# CONSTELLATIONS IN THE CLOUD: VIRTUALIZING REMOTE SENSING SYSTEMS

*Andrew G. Schmidt, Vivek Venugopalan, Marco Paolieri, Matthew French*
Information Sciences Institute
University of Southern California
{aschmidt, vivekv, paolieri, mfrench}@isi.edu

## ABSTRACT

This work presents the Virtual Constellation Engine (VCE), a software framework and run-time system designed to facilitate exploration of different remote sensing constellation and sensor web configurations from an on-board processing perspective, using the cloud. Users can launch heterogeneous constellations, describe different on-board processing configurations, and simulate and verify autonomous operations. VCE eases development for applications leveraging complex compute resources and has been demonstrated on Amazon Web Services to provide speedups over *20,000*× of state of practice systems.

***Index Terms*—** Remote sensing, distributed systems, constellation, sensor web, on-board processing, emulation

## 1. INTRODUCTION

A myriad of advances in remote sensing technologies is spawning a renaissance in the types of observing strategies that can be deployed from constellations of several small-sats or cubesats collaborating on an objective autonomously [1] to sensor webs of distributed in-situ, airborne, and satellite based sensors. Such scalable, heterogeneous sensing solutions have the potential to greatly impact the availability of low latency data products for emergencies and the richness and quality of science products for hydrology, cryosphere, biodiversity, weather and more [2]. Example capabilities include satellite constellations flying to take advantage of increased temporal sampling [3], enhanced hyperspectral instrument capability that enables next generation land imaging, ocean biology and ecology, spectral reflectance measurements, and geobased hyperspectral IR imagers/sounders [4].

While these capabilities are feasible, they require fundamentally new approaches to plan, develop, and test what is now a Systems-of-Systems approach to remote sensing. In particular, these objectives have new impacts on the requirements for on-board science computation. Previously, the requirements for on-board processing were driven by increases in sensor fidelity and the move from snapshot to continuous observations [5]. These were largely static signal processing chains, and it was found that heterogeneous on-board processing solutions (i.e. combined multi-core CPU with customized accelerators) could deliver orders of magnitude higher processing power and meet these demands [6, 7, 8, 9].

This work was supported by the National Aeronautics and Space Administration (NASA) under grant 80NSSC17K0286. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of NASA or the U.S. Government.

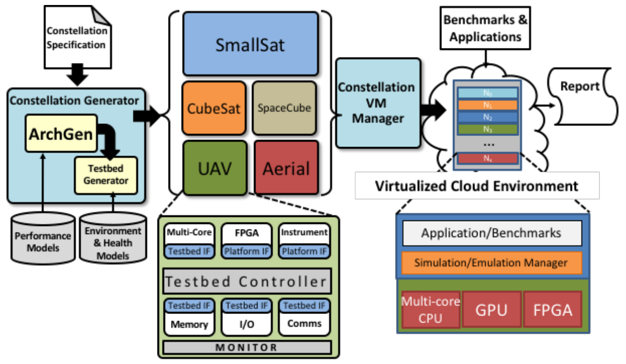

**Fig. 1**. Virtual Constellation Engine (VCE) framework to generate and evaluate constellations in a cloud environment

These new dynamic sensing capability objectives stress different computational requirements. Autonomy, either to capture dynamic science events, or to execute sensor health preservation functions, invokes not only low latency science data product calculation, but also responsive context switching capabilities to react properly to external events in a safe and controlled manner. Distributed sensor webs invoke heterogeneous platforms with different processor types for satellites (radiation hardened processors and Field Programmable Gate Arrays (FPGAs)), airborne platforms (embedded GPUs), and in-situ sensors (IoT processors such as RaspberryPi). Porting science applications between these requires computer engineering level expertise due to GPU and FPGA performance optimizations and can take several months to complete.

