EMULATING AND VERIFYING SENSING, COMPUTATION, AND COMMUNICATION IN DISTRIBUTED REMOTE SENSING SYSTEMS

Matthew French,[1] Marco Paolieri,[2] Vivek V. Menon,[1] and Andrew G. Schmidt[1]
[1]Information Sciences Institute, University of Southern California
[2]Department of Computer Science, University of Southern California
mfrench@isi.edu, paolieri@usc.edu, vivekv@isi.edu, aschmidt@isi.edu

ABSTRACT

This paper presents updates to the Virtual Constellation Engine (VCE), a simulator and emulator which enables modeling of sensing, computation, and communication of multi-platform remote sensing systems. Users can launch heterogeneous constellations, describe different on-board processing configurations, and simulate and verify autonomous operations. Updates in the past year include emulation of on-board computing utilizing cloud based processors, inclusion of the Delay Tolerant Networking (DTN) protocol in the communication modeling, and visualization and diagnostic tools which facilitate analysis of the distributed system. To illustrate the benefits, VCE was utilized in the development of on-board processing implementation of a new, high data rate sensor, Multispectral, Imaging, Detection, and Active Reflectance (MiDAR), for both an embedded Nvidia Tegra GPU and a Xilinx MPSoC FPGA achieving a 7.6x speedup over a desktop workstation and enabling this application for integration into a distributed sensing system.

Index Terms— Remote sensing, distributed systems, constellation, sensor web, on-board processing, MIDAR, DTN.

1. INTRODUCTION

Remote sensing systems are experiencing a bredth of technology advances, leading to a renaissance in how these missions can be conceived, realized, and deployed. UAVs, CubeSats and SmallSats equipped with science-quality instruments, increased on-board processing capabilities, and machine learning techniques now permit handling large volumes of data. These increased capabilities and reduced costs have led to concepts such as massively scaled constellations of satellites and distributed measurement systems involving a mixture of heterogeneous ground, UAVs, airborne, and satellite based sensors. Several organizations are now planning or implementing satellite constellations with over 1,000 nodes targeting global continuous missions for internet provision, weather sensing, broadband communications, and other applications. Distributed measurement missions are receiving increased attention due to the realization that the ability to capture dynamic observations of transient events has a drastic impact in creation of low latency data products for emergency events such as floods, fires, or volcanic eruptions and the ability to capture higher quality science products for hydrology, cryosphere, biodiversity, weather and more [1, 2].

These capabilities require fundamentally new approaches to plan, develop, and test what is now a Systems-of-Systems approach to remote sensing. High performance on-board computing is a critical requirement in distributed measurement systems. Traditional transmission of all raw data to the ground and the delays with postprocessing and redistributing it, are no longer viable. Emerging high fidelity sensors, such as Multispectral Imaging, Detection, and Active Reflectance (MiDAR) [3, 4], are producing volumes of data that exceed the communication link capacity, even when compressed.

Additionally, to support emergencies or capture of dynamic science events, on-board processing is needed to more rapidly detect the event and task other sensing platforms to respond. Typically, lossless data compression achieves about 3x data reduction, whereas calculation of Level 1 science products achieves 10 to 10,000x data reduction. On-board processing enables higher level data representing the event of interest to move rapidly through a distributed system, enabling coordination with other sensing instruments in time scales needed for mission success.

Accurately simulating and modeling the sensing, on-board science computation, and communication of every node in a constellation or distributed measurement system, along with its position and trajectory, are needed to ensure a distributed system will operate as expected. The Virtual Constellation Engine (VCE) presented here allows mission planners and systems engineers to perform evaluations of different constellations and sensor web implementations.

2. VIRTUAL CONSTELLATION OVERVIEW

The Virtual Constellation Engine (VCE), depicted in Figure 1, allows mission designers to perform evaluations of different

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.

