Section 1: Week 1: Basic Distributed System

Nate Bachmeier

TIM-8120: Distributed Systems

September 22nd, 2019

North Central University

# Distributed System Structure

Contoso Clothing is an international manufacturer and retailer of personal attire. Their manufacturing processes span three continents and need to meet the needs of their several thousand stores. Each location has dozens to hundreds of employees that need to have access to computing environments, point of sales services, and printers.

There are multiple configuration options for the construction of this environment, each with different pros and cons. If they lack the understanding of these trade-offs, then the system will be (1) be too expensive to operate, (2) unable to meet peak loads and (3) unreliable during complex scenarios.

# Command and Control Structured

Traditional systems, such as presented in figure one, might use a static hierarchical structure where the headquarters will replicate policy through a tree structure. Within the tree there each branch represents an aggregation point, such as North America or Europe. There can be child branches to further distribute the load to management points, such as Washington and Spain. These management points will then execute commands on clients (leaf nodes) and collect any local results.

## Tree-Based Distribution Strategies

While these trees are good at distributing load across a vast breadth of systems, they introduce several single points of failure (Annadurai & Vijayalakshmi, 2016). To mitigate these risks, administrators implement these branch nodes as complex systems, not individual compute units (Khaneghah & Sharifi, 2013). For instance, the *Washington Management Point* might not be a single server but a load-balanced ring of servers.

By introducing a load-balancer, the administrators are trading availability for additional complexity. Consider the impact of a client sending three requests to the load-balancer, which in turn hands them to three different service instances. These scenarios can lead to (1) out-of-order eventing, (2) partial conversation failure, and (3) redundant resource allocations – to name a few challenges.

## Influence on Software Rejuvenation

However, it can simplify other scenarios such as software rejuvenation strategies as there are multiple identical processors within the functional group. Rejuvenation is the operational procedure of recycling private instance state after it has exceeded a threshold (Yang, Min, Yang, & Li, 2013). Since the functional group contains two or more nodes, the rejuvenation can be applied to one node while the other continues to service requests.

Perhaps the message processor leaks memory and becomes unresponsive after the working set exceeds 1GB. In this scenario having an external process (1) monitor the performance metric then (2) cleanly cycle the worker process as it approaches the threshold, would (3) increase the perceived reliability of the message processor.

# Transition Manufacturing Hubs to Pub/Sub

Contoso might decide that it is not economical to manage hierarchies of proprietary configuration on physical servers across their manufacturing presence. Instead, they could leverage their existing internet connections to move into a hosted cloud.

## Data Processing Networks

Data Processing Networks (DPNs) is a simple and elegant pattern that leverages event publications and subscriptions to route messages (Celar, Mudnic, & Seremet, 2016) (Baudisch & Schneider, 2013). The process begins with the client posting, directly or via a broker, their notification into a multicasting service. The multicast applies any local transformations, and then forwards to zero or more subscriber’s FIFO queues. The arrival of the message triggers a *function* to process the event and remove it afterward.

Generally speaking, these functions can exist in any programming language, as the transmission and implementation are decoupled. Assuming that the message within the queue represents a self-contained task description, it then becomes possible to execute each task in parallel asynchronously (Baudisch & Schneider, 2013). When combined with either Function as a Service (FaaS) or High-Performance Compute Clusters (HPCS), these parallel invocations can handle Internet-scale requirements.

## Pitfalls of Distributed System Design

Peter Deutsch formulated these flaws as the common assumptions that new distributed systems engineers make: (1) the network is reliable, secure, and homogeneous; (2) topology does not change; (3) latency is zero, bandwidth is unlimited, transport costs are free; and (4) there is only one administrator (Steen & Tanenbaum, 2016).

While the Data Processing Network can address elements of these false assumptions, there are additional requirements for the systems engineers. For example, the topology is decoupled from the DPN as messages flow to topic subscribers. Similarly, the message processors are allowed to be heterogeneous as communication occurs through serializable payloads are placed in FIFO queues.

Other scenarios, such as mitigating network reliability issues and efficiently using network capacity, require efficient protocols and procedures within the function implementation (Steen & Tanenbaum, 2016). Steen and Tanenbaum propose that these characteristics need to be locally handled instead of assuming an external third-party will “do the right thing.”

For instance, when the function pulls a message from its queue (1) there needs to be positive hand-off confirmation from (2) each of the caller services they have (3) accepted their copy of the event and (4) replicated it into a durable store.

Only then, can the original message be removed from the function’s queue with confidence the payload won’t become lost. Simple scenarios, such as rejuvenation is overdue or incorrectly compensated cluster load, will cause middleware based solutions running on the caller service to become error-prone over time (Steen & Tanenbaum, 2016) (Khaneghah & Sharifi, 2013) (Yang, Min, Yang, & Li, 2013).

# Retail Stores to Orchestration Services

Orchestration Services are systems that execute business process workflows. They describe the sequencing requirements to perform some action, along with optional compensation actions to remediate failures (Venkatesan & Sridhar, 2015). An example implementation would be the Web Services-Business Process Execution Language (WS-BPEL) open standard running on Apache Orchestration Director Engine (ODE).

## Design Paradigm

*Programming in the Large* refers to the ability to author complex workflows using Extensible Markup Language (XML) or equivalent tags. This authoring process tends to be visual and targeted toward the business domain experts.

*Programming in the Small* refers to the code behind for these workflow tags, and any custom integration logic that is needed. This authoring process tends to target systems engineers who use their existing Integrated Development Environments (IDE).

Applications written for WS-BPEL can follow this paradigm to build highly reliable automation. The standard can also integrate into existing web service technologies such as Simple Object Access Protocol (SOAP), Representational State Transfer (REST), Web Service Description Language (WSDL), Web Service Security (WS-Security), and more (Keller, 2007). These touchpoints can significantly reduce the costs of bringing legacy systems into a more modern design.

## Challenges