The Virtual Constellation Engine (VCE), depicted in Figure 1, allows mission planners and systems engineers to perform evaluations of different constellations and sensor web implementations. VCE enables constellations of different formations to be emulated within a scalable cloud environment, where each platform can emulate a different on-board processing configuration. A wide variety of processors is sup-

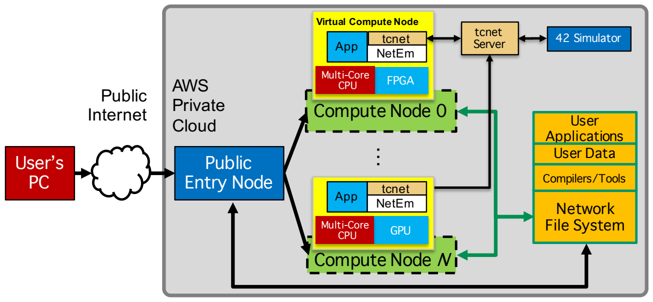

**Fig. 2**. Architecture of network emulation on AWS

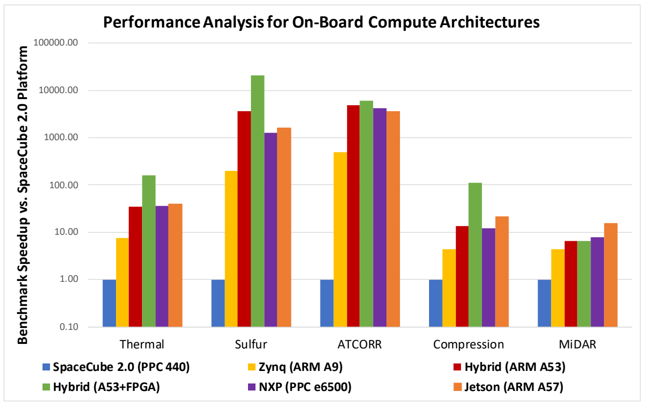

**Fig. 3**. VCE performance analysis on five representative benchmark applications on different compute platforms relative to a SpaceCube 2.0 system

ported and a novel Python based source to source translator is provided to support rapid application porting between processor types. The resulting system can emulate missions at the processor level and report to end users the overall processing throughput and power achieved on the emulated application. This is an important step to providing system-of-system level validation of system of system remote sensing applications.

## 2. DESIGN AND EVALUATION

The Virtual Constellation Engine (VCE), from Figure 1, provides a framework for mission designers and systems engineers to rapidly define heterogeneous constellations and their processing payloads and execute experiments to validate autonomous and dynamic objectives. By leveraging the cloud, constellations or sensor webs of arbitrary size can be explored, and multiple candidates can be evaluated in parallel.

An end user defines the constellation level topology and movement using *42*, a general purpose orbit dynamics simulator [10]. VCE then utilizes a Virtual Constellation generator to spawn constellation platforms in a cloud environment and emulate their performance. The computing payload of each platform is defined by the On-board Computing Framework, where specific processor and memory types, and topologies are defined. VCE then executes simulators and emulators of these processors in the cloud environment. The final piece in this architecture is to map specific remote sensing applications and autonomous algorithms onto these heterogeneous platforms. Hot & sPyC, a source to source translator which facilitates porting Python based applications between customized accelerators such as GPUs and FPGAs was developed to support such platform migration [11].

The output of VCE contains reports identifying the processing throughput of the constellation and its nodes, highlighting stalls that can translate into missed opportunities to capture dynamic science events. This system enables a scientist to evaluate applications spanning multiple satellites and with different instruments to create, assess, or refine complex applications in a sensor web environment.

