Cloud Computing for Mission Design and Operations

J. Arrieta    J. Senent

Jet Propulsion Laboratory, California Institute of Technology

SpaceOps 2014
Pasadena, CA
May 5–9, 2014

Copyright 2014 California Institute of Technology
Government Sponsorship Acknowledged
Recalling Our Previous Message\textsuperscript{1} — A Vision to Reduce Complexity

<table>
<thead>
<tr>
<th>Sources of Complexity</th>
<th>Architectural Evolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proliferation of systems</td>
<td>Associative Elasticity</td>
</tr>
<tr>
<td>Proliferation of formats</td>
<td>Semantical Hyperdata</td>
</tr>
<tr>
<td>Proliferation of protocols</td>
<td>Living Workflows</td>
</tr>
<tr>
<td>Siloing of datasets</td>
<td>Abstraction</td>
</tr>
</tbody>
</table>

**Summary of Architecture**

Expose data and algorithms as resource-oriented Web services, coordinated via messaging and running on virtual machines

- Virtual Machine
- HTTP Server
- Database
- Message Broker
- Serialization Format
- Web Application
- Web Client

\textsuperscript{1}Arrieta, J. et al. *Cloud Computing for Mission Design and Operations* SpaceOps 2012 Conference
A Vision to Reduce Complexity (continued)

The Message

- We recognize the value of the cloud computing model, and would like to capture its benefits
- Valid concerns prevent widespread and expedited adoption
- It is possible to expedite adoption by internally adopting the cloud computing philosophy

The Roadmap

- Adopt a common data interchange format → make data usable
- Evolve raw data to semantical hyperdata → increase information content
- Decompose workflows into simple actions → granular, reusable units of work
- Categorize data and algorithms as resources → addressable, independent entities
- Provide a common interface to resources via HTTP → access standardization
- Coordinate the system interaction via messaging → living, adaptable workflows
- Deploy worker and data nodes in virtual machines → abstract, elastic, configurable system

![Diagram of database, unit of work, and interactions](image)

J. Arrieta, J. Senent
Cloud Computing for Mission Design and Operations
How Does Great Software Look Like? — An Independent Case

SPICE\textsuperscript{2}

- Supported thousands of users—worldwide—over many years
- One-of-a-kind, critical capability
- Professional development team
- Impeccable source code (seriously! Take a look!)
- Extensive training and documentation
- Over 50 different platform combinations!

\textsuperscript{2}NASA’s Navigation and Ancillary Information Facility http://naif.jpl.nasa.gov/naif/
How Does Great Software Look Like? (continued)

WebGeocalc

- 80/20: time conversions and geometry calculator, event finder
- Does eighteen things really well — 18 units of work
- One platform — the browser
- Audience claps at the end of their demos (I did)

How Does Great Software Look Like? (continued)