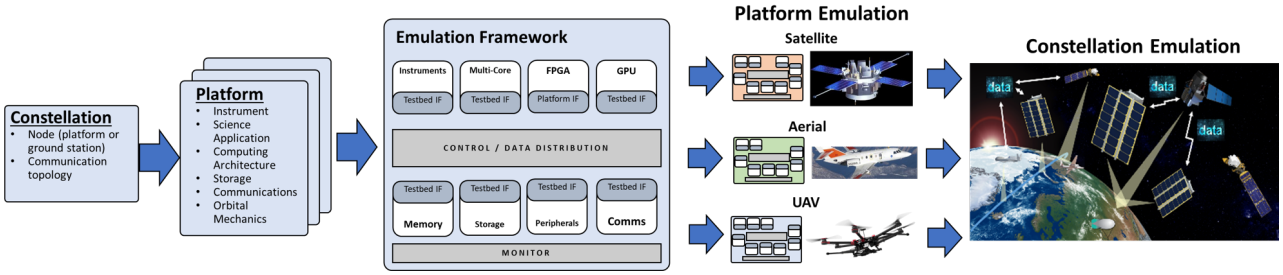

Fig. 1. Virtual Constellation Engine (VCE) flow to generate and evaluate constellations in a cloud environment

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 supported and a novel Python based source to source code 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.

To use VCE, an end user defines the orbital elements (as two-line element sets) of the satellites of the constellation, the coordinates (latitude/longitude) of each base station, and the AWS virtual machine type of each node (e.g., `p3.2xlarge` for nodes equipped with a GPU or `f1.2xlarge` for nodes equiped with an FPGA).

Each platform is defined as an instrument, which provides input data, an onboard processor which executes the science application, and the communications transceiver, which provides the output. The platforms are provided as virtual machine images (using Amazon Machine Images), together with a script of commands that VCE runs at startup. From the configuration of the constellation (provided as an input YAML file), the command-line tool of VCE generates an AWS CloudFormation template (also in YAML format) with a complete and reproducible setup of the cloud stack; this includes the provisioning and configuration of VM nodes, routers, network interfaces, and firewalls. The user can then start and manage the cloud stack of the constellation with the standard AWS tools.

When the cloud stack is started, VCE agents running on each node periodically update networking parameters (based on satellite orbits) and collect time series of measurements (using `collectd` [5]) including CPU, memory, disk, and network usage; these metrics allow the user to quantify the processing throughput of the constellation and of each node, highlighting stalls that can translate into missed opportunities to capture dynamic science events.

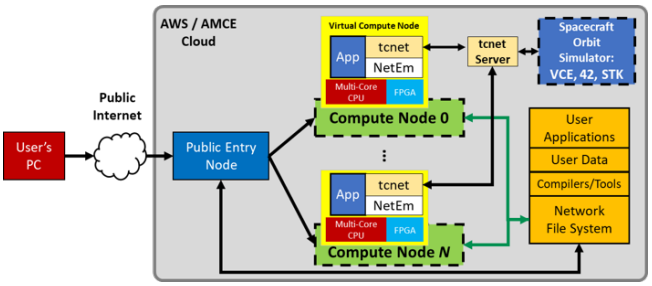

Fig. 2. VCE AWS Architecture

2.1. Virtual Constellation Generator

In order to run a virtual constellation, end users leverage VCE's virtual constellation generation framework. Resources are provisioned on Amazon's Web Services (AWS) through AWS CloudFormation, to obtain a fully-defined and reproducible cloud stack. AWS provides a flexible framework that allows the constellations to scale to hundreds or thousands of nodes, representing future large-scale systems. Furthermore, any developer can simply instantiate one or more of the nodes of the constellation and immediately connect over SSH and begin developing or testing payload applications, as shown in Figure 2. These virtual platforms dramatically increase the development capabilities by removing the hurdle of physical access or manual maintenance of the system.