### 2.1. Virtual Constellation Generator

In order to run a virtual constellation, end users leverage VCE's virtual constellation generation framework. For a virtual constellation, multiple heterogeneous nodes can be specified through the constellation specification parameters (number and type of resources, position, and data-rates) and integrate with performance, environment and health models, shown in Figure 1. The resources are implemented using Amazon's Web Services (AWS) through a Constellation Virtual Machine (VM) Manager that allows applications and emulation data to be distributed across the platform. AWS provides a flexible framework that allows the constellations to scale to hundreds or thousands of nodes to represent future large scale systems. Furthermore, after a node has been specified any developer can simply instantiate one or more of these nodes and immediately begin running their experiments in the Cloud Environment, shown in Figure 2. Users log into a public entry node in the AWS private cloud created through the Constellation VM Manager and provides direct access to a Network File System (NFS) where the user's application, data, and tools to run experiment, collect and analyze results. These virtual platforms dramatically increase the development capabilities by removing the hurdle of physical access or manual maintenance of the system. VCE includes support for the different compute resources made available by AWS, such as multi-core CPUs, GPUs and FPGAs, by providing an intermediate Application Programming Interface (API) to simplify exchanging data between the different resources.

### 2.2. On-board Computing Framework

At the core of our framework is *ArchGen*, which generates platforms based on user input parameters to select processor type, accelerator type, memory, interfaces, and sensor parameters. VCE provides a number of on-board compute re-

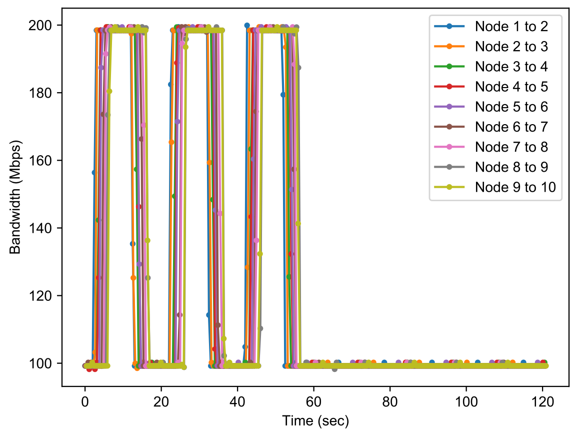

**Fig. 4**. Emulation of network bandwidth in an AWS cluster

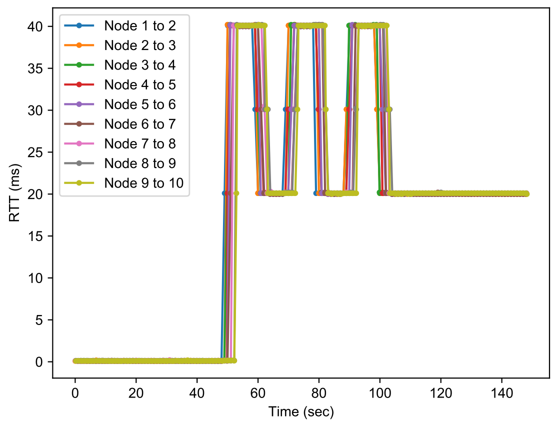

**Fig. 5**. Emulation of network latency in an AWS cluster

sources, such as multi-core CPUs (ARM A9, A53, PowerPC 440), FPGAs (Xilinx Zynq, UltraScale, Zynq UltraScale+) and now provides support for GPUs. GPUs include both server class and embedded class platforms which can be found in Aerial and UAV platforms. This has a tremendous advantage in that it is now possible to model distributed Earth sensing applications where portions of the data is collected at different levels of precision or coverage. The end user codes their application and VCE distributes the application across one or more GPU platforms in an AWS cloud environment. This is accomplished through a socket-based API to distribute data to either a GPU or FPGA, task these compute resources to perform the computation, and collect the results. The framework is abstracted from the developer such that data sent to the platform emulates a sensor running in the system. To achieve realistic data-rates the platform incorporates performance and health models as input specifications for each sensor.

VCE's framework allows for rapid porting and analysis of benchmark applications on different compute platforms. The platform evaluation process is useful for determining which systems should be included in a virtual constellation. Figure 3 shows the relative performance for five benchmarks on different platform types, all generated using *ArchGen*. The applications represent hyperspectral imaging computational kernels that would be tightly coupled with the compute resource to provide high-performance distributed computing in a sensor web environment. By providing a developer performance analysis early in the development process based on different architectures one can quickly select platforms that best suite performance needs.