J. Arrieta, J. Senent  
Cloud Computing for Mission Design and Operations

<table>
<thead>
<tr>
<th>Calculation</th>
<th>TDT Calendar Date</th>
<th>Distance (km)</th>
<th>X (km)</th>
<th>Y (km)</th>
<th>z (km)</th>
<th>dST1 (km)</th>
<th>dST1 (km)</th>
<th>dST1 (km)</th>
<th>Time at Target</th>
<th>Light Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-01 00:00:00.0000001</td>
<td>0.0000001</td>
<td>30.0000000</td>
<td>127.18484</td>
<td>4.1276</td>
<td>1573.166</td>
<td>9.3349678</td>
<td>20.7064417</td>
<td>-5.0789496</td>
<td>-2.20163762</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-01 00:00:00.0000001</td>
<td>0.0000001</td>
<td>30.0000000</td>
<td>127.18484</td>
<td>4.1276</td>
<td>1573.166</td>
<td>9.3349678</td>
<td>20.7064417</td>
<td>-5.0789496</td>
<td>-2.20163762</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-02 00:00:00.0000002</td>
<td>0.0000002</td>
<td>30.0000000</td>
<td>127.18484</td>
<td>4.1276</td>
<td>1573.166</td>
<td>9.3349678</td>
<td>20.7064417</td>
<td>-5.0789496</td>
<td>-2.20163762</td>
</tr>
<tr>
<td>4</td>
<td>2014-01-02 00:00:00.0000002</td>
<td>0.0000002</td>
<td>30.0000000</td>
<td>127.18484</td>
<td>4.1276</td>
<td>1573.166</td>
<td>9.3349678</td>
<td>20.7064417</td>
<td>-5.0789496</td>
<td>-2.20163762</td>
</tr>
<tr>
<td>5</td>
<td>2014-01-02 00:00:00.0000002</td>
<td>0.0000002</td>
<td>30.0000000</td>
<td>127.18484</td>
<td>4.1276</td>
<td>1573.166</td>
<td>9.3349678</td>
<td>20.7064417</td>
<td>-5.0789496</td>
<td>-2.20163762</td>
</tr>
</tbody>
</table>

Click and drag to zoom, shift-click and drag to pan. Double-click or use button to reset zoom level.

Distance vs. Time

Click and drag to zoom, shift-click and drag to pan. Double-click or use button to reset zoom level.

Speed vs. Time
Serialization — Plain Text

- It is hard to beat JSON\(^4\): readable, relatively lightweight, and relatively fast.

**Current SPICE Text Kernel Format** (example)

\[
\text{BODY399\_POLE\_RA} = ( 0. \quad -0.641 \quad 0. ) \\
\text{BODY399\_POLE\_DEC} = ( 90. \quad -0.557 \quad 0. ) \\
\text{BODY399\_PM} = ( 190.147 \quad 360.9856235 \quad 0. ) \\
\text{BODY399\_LONG\_AXIS} = ( 0. \quad ) \\
\text{BODY399\_RADII} = ( 6378.1366 \quad 6378.1366 \quad 6356.7519 )
\]

---

\(^4\text{http://json.org}\)
JSON Serialization (example)

{"id" : "Jupiter",
"components" : [
{""type": "GravitySource",
"gm" : 126686536.75178400},
{"type" : "TriaxialEllipsoid",
"radii" : [71492.0, 71492.0, 66854.0]},{
{""type" : "IAUPolePrimeMeridian",
"ra" : [268.056595, -0.006499],
"dec" : [64.495303, 0.002413],
"pm" : [284.95, 870.5360000],
"np_ra" : [0.000117, 0.000938, 0.001432, 0.000030, 0.002150],
"np_dec" : [0.000050, 0.000404, 0.000617, -0.000013, 0.000926],
"np_angles" : [[99.360714, 4850.4046],
[175.895369, 1191.9605],
[300.323162, 262.5475],
[114.012305, 6070.2476],
[49.511251, 64.3000]]}]}
HDF5\textsuperscript{5} is a strong, mature, high-performance solution, ideal for large datasets

Google Protocol Buffers\textsuperscript{6} is ideal for simple (small) messages, but not as convenient for numerical datasets

Apache Avro\textsuperscript{7} is modern, promising and competitive, but it is young (at least in C++)

\textsuperscript{5}http://www.hdfgroup.org/HDF5/
\textsuperscript{6}https://developers.google.com/protocol-buffers/
\textsuperscript{7}http://avro.apache.org/
/*
 * @file    hdf5-test05.cpp
 * @brief   Demonstration of ability of HDF5 to serialize large data structures.
 * @author  J. Arrieta <Juan.Arrieta@jpl.nasa.gov>
 */

#include<random>
#include<iostream>
#include<sstream>
#include<algorithm>
#include<vector>

#include<H5Cpp.h>

// A simple vector container capable of being serialized to and from HDF5
class SimpleVectors {
public:
    SimpleVectors(const std::string& filename)
    : m_filename(filename),
     m_fp(m_filename.c_str(), H5F_ACC_TRUNC) {
    }