2.2. Distributed Communications and Control

Constellations require the intertwined deployment of computation, communication, and control strategies. The end user needs to be able to specify not only the computational capacity of a system, but also its communication parameters and controlled inputs based on environmental or flight mission scenarios. To support the accurate emulation of the operation of a constellation, VCE emulates not only the heterogeneous computing devices, but also the networking environment experienced by each node. To achieve this goal, VCE leverages the `NetEm` component [6] of the Linux kernel to control delay, bandwidth, and packet loss probability between each pair of nodes in the AWS EC2 cluster of the constella-

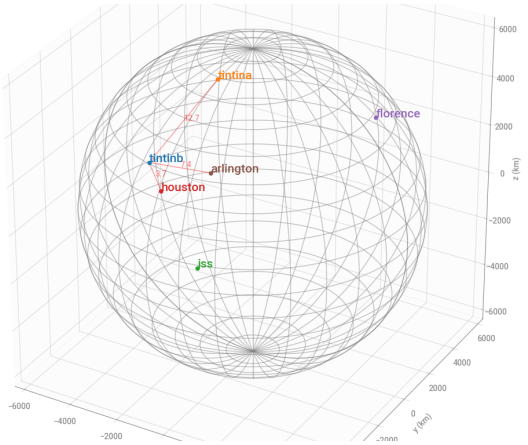

Fig. 3. Orbits computed by VCE for bandwidth/latency emulation. Nodes with line of sight are connected by a red line annotated with the corresponding communication latency.

tion. To orchestrate changes of these networking parameters, each node runs a client program (`tcnet`) which periodically polls a control server, receives current parameters, and issues commands necessary to produce changes in the Linux kernel configuration of `NetEm`.

As illustrated in Figure 2, the nodes of the emulated satellite application (*App*) communicate with each other through the standard TCP/IP stack of Linux, where networking parameters are managed by `tcnet` and enforced by `NetEm`. By controlling network parameters at the IP level, we can accurately reproduce the effects of satellite latency and bandwidth limitations on applications and higher-layer network protocols such as TCP [7] with minimal overhead.

A new higher-layer network protocol supported is Delay/Disruption Tolerant Networking (DTN). VCE provides support for the configuration, deployment and testing of (DTN) protocols [8]. On startup, VCE installs the ION-DTN protocol implementations [9] (including the DTN Bundle Protocol, the Licklider Transmission Protocol, and two CCSDS File Delivery Protocols) on each cloud node; their configurations can be defined by the user in VCE using programming constructs such as `for` loops over the nodes, their names, and assigned IP addresses on AWS. Programmable configuration tools are especially important to run cloud experiments to constellations with a very large (or variable) number of nodes.

Satellite orbitology used within the constellation simulation can be defined using different methods. For fast, coarse simulation, VCE computes orbits from two-line element sets specified by the user in the configuration. For more accurate orbits, VCE can import orbitology from 3rd party orbital dynamics simulators such as *42* [10] or the *Systems Tool Kit* [11]. Before starting the cloud emulation, the user can review the 3D animated satellite orbits to validate the testing scenario of the mission, Figure 3; during the animation, nodes (satellites or base stations) are connected when line-of-sight communication is available between them.

Finally, additional control signals and instrument parameters varying over time can be imported into VCE as time series; when cloud execution is started by the user, applications running on each node are able to read the current value of these parameters from YAML files updated by the VCE framework.

2.3. On-board Computing Emulation

At the core of VCE is the onboard computing emulator, which generates compute platforms based on user input parameters to select processor type, accelerator type, memory, interfaces, and sensor parameters. A board model is generated from the user inputs, which is then used to generate the appropriate board level simulation from processor simulators and emulators. Simulation is provided through either QEMU for processors or hardware simulators such as VSIM for FPGAs. Emulation is performed by operating on processors resident within the cloud environment. Specialized governors are used to throttle back the clock speed, bus width, number or cores, or other features of the state of the art processors in the cloud to more accurately reflect the capabilities of embedded processors utilized in remote sensing systems. [12] describes the hardware emulation in more detail. Table 1 details all of the processors currently supported by VCE.

Table 1. VCE Supported Processors

Processor Type	Processors	Devices or Platform
CPU	ARM A9, A53	HPSC, FPGA SoCs
CPU	PowerPC440	RAD750, NXP
FPGA	Xilinx Virtex series, Ultrascale, MPSOC	SpaceCube and others
FPGA	Microsemi RTG4	Several
GPU	Nvidia Volta, Tegra	Server and mobile

VCE distributes the application developed by end users 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 an instrument running in the system.