## 2.3. Distributed Communications and Control

Constellations require the intertwined deployment of computation, communication and control strategies. An end user should be able to not only specify the computational capacity of a system, as previously described with the Constellation VM Manager, but the communication and control based on environmental or flight mission scenarios. This means to provide an accurate emulation of the operation of a constellation, we must emulate not only the heterogeneous computing devices (CPU, GPU, FPGA), but also the networking environment experienced by each node. To achieve this goal, our framework leverages the `NetEm` component [12] of the Linux kernel to control delay, bandwidth, and packet loss probability between each pair of nodes in an AWS EC2 cluster. To orchestrate changes of these networking parameters, we develop a client/server tool called `tcnet`: each node of the satellite constellation (emulated on EC2) periodically polls a control server, receives current parameters, and issues commands necessary to produce changes in the Linux kernel configuration.

As illustrated in Figure 2, the nodes of the emulated satellite application (*App*) communicate with each other through Linux, where networking parameters are managed by `tcnet` and enforced by `NetEm`. The communication between `tcnet` clients and the server is not subject to delay or bandwidth limitations, and it can access all of the available networking resources of AWS EC2 (possibly exchanging control information on a separate network interface). By controlling network parameters at the IP level, we can accurately reproduce the effects of latency and bandwidth limitations on applications and higher-layer network protocols such as TCP [13] with minimal overhead (networking emulation is performed in the kernel space).

In addition, a central control server allows us to easily integrate *arbitrary models* for the evolution of network parameters of the satellite constellation over time; in particular, it enables the use of network parameters computed from the output of simulators such as the *42* [10], orbital dynamics simulator. This feature is particularly important to emulate the variable delay of inter-satellite communications in low Earth orbits (LEO), or the intermittent characteristics of communications between satellites and ground stations, UAVs and airborne platforms.

To evaluate the effectiveness of our network emulation solution, we run a benchmark scenario on AWS. We start $n = 10$ virtual machine instances and run the `tcnet` client program to fetch and apply, at time intervals of 1 second, the network parameters from a `tcnet` server running on a separate virtual machine. We consider the linear topology $1 \to 2 \to \cdots \to n$ and measure, at each time instant, the round-trip time (RTT) from VM $i$ to VM $i + 1$ for all $0 < i < n$ (through simple `ping` requests). The AWS VM type is `c5.large` (2 vCPU, 4 GB of RAM, 0.7 Gbps bandwidth with bursts up to 10 Gbps) and the operating system is Ubuntu Linux 18.04 for all VMs.

In the experiment we use a network model with static schedule to drive the `tcnet` server: every 10 seconds, we change the available bandwidth. As Figure 4 illustrates, we change the bandwidth between each pair of nodes from 100 Mbps to 200 Mbps to emulate bandwidth bottlenecks similar to sensor overflow, loss of downlink, or when the computation cannot keep pace with the input data. TCP bandwidth is measured over time using the `iperf3` tool. The results highlight precise control over assigned bandwidth. In addition, `tcnet` also allows control of packet loss probability and packet corruption probability. Likewise, a delay between each pair of nodes (in each direction) from 20 ms to 10 ms experiment is performed. This benchmark allows us to test the reaction time of our network emulation layer; as illustrated in Figure 5, the application of network parameters is almost instantaneous.