    SimpleVectors& create_group(const std::string& name) {
        m_fp.createGroup(name.c_str());
        return *this;
    }

    template<typename Container>
    SimpleVectors& add_data(const std::string& name, const Container& data) {
        hsize_t dim[] = {data.size(), 1};
        H5::DataSpace dspace(dim);
        H5::DataSet dataset = m_fp.createDataSet(name.c_str(), H5::PredType::NATIVE_DOUBLE, dspace);
        dataset.write(data.data(), H5::PredType::NATIVE_DOUBLE);
        return *this;
    }

    ~SimpleVectors() {
        m_fp.close();
    }

private:
    std::string m_filename;
    H5::H5File m_fp;
};

int main() {
    // in this case we will store the dataset to disk. However, it
    // could just as easily be persisted into a buffer and sent
    // across a wire
    SimpleVectors sp("../data/test05.h5");

    sp.create_group("/Legs");

    std::vector<double> data;

    for(size_t k=0; k<100; k++) {
        std::stringstream ss;
        ss << "Legs/" << k;
        data.clear();
        std::generate_n(std::back_inserter(data), 1000000, std::rand);
        sp.add_data(ss.str(), data);
    }

    return 0;
}
// Sample application which uses the CNAV serialization provided by
// Protocol Buffers

// C++ Standard Library
#include<iostream>
#include<fstream>
#include<string>
#include<cnav.pb.h>

// automatically generated by `protoc`
using namespace std;

int main(int argc, char* argv[]){
    // add two OTMs to the Strategy
    cnav::Strategy strategy;
    cnav::OTM* otm1 = strategy.add_otms();
    cnav::OTM* otm2 = strategy.add_otms();

    otm1->set_name("281");
    otm1->set_epoch(0.0);
    otm1->set_window(cnav::OTM::PRIME);
    otm1->set_engine(cnav::OTM::MEA);
    otm1->set_frame(cnav::EME2000);
    otm1->set_magnitude(2.8563);
    otm1->set_ra(20.123);
    otm1->set_dec(0.6543);

    otm2->set_name("282");
    otm2->set_epoch(otm1->epoch() + 86400.0 * 6); // six days after previous maneuver
    otm2->set_engine(cnav::OTM::MEA);
    otm2->set_frame(cnav::EME2000);
    otm2->set_magnitude(0.12345);
    otm2->set_ra(10.0);
    otm2->set_dec(20.0);

    // create a new Encounter
    cnav::Encounter* enc = strategy.mutable_encounter();
    enc->set_name("T81");
    enc->set_epoch(otm2->epoch() + 86400.0 * 3); // three days after approach maneuver
    enc->set_frame(cnav::EME2000);
    enc->set_bdotr(-1500.0);
    enc->set_bdotr(0.0);

    // write to durable storage
    fstream output("strategy.bin", ios::out | ios::trunc | ios::binary);
    strategy.SerializeToOstream(&output);

    // read from durable storage
    cnav::Strategy strategy2;
    fstream input("strategy.bin", ios::in | ios::binary);
    strategy2.ParseFromIstream(&input);

    std::cout<<"Count:	"<<strategy2.otms_size()<<std::endl;
    for(unsigned int k=0; k < strategy2.otms_size(); k++){
        const cnav::OTM* otm = strategy2.otms(k);
        std::cout<<"OTM-\n  "<<otm->name()<<std::endl;
    }
    return 0;
}
Apache Avro Sample Program

/*
 * file measurement.cpp
 * brief Demonstration of Apache's Avro for serializing composite data types.

This C++ program will read and write "Measurement" objects. The included header file "measurement.hpp" was created automatically by issuing the following command:

    avrocodegen -I measurement.json -o a measurement.hpp

where "measurement.json" is the file where I provided the data schema. Apart from the "measurement.json" file and the code contained here, I did not write anything else.

