The Apstra Operating System (AOS) and The Distributed Systems Challenge in Data Center Automation

David Cheriton, Apstra

INTRODUCTION

You have to admit: The modern data center is an astounding accomplishment in computing. It is capable of delivering an enormous level of computational power, storage, and communication with cost-efficiency in both CAPEX and OPEX, availability and extensibility that would be unimaginable just a few years ago. As a consequence, enterprises are able to deploy applications and services without being limited by computational resources. It has spawned a whole new thinking in the design and deployment of these application and services under the name of cloud computing. Although you may find it a flakey term, the cloud has come to denote the almost limitless computing capacity provided by modern data centers, whether as a public cloud or a private cloud.

With computational resources no longer the first-order limiting factor, your organization is likely pushed to provide more extensive services and applications, delivering always-on responsive interactions with their customers with rapid introduction of new features and services. Whether coming from internal innovation or driven by external competitive forces, you cannot ignore this trend, and the pace is accelerating. As a consequence, the data center becomes foundational to a business. Failures and poor operational practices have direct business consequences. Lack of agility and excessive cost are a significant competitive disadvantage. A single typo by an operator in a data center can put your company's name in an (unwelcome) front page article in the Wall Street Journal.

"You can't manage what you can't monitor."

The cost-effective provisioning of this astounding level of computational power is achieved by a complex interconnection of commodity servers and increasingly interchangeable switches. The diversity of protocols and features available on switches and services makes the configuration of a data center a complex endeavor. Moreover, while the initial stand up is challenging, configuration is not a one-time thing. You are going to have a continual need to incrementally deploy new servers and switches and undeploy others, with frightening opportunities for errors and failures at every stage.

At the same time, components can fail, causing disruption. In fact, at scale, you have to expect that something is broken or not working correctly at any given time.

If you are an experienced data center operator, you surely recognize that the "friendly" form of failure is when a switch or a server just totally stops working. This form of failure is very evident and can be easily handled by deconfiguring this particular node if a quick repair, such as plugging the power supply back in (not an uncommon problem), is not feasible. The nastier form of failures manifests as transient failures of internal components. A switch is dropping an excessive number of packets on a given port or perhaps just dropping packets with specific characteristics. The result is that the response time for certain applications goes up significantly, not a hard failure but still unacceptable from a business standpoint. You are then confronted with the problem of root causing and remedying this "grey-out".

Therefore, what you need is not just the tools to help configure and reconfigure the data center, but also, the means to monitor its operation and detect both subtle and blatant forms of failures. Peter Drucker, the famed business consultant, was often quoted as saying "You can't manage what you can't measure." We bring this wisdom into our domain by slightly modifying the quote to say, "You can't manage what you can't monitor."

The data center is a highly distributed system. It thus requires a highly distributed system to configure and monitor it. Many others dramatically underestimate the distributed system challenge underlying data center automation. The automation system itself needs to be responsive, needs to scale, needs to gracefully deal with overload, needs to be fault-tolerant, and needs to be secure. It is not adequate to just provide various application-specific solutions within the context of the automation software. It is also not adequate to pull together a collection of open source point solutions and try to get them to interoperate. Let's make this clear by considering this monitoring challenge first: how to provide the necessary telemetry.

DATA CENTER TELEMETRY

How do you monitor the operation of a data center?

A modern data center can produce billions, if not trillions, of events per second. You cannot afford to monitor every event. On the other hand, conventional SNMP approaches that poll every 5 minutes generally fail to indicate anything about the performance dynamics, proving unhelpful for root causing performance degradation type of failures, as mentioned above, unless alerts are triggered.

Because of the huge rate of events, standard techniques of aggregating into counters and other statistics, as well as selective reporting of critical of events and alerts are necessary. Nevertheless, when you do not know what the next problem will be, it is hard to know what information you will need to root cause it. Therefore, more reporting is better than less. This is clearly one of the lessons from experience with "big data."