### 2.4. Application Mapping

While targeting customized accelerators yields high real time performance, programming and optimizing heterogeneous resources such as FPGAs and GPUs can add several months to the software development flow. In order to curtail these long development tails we developed Hot & sPyC [14, 11], an open-source infrastructure and tool suite for integrating FPGA accelerators in Python applications. The aim of Hot & sPyC is to ease the packaging, integration, and binding of accelerators and their C/C++ based drivers callable from a Python application. This flow allows a developer to take existing Python code and quickly compile computationally complex functions into efficient FPGA designs, reducing the development time down to hours or days rather than months or years. The platform has been demonstrated on image processing applications and shown to achieve significant performance gains of up to 39,137$\times$ in hardware compared to standard software running on an ARM A9 processor [11]. Hot & sPyC significantly improves development productivity by allowing developers quick software simulation validation of their algorithms using industry standard simulation tools, such as ModelSim, while still leveraging high-level languages for benchmark creation and checking results.

## 3. CONCLUSION AND FUTURE WORK

The Virtual Constellation Engine (VCE) represents a framework prototype which enables remote sensing architects to rapidly posit and experiment with different satellite constellation or sensor web configurations. Ongoing work is investigating different methodologies by which designers can frame constellation level experiments and define which metrics they wish to observe. VCE is also continuously being upgraded to support additional processor types. The authors intend to make the software publicly available to support the remote sensing community more broadly.

## 4. REFERENCES

[1] National Academies of Sciences, Engineering, and Medicine, “Thriving on our changing planet: A decadal strategy for earth observation from space,” http://sites.nationalacademies.org/DEPS/esas2017/DEPS_169443.

[2] M. Little, “Distributed measurements and spacecraft missions,” in *Earth Science Technology Forum*, 2018, https://esto.nasa.gov/files/AIST/M03.R8.01_Little_v0.pdf.

[3] W. Blackwell, “Time-resolved observations of precipitation structure and storm intensity with a constellation of smallsats,” MIT Lincoln Laboratory, 2016.

[4] National Research Council, “Landsat and beyond: Sustaining and enhancing the nation’s land imaging program,” The National Academies Press, 2013.

[5] The National Academies of Sciences, Engineering, and Medicine, “A framework for analyzing the needs for continuity of nasa-sustained remote sensing observations of the earth from space,” 2013.

[6] A. G. Schmidt, G. Weisz, M. French, T. Flatley, and C. Y. Villalpando, “SpaceCubeX: A framework for evaluating hybrid multi-core CPU/FPGA/DSP architectures,” in *IEEE Aerospace Conference*, 2017.

[7] M. French, A. G. Schmidt, G. Weisz, T. Flatley, G. Crum, J. Bobblit, C. Y. Villalpando, and R. Bocchino, “SpaceCubeX: Initial simulation-level results of hybrid on-board processing architectures,” *NASA Earth Science Technology Forum*, June 2016.

[8] C. Y. Villalpando, R. Werner, J. Carson, G. Khanoyan, R.A. Stern, and N. Trawny, “A hybrid FPGA/tilera compute element for autonomous hazard detection and navigation,” in *IEEE Aerospace Conference*, 2013.

[9] N. Trawny, J.M. Carson, A. Huertas, M.E. Luna, V.E. Roback, A.E. Johnson, K.E. Martin, and C.Y. Villalpando, “Helicopter flight testing of a real-time hazard detection system for safe lunar landing,” in *AIAA Space*, 2013.

[10] “42: A Comprehensive General-Purpose Simulation of Attitude and Trajectory Dynamics and Control of Multiple Spacecraft Composed of Multiple Rigid or Flexible Bodies,” .

[11] S. Skalicky, J. Monson, A. G. Schmidt, and M. French, “Hot & Spicy: Improving Productivity with Python and HLS for FPGAs,” in *IEEE International Symposium on Field Programmable Custom Computing Machines (FCCM)*, 2018.

[12] “`NetEm` Homepage,” wiki.linuxfoundation.org/networking/netem.

[13] Stephen Hemminger, “Network emulation with NetEm,” in *Linux Conference Australia*, 2005.

[14] A. G. Schmidt, G. Weisz, and M. French, “Evaluating Rapid Application Development with Python for Heterogeneous Processor-based FPGAs,” in *IEEE International Symposium on Field Programmable Custom Computing Machines (FCCM)*, 2017.