Compile this code with clang++, or g++. I used the following clang version:

Apple clang version 4.1 (tags/Apple/clang-421.11.66) (based on LLVM 3.1svn)
Thread model: posix

and the following compilation line

    clang++ -I measurement measurement.cpp -lavrocpp

Notice that I have installed the avro library beforehand.

Program output:
Station: JPL-01
Lat: 34.1996
Lot: 118.1747
Epoch: 123.4560
Temperature: 23.15

// C++ Standard Library
#include<iostream>

// Avro Library
#include<avro/Encoder.h>
#include<avro/Decoder.h>

// Local includes
#include "measurement.hpp" // this file was automatically generated

int main(){
    // create a new measurement - notice that I did NOT write this
    // class, it was automatically generated by avro (it contains the
    // getter and setter methods)
    Measurement my_measurement;

    // store some values in my measurement
    my_measurement.station = "JPL-01";
    my_measurement.lat = 34.1996;
    my_measurement.lon = 118.1747;
    my_measurement.epoch = 123.4560;
    my_measurement.temperature = 23.1500;

    // at this point I will work exclusively on memory, so I open a
    // memory output stream to which I will write the encoded data; I
    // want binary data, so I choose a binary encoder.
    std::auto_ptr<avro::OutputStream> out = avro::memoryOutputStream();
    avro::EncoderPtr e = avro::binaryEncoder();
    // initialize the encoder to write to the memory stream
    e->init("out");
    // encode the measurement into the memory stream
    avro::encode("e", my_measurement);

    // at this point I have serialized 'my_measurement' to a buffer in
    // memory, I could read it back into a new measurement object. To
    // this end, I create a memory input stream (mapped to the memory
    // output stream created earlier), and a binary decoder
    std::auto_ptr<avro::InputStream> in = avro::memoryInputStream("out");
    avro::DecoderPtr d = avro::binaryDecoder();
    // initialize the decoder to read from the memory stream
    d->init("in");

    // create a fresh instance of 'my_measurement'
    Measurement my_fresh_measurement;

    // read the decoded data into my_fresh_measurement
    avro::decode("d", my_fresh_measurement);

    // now print the fresh measurement to screen, just to confirm that
    // the data is what we expected
    stdout<<"Station: "<<my_fresh_measurement.station<<std::endl
    <<"Lat: "<<my_fresh_measurement.lat<<std::endl
    <<"Lon: "<<my_fresh_measurement.lon<<std::endl
    <<"Epoch: "<<my_fresh_measurement.epoch<<std::endl
    <<"Temperature: "<<my_fresh_measurement.temperature<<std::endl;

    return EXIT_SUCCESS;
}
### Our Attempt — The *Rocket Science Computing Platform*

| Database          | sqlite3                
|-------------------|------------------------
|                   | mongodb                |
| Linear Algebra    | Eigen                  
|                   | blas                   
|                   | LAPACK                 |
| Astrodynamics     | SPICE Toolkit          
|                   | CSPICE Toolkit         
|                   | hx                     |
| Compilers         | GCC (Fortran, C, and C++)  
|                   | LLVM/Clang (C and C++)  
|                   | Haskell                
|                   | Erlang                 |
| Scripting         | Python                 
|                   | Perl                   
|                   | Lua                    |
| Messaging         | RabbitMQ               
|                   | ZeroMQ                 |
| Web Server        | nginx                  
|                   | nodejs                 |
| Data Sources      | Generic Kernels        
|                   | DE421                  
|                   | JUP230                 |

- What if we distribute environments instead of software?
- What if everyone could collaborate in maintaining *the* platform?
- What if we focused on creating *with* the tools rather than creating *the* tools?
Visualization with \texttt{three.js} and \texttt{D3.js}

\texttt{http://threejs.org}
\texttt{http://d3js.org}
Visualization with three.js and D3.js (continued)
Visualization with three.js and D3.js (continued)