Thus, a key distributed challenge is providing efficient highly available communication, recording and processing of important events from the lowest levels of components in the data center through to real-time analysis and alerting, and on to a central repository for detailed big data post-processing. This must be done such that it provides low latency between critical situations arising and the communication of operator alerts, if not auto-remediation. This must also be done so that it does not interfere with the operation of the data center. Additionally, it must deal with overload situations when peak demand and failures conspire to produce "event storms." In fact, it is at the time of peak load that failures are more common, and it is at the time of failures and peak load that responsive monitoring is most needed. Finally, it must be done such that the telemetry system itself does not fail in spite of the overload conditions and failures that are taking place.

Rather than trying to build telemetry-specific solutions from the ground up, so to speak, or trying to couple various open-source point solutions, the Apstra Operating System (AOS) relies on an underlying distributed operating system infrastructure that properly factors the telemetry service specific code from that required to solve these distributed system challenges.

Rather than trying to build telemetry-specific solutions from the ground up, so to speak, or trying to couple various open-source point solutions, the Apstra Operating System (AOS) relies on an underlying distributed operating system infrastructure that properly factors the telemetry service-specific code from that required to solve these distributed system challenges. This infrastructure is discussed next.

AOS DISTRIBUTED OPERATING SYSTEM INFRASTRUCTURE

AOS is based on an innovative distributed systems infrastructure that provides efficient communication, distributed processing, fault-tolerance, security, and extensibility. This provides the basis for AOS closed-loop monitoring and configuration, distributed across the components of the data center.

A key part of the AOS distributed infrastructure is its in-memory database technology.

AOS DISTRIBUTED IN-MEMORY DATABASE

AOS is based on a highly efficient in-memory distributed database technology with built-in replication and failure recovery. Events are recorded in each device as well as propagated to nearby instances of the relevant partitions of this in-memory database, running on AOS server hosts, from where it is replicated as desired. The database is configured to store recent monitoring data. It can also be configured to forward this data to a separate big-data processing cluster for off-line analysis.

On server failure affecting this in-memory database, fail-over to a secondary and recovery of the primary takes place automatically and in seconds. On disconnection of a device from the central database, monitoring continues locally by the AOS device agent, which reconnects and resynchronizes with the central system database as soon as possible.

The in-memory database provides automatic incremental replication of state as well as monitoring and fail-over of partitions to ensure on-going state availability. It also supports system-wide naming of objects and relationships, facilitating meaningful reporting and extensibility.

Under overload, the database automatically coalesces statistics reporting to larger intervals, thereby reducing the load. It also prioritizes events and alerts while suppressing duplicates. This coalescence and prioritization is built into the in-memory database technology at the lowest levels of the system.

In contrast, other systems attempt to handle overload on a case-by-case application basis for specific types of events, but fail to handle overload cases that can occur at lower levels. Also, many of the specific application cases are not considered or not handled properly.

The in-memory database technology also allows separate agent processes, on the same host or a different host, to get efficient access to selective portions of the real-time state of the data center as well as to subscribe to particular state updates and events. These agents can further update the central database with derived data, events, and alerts using the same means of update as the device reporting. Using this capability, AOS provides real-time analysis of the operational state, including closed loop monitoring.

Many other systems treat monitoring as a communication problem, getting events to customer-provided equipment. However, that puts the entire burden on the customer to deal with state maintenance, sharding, overload, and fault recovery.

Some systems attempt to deal with their inefficiency by providing selective reporting of events. That is, detailed reporting does not load the system in general because it is only turned on when needed. Unfortunately, turning on the detailed reporting when the system is already under severe load tends to aggravate the situation, if not cause catastrophic failures. Thus, the capabilities cannot be used at precisely the time at which they are most needed.

In contrast, AOS is designed around an "always-on" model of real-time monitoring, with a data pipeline from the originating component/device through to real-time storage and processing through to bulk off-line storage and processing.

In contrast, AOS is designed around an "always-on" model of real-time monitoring, with a data pipeline from the originating component/device through to real-time storage and processing through to bulk off-line storage and processing.

