it produces subtle defects under production loads. Businesses combat these risks through high-availability architectural patterns that promote self-healing (Yang, Min, Yang, & Li, 2014). These strategies follow combinations of reactive (e.g., heart-beating) and proactive solutions (e.g., rejuvenation tactics).

There are inherent challenges with every high-availability solution. For instance, state check-pointing requires periodically writing memory to disk. This operation is exceptionally I/O intensive and significantly degrades performance, despite only a few of these snapshots ever used (Cheng, Huang, & Lee, 2019; Wu, Shang, Peng, & Wolter, 2020). Organizations that can remove these performance penalties could reduce resource requirements, improve Quality of Service (QoS), and become more competitive through cost reductions.

Another standard approach is through fail-over clustering and disaster recovery technics. This requirement translates into resource over-allocation and accepting wastefulness (Yan & Wang, 2020). Meanwhile, cloud-native systems support instantaneous provisioning, elasticity and can go global in minutes. These capabilities promote more efficient scheduling and allocation methodologies. However, even mature businesses limit their cloud exploitation to stateless, not stateful, services. Architects need to define frugal patterns that lead to reliable systems operating above unreliable and dynamic hardware.

# Problem Statement

           Traditional monolithic systems implement high availability within the finite constraints of private data centers. In contrast, cloud-native solutions exploit virtually infinite scalability across multiple global regions. Despite this additional flexibility, most businesses do not fully use the performant high availability potential of operating on public cloud platforms. Researchers must define new architectural tactics that leverage the cloud’s unique characteristics.

The dichotomy of traditional and the potential for cloud-native high-availability is most apparent with stateful services. Unlike stateless services, it is challenging to handle stop-faults elegantly. Platforms like Apache Spark, Flink, and Storm mitigate these concerns through checkpointing. However, this solution decreases overall throughput by 35-40% (455-570MB/s versus 755-900MB/s) (Cheng, Huang, & Lee, 2019, p. 12). Businesses must provide extra resources to offset this degradation. Increasing the cluster size also means greater chances of failing due to greater cross-component communication needs and higher I/O requirements.

More ideally, software platforms decompose the processing nodes into a collection of micro-services. These components can execute across dedicated hardware that does not impact the performance-critical path. Cloud providers also expose features that replicate and multicast information across distinct nodes. While these technologies are not unique to the cloud, they are historically cost-prohibitive and only available within enterprise topologies. In contrast, public cloud providers democratic access to sophisticated technologies.

# Goal

Architecture teams must define modern high-availability mechanisms that exploit the cloud’s capabilities. This constructive research core deliverable implements a benchmark of streaming applications that follow cloud-native patterns. The model will cover several standard use-cases (e.g., WordCount and StreamGrep) while minimizing overhead through high-availability micro-structures. Second, an assessment will confirm these micro-structures are generalizable by examining an open-source platform’s internal requirements, such as Apache Spark or PostgreSQL.

# Relevance and Significance

Modern applications are moving toward micro-service architectures that require zero downtime (Rudrabhatla, 2020). Businesses meet these requirements through mechanisms that use excessive resources. Alternatively, cloud-native solutions would reduce costs and complexity. When organizations become more efficient, it increases their competitiveness. This characteristic makes these optimizations broadly applicable.

# Literature Review

## Disaster Recovery Techniques in Cloud Computing

Distributed systems are complex environments that must defend against a litany of sources. Tamimi et al. (2019) provide a taxonomy with top-level origins, including natural disasters, network failures, network intrusions, system failures, malicious code, and human errors. Unsurprisingly, human error accounts for nearly 60% of all issues.

When a service outage occurs, the business needs to restore operations through a remediation strategy. “Broadly speaking, all those [recovery] techniques focus on three different aspects, such as cost control, data duplication and security issues (Tamimi, Dawood, & Sadaqa, 2019, p. 847).” Each workload within an organization comes with unique characteristics that dictate the Recovery Time/Point Objective (RTO/RPO) requires. Consider a FinTech company that hosts both an internal payroll portal and a real-time trading application. If the trading system is offline, then the business risks significant losses relative to the human resource app. Mitigating that financial loss justifies higher costs and complexity.

## Deployment Challenges

Modern businesses use Continuous Integration and Deployment (CI/CD) pipelines to release production updates multiple times per day. This requirement introdunces complexity for high-availability systems that seek to maintain zero downtime. Rudrabhatla (2020) presents constructive research into rolling, blue-green, and canary methodologies, stating the core difference is the cost, duplication, and security. These observations directly overlap with Tamimi et al. (2020). Blue-Green deployments refer to duplicating the environment and using load-balancer or DNS (Domain Name Service) magic to toggle between them. This strategy has the most safety, but also the highest cost.

## Evolution of Clustering Strategies

Cheng et al. (2005) describe a case-study with International Technology Roadmap for Semiconductor’s Factory Information and Control Systems (FICS). This mission critical system cannot exceed 240 minutes downtime annually (99.95% uptime). Similar to other companies, FICS relies on clustering services. Under the covers these solutions employ a heartbeating mechanism to discover compute node failures. Additional custom client middleware also transparently retries failures. Despite these architectural protections, the operations team still found it challenging to meet this aggressive Service Level Objective (SLO).

The engineers began collecting metrics, leading to the discovery that node failure is proportional to service instance age (total runtime). This makes logical sense, as long running processes have more opportunity to become corrupt or leak resources. They began collecting data to predict Mean Time To Failure (MTTR) per service. Today, FCIS experiences than 120 minutes annual downtime (99.98% availability).

Wen et al. (2020) present constructive research into placement strategies on Kubernetes. Kubernetes is a container orchestration system that supports policies for physically distributing workloads. For instance, they span Elastic Search across four physical nodes, then demonstrate that rebooting hosts does not impact system availability. Finally, the authors present a series of equations that maximize resilency and minimize hardware requirements.

Dedicated fail-over clusters increase availability and operational costs by pre-provisioning idle resourc. Yan & Wang (2020) propose upgrading traditional active & passive models (1+1) for H+K configurations. Essentially, administrators deploy H-service instances plus K-extra nodes. Then the environment can maintain full availability until K-concurrent failures.

## Influence of Checkpointing

Zhao (2014) documents multiple strategies for building dependable distributed systems. Their patterns generally cluster into partitioning, checkpointing, and concensus constructs.

|  |  |
| --- | --- |
| Construct Method | Unique Characteristics |
| Partitioning | 1. Failure domain that constraining impact of error 2. Serve as logical horizontal scaling unit |
| Checkpointing | 1. Follows a fetch-do-persist-acknowledge loop 2. State is Write Many Read Maybe |

Partitioning methods both constrain the blast radius of failures and expose a logical unit for horizontal scaling. Checkpointing structures Fetch-Do-Acknowledge-Persist loop

1. Building Dependable Distributed Systems (Zhao, 2014)
   1. See 8120-7 about checkpointing and related strategies
2. Reactive Batching Strategy
   1. Influence of intelligently removing batches
3. Performance of distributed systems
   1. AF-Stream avoiding checkpointing

## State-of-the-Art Solutions

1. Aurora: Avoiding Distributed Consensus

# Approach