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 [13, 14], an open-source infrastructure and tool suite for integrating FPGA accelerators in Python applications. This flow allows a developer to take existing Python code and quickly compile computationally complex functions into efficient

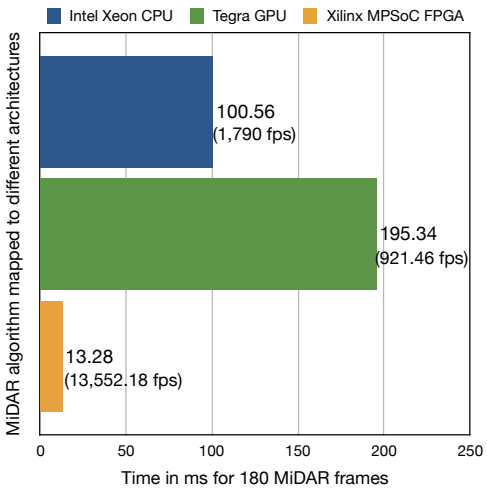

Fig. 4. MiDAR results for all the architectures

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 39,137x in hardware compared to standard software running on an ARM A9 processor [14].

2.4. Application Mapping

To demonstrate the capabilities and value of VCE, an advanced instrument and application, Multispectral, Imaging, Detection, and Active Reflectance (MiDAR), was evaluated for suitability in a distributed measurement mission. MiDAR is currently used in coral reef erosion studies, and has broader applicability to atmospheric correction, multispectral mapping, mineral detection, and optical communication [4]. The original MiDAR prototype is UAV based. The platform collects raw data until the hard drives are full and data is post processed on the ground. VCE was used to evaluate the viability of processing the MiDAR receiver algorithm both on-board the UAV on its resident Nvidia Tegra GPU and on a satellite on a SpaceCube3 board with Xilinx MPSoC FPGAs [15].

The summarized results for the platforms are shown in Figure 4 for 180 MiDAR frames. The MiDAR instrument currently has a sampling rate of 140fps, but has increased SNR with higher frame rates and has a goal of acheiving 2,500fps. The Intel Xeon CPU, the base AWS processor, achieves a throughput of 1,790 fps. The Tegra GPU emulation for UAV is slower due to memory transfers and being an embedded processor. The FPGA fabric implementation is the fastest and has highest throughput as it can have multiple MiDAR datapaths to compute frames in parallel. The Tegra GPU can support up to 921 fps whereas the FPGA can support up to 13,552 fps, both of which are sufficient for supporting MiDAR in distributed measurement missions.

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 has been open-sourced and can be found at: `https://isi-rcg.github.io/vce`.

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] Ved Chirayath and Sylvia A. Earle, "Drones that see through waves – preliminary results from airborne fluid lensing for centimetre-scale aquatic conservation," *Aquatic Conservation: Marine and Freshwater Ecosystems*, vol. 26, no. S2, pp. 237–250, 2016.

[4] Ved Chirayath and Alan Li, "Next-Generation Optical Sensing Technologies for Exploring Ocean Worlds—NASA FluidCam, MiDAR, and NeMO-Net," *Frontiers in Marine Science*, vol. 6, Sep 2019.

[5] "Collectd homepage," `https://collectd.org/`.

[6] "`NetEm` Homepage," `wiki.linuxfoundation.org/networking/netem`.

[7] Stephen Hemminger, "Network emulation with NetEm," in *Linux Conference Australia*, 2005.

[8] "Ietf drafts of delay/disruption tolerant networking protocols," `https://datatracker.ietf.org/wg/dtn/documents/`.

[9] "Ion-dtn homepage," `https://sourceforge.net/projects/ion-dtn/`.

[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] "`Systems Tool Kit` Website," `https://agi.com/products/satellite-design-and-operations`.

[12] 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 *2017 IEEE Aerospace Conference*, March 2017, pp. 1–10.

[13] 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.

[14] 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.

[15] Alessandro Geist, Cody Brewer, Milton Davis, Nicholas Franconi, Sabrena Heyward, Travis Wise, Gary Crum, David Petrick, Robin Ripley, Christopher Wilson, et al., "SpaceCube v3.0 NASA Next-Generation High-Performance Processor for Science Applications," in *33rd Annual AIAA/USU Conference on Small Satellites*, August 2019.