INTEGRATED EFFICIENT PUBLISH-SUBSCRIBE COMMUNICATION

The AOS distributed operating system provides an integrated binary state-oriented publish-subscribe protocol for all communication within AOS. By using publish-subscribe, data can flow asynchronously and selectively to any portion of the system that requires that data, avoiding the round-trop times and overheads of conventional request-response RPC or polling communication. By state-oriented, we mean the protocol is designed around specifying state updates, allowing optimization of the protocol representation which, in conjunction with its compact binary wire format, allows further reduction in communication and processing overhead.

By integrated, we mean the protocol is designed directly into the in-memory database, allowing it deliver updates to database state without application intervention, thereby reducing processing overhead and update latency. It is used for all interprocess communication, including propagating updates from devices to the central database partitions, from primary partitions to secondary partitions, and for logging and recovery.

The integrated protocol also allows for graceful handling of overload. Under overload and congestion, messages can be dropped while still recording that remotely cached state is now stale, later resynchronizing the state as necessary when the load subsides. In doing so, the protocol explicitly recognizes that the most recent state is of the greatest value in a real-time system, not working off a backlog of old messages, as many systems do.

Many systems use text-based presentations of events, such as JSON, to provide ease of use and claimed extensibility. However, this approach incurs significant communication and processing overhead throughout the system. In contrast, AOS provides export of event data, in real-time if so desired, in a variety of formats including JSON, only incurring this overhead when called for, and only on the selected data of interest. These export processes do not impact the rest of the AOS telemetry system.

On top of this asynchronous publish-subscribe communication, the AOS distributed infrastructure allows applications to optionally use transactions to impose atomicity and synchrony. The transactional model proves useful for configuration, but not for telemetry relative to its performance overhead. In contrast to this approach, conventional systems provide synchronous communication in the form of RPC and polling and then attempt to retrofit an asynchronous event mechanism. Our approach is faster, simpler, and leads to more predictable dynamics and graceful recovery during overload and failures. It is optimized for the most performance demanding aspect of data center automation, namely the telemetry.

Providing a distributed in-memory database entails scheduling and running multiple processes across a collection of servers. It also entails detecting when one of these processes has failed and restarting it. The AOS distributed infrastructure provides this capability as a general distributed scheduling and process management service.

DISTRIBUTED SCHEDULING AND PROCESS MANAGEMENT

Providing a distributed in-memory database entails scheduling and running multiple processes across a collection of servers. It also entails detecting when one of these processes has failed and restarting it.

The AOS distributed infrastructure provides this capability as a general distributed scheduling and process management service. Consequently, other processes, such as those doing telemetry-specific processing can be scheduled by the same mechanism. In fact, this service allows third-party and customer-specific to be scheduled and monitored as part of the same service.

This scheduling service is implemented using the same in-memory database technology (to store the scheduling configuration and status) and base publish-subscribe protocol, minimizing the mechanism required to run AOS and providing visibility (through the database) into the operation of this scheduler facility. This approach further facilitates extensions to the scheduling, both in terms of the state that is maintained, and by additional processing agents that monitor or control the scheduling.

In contrast, other solutions often handle this process scheduling, monitoring, and restart requirement with an application-specific solution or else introduce a separate distributed scheduler with its own protocols, database, and performance characteristics. The former forces other processing in the system to solve their own scheduling, monitoring, and restart problem. The latter leads to complex dependencies and performance problems, particularly with failures and overload.

EVENT REPORTING AND LOGGING

The infrastructure provides an efficient form of event reporting based on the common binary publish-subscribe protocol, allowing event reporting to work with the in-memory database technology. This event reporting is used by the infrastructure itself as well as applications running on top of this infrastructure, such as the telemetry, configuration, and verification processes.

This event reporting and logging support allows the AOS applications to efficiently provide detailed event reporting integrated with the event reporting of AOS itself, providing visibility and tracking for all aspects of AOS.

SECURE OPERATION

The AOS distributed operating system provides a unified authentication and access control basis for all applications running on top of it. Thus, one authentication mechanism supports access control on all AOS data, configuration, and computing resources. The integrated publish-subscribe protocol supports full secure communication between AOS agent processes and the in-memory database, tying this communication to principals authenticated by the above mechanism. Consequently, the security of AOS is relatively simple to describe yet unified and comprehensive in realization.

