Event-driven workflows for large-scale seismic imaging in the cloud

Philipp A. Witte[†], Mathias Louboutin[†], Henryk Modzelewski[*],
Charles Jones[‡], James Selvage[‡] and Felix J. Herrmann[†]

SLIM

[†] Georgia Tech

[*] UBC THE UNIVERSITY OF BRITISH COLUMBIA

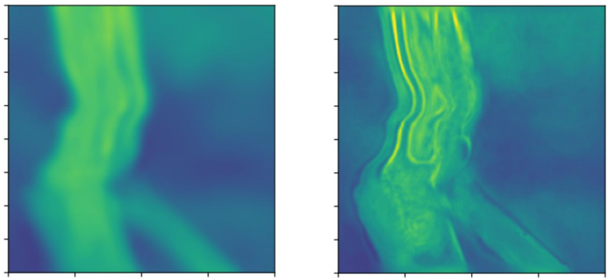
[‡] osokey

Disclaimer

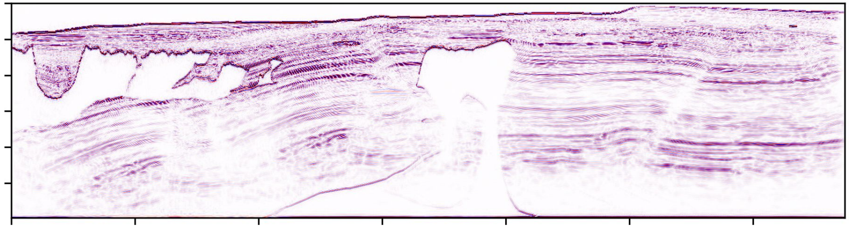
- The following work was developed using Amazon Web Services (AWS) and therefore contains terminology referencing AWS services and product names

- Technology presented in this talk is not tied to one specific cloud provider and has been replicated on other platforms

Seismic wave equation-based inversion

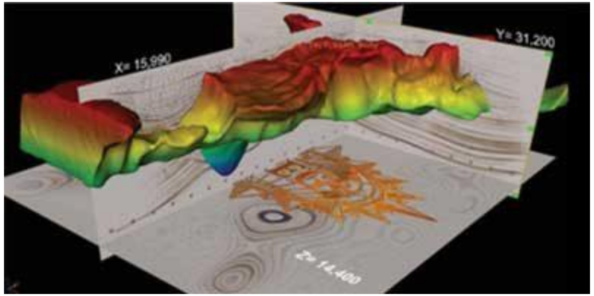


FWI



LS-RTM (< 120$)

- Solve wave equations for many sources
- Propagate over many wavelengths
- Massive data I/O
- Curse of dimensionality
- Infeasible for very large models (e.g. SEAM)
- Requires HPC environments



SEAM (?$)

Seismic inversion on HPC clusters

Conventional compute environment: **HPC clusters**

✓ **Pros**

- Best achievable performance
- 40+ years of experience and existing software
- Low mean-time-between failures (MTBF)
- Very fast inter-node connections possible (