Overall, the AOS distributed infrastructure provides a unified platform for running distributed applications, supporting telemetry, but also supporting configuration, verification, and auto-remediation, as discussed next.

AOS device telemetry agents are scheduled on each monitored device, periodically transmitting telemetry data to the AOS in-memory database. Additional telemetry agents source data from this in-memory database and provide time series presentation, analysis, and alerts in real-time.

TELEMETRY

Running on top of the distributed operating system infrastructure used by AOS, telemetry in itself can be considered a distributed application of this distributed operating system. AOS device telemetry agents are scheduled on each monitored device, periodically transmitting telemetry data to the AOS in-memory database. Additional telemetry agents source data from this in-memory database and provide time series presentation, analysis, and alerts in real-time. One or more additional agents provide post-processing and storage of the telemetry data in an off-line compute cluster for subsequent processing.

This distributed multi-process data pipeline relies on the in-memory database for data persistence, the publish-subscribe binary protocol for efficient communication between components, the distributed scheduling and process management to execute these processes and to ensure these processes continue execution, despite server failures and restarts. Finally, the actual execution of this telemetry application can be monitored by the event log mechanism.

The clean separation of the AOS distributed operating system support from this “application” really shows benefits by considering the other applications or subsystems that run as part of AOS, as considered next.

INTENT-BASED CONFIGURATION

Intent-based configuration requires a collection of processes to translate the high-level intent specification into a generic configuration based on available resources and then render to device-specific configuration specifications. At scale, and a diversity of devices, it is not feasible to use a monolithic single application process to provide this capability.

AOS uses its distributed infrastructure to structure the configuration rendering as a distributed application that stores its state in the in-memory database, makes this state available to other agents by the binary publish-subscribe protocol, relies on the distributed scheduler for execution, and reports events using the provided event log reporting mechanism. The configuration information with its significantly different properties from the telemetry data, is stored in separate partitions from the telemetry, allowing clear separation of resources and reliability policies.

AOS uses its distributed infrastructure to structure the configuration rendering as a distributed application that stores its state in the in-memory database, makes this state available to other agents by the binary publish-subscribe protocol, relies on the distributed scheduler for execution, and reports events using the provided event log reporting mechanism.

As such, configuration is essentially one more application using this distributed infrastructure, and is thereby able to run on the same pool of system resources in harmony with the telemetry subsystem.

CLOSED LOOP TELEMETRY AND VERIFICATION

A key feature of AOS is providing closed-loop telemetry, namely verifying that the data center is operating according to the specified configuration. A variety of incidents can cause departure from the specified configuration, including device failure, cables being misconfigured, local console changes, and others.

In AOS, verification is implemented as a set of separate agent processes that monitor the telemetry data as it is stored into the central in-memory database, check this data for consistency with the intent-specified configuration, and provide alerts on discrepancies between the observed and the intended.

The in-memory database and underlying publish-subscribe protocol make the data required for this verification available with low overhead and latency to the verification processes. The distributed process scheduling, monitoring, and restart capability ensures that the verification is running continuously with the same availability as the in-memory database capability.

AUTO-REMEDIATION

The holy grail of data center automation is auto-remediation, the ability to automatically diagnose and remedy a situation revealed by the telemetry. However, auto-remediation is not a single feature that can be realized within any foreseeable time frame. Different problem scenarios may require different diagnosis logic and entail different remediation steps. Thus, there is a need to tackle one problem after another. For example, a simple case of a device losing its configuration can be auto-remedied by reloading the generated configuration. However, incorrect behavior of a device that appears to have the correct configuration requires more in-depth diagnosis and requires physical device reboot or replacement.

A key feature of AOS is providing closed-loop telemetry, namely verifying that the data center is operating according to the specified configuration.

Auto-remediation is also not without its risks. Failure scenarios can be complex and difficult to diagnose. Incorrect diagnosis can clearly lead to incorrect remediation, making the situation worse. Moreover, diagnosis and remediation can require complex, evolving software that is infeasible to stabilize to the same level as the rest of the system.

AOS executes auto-remediation as separate processes, isolated from the other subsystems, yet with the required visibility into the state of the data center through the distributed in-memory database. Again, the auto-remediation relies on the AOS distributed operating system infrastructure to be scheduled, restarted, and properly monitored.

While the initial form of auto-remediation in AOS is modest, the distributed operating system infrastructure provides the means to evolve and scale this capability over time without compromising the availability, responsiveness or security of the rest of AOS.

This AOS emphasis on extensibility not only allows Apstra to provide responsive feature velocity as we go forward, but also allows third party and customer extensibility, again without compromising the operational properties of AOS.

THIRD PARTY AND CUSTOMER EXTENSIBILITY

While Apstra is working hard to extend AOS to meet various customer requirements, we recognize that third parties and customers themselves can significantly contribute to solving aspects of the data center automation problem. Moreover, specific customers may have unique configuration practices and unique monitoring requirements that necessitate customization of AOS.

To this end, the AOS distributed operating system infrastructure supports introducing customer or third-party specified processes using the same facilities as used by the core AOS subsystems described above, refined with extensibility features oriented towards customer usage.

Consequently, customers are not locked into a particular AOS model of data center operation, but can treat AOS as a platform on which to customize to their unique needs and practices. The telemetry, configuration, verification and auto-remediation subsystems provided by AOS are highly configurable and minimize the specialization and coding that each customer has to perform. However, the AOS distributed operating system infrastructure enables this customer extensibility while preserving the protection, security and functionality of the rest of the AOS core operation.

...customers are not locked into a particular AOS model of data center operation, but can treat AOS as a platform on which to customize to their unique needs and practices.

CONCLUDING REMARKS

Data center automation means automating management. While automatically configuring all the devices in the data center from a high-level specification is important and complicated to do right, more challenging is monitoring all these devices to ensure they remain properly configured and operating correctly. In contrast to other purported solutions in this space that focus first on configuration, AOS starts from the premise that monitoring, and thus telemetry, is foundational, as Peter Drucker essentially recognized years ago.

Rather than building this telemetry support as a specialized solution from the ground up, AOS recognizes there is a significant distributed systems challenge underlying providing telemetry.

Consequently, AOS is built on a scalable real-time distributed infrastructure that provides high availability, low latency response, security and extensibility well-suited for distributed event monitoring, but more broadly applicable. The key components of this infrastructure are:

- In-memory distributed database
- Integrated efficient secure publish-subscribe binary-level communication
- Distributed scheduling and process management
- System event reporting and logging
- Distributed authentication and access control

The result is a unified platform on which real-time telemetry is provided as a subsystem or application, with low latency response, high availability, security, and extensibility, yet with the telemetry aspect properly factored from these underlying distributed services. Conceptually, there is a full-function distributed operating system on top of which telemetry services run as one set of applications. Thus, rather than ad hoc application-specific solutions to failures, overload, and security, there is one unified solution in this distributed operating system using a unified pool of computing resources.

With this strong foundation, data center configuration from high-level intent can be provided as yet another distributed application, co-existing with telemetry. This configuration application uses the same means to handle data replication, fault recovery, scheduling, and security as used by telemetry.

Furthermore, verification applications are provided running on the same platform, coupling telemetry to the intent-based configuration, and ensuring the data center is operating as intended or else immediately alerting the operator.

Taking it a step further, auto-remediation agents can be provided as additional capabilities over time.

In each of these "applications," the distributed infrastructure is providing the same efficient communication, distributed state access and replication, fault-tolerant recovery, scheduling and security, and thus a unified response to the key distributed system challenges of failures, overloads, scalability, and extensibility.

Building on this foundation, we are confident that AOS is positioned to not just handle these key distributed system challenges, but to exploit the potential of a high distributed solution in achieving industry-leading fine-grain telemetry, availability, and feature velocity, the latter not only by our own innovation, but also by empowering our customers.

In each of these "applications," the distributed infrastructure is providing the same efficient communication, distributed state access and replication, fault-tolerant recovery, scheduling and security, and thus a unified response to the key distributed system challenges of failures, overloads, scalability, and extensibility.

www.apstra.com, @ApstraInc, and for more information, contact us at info@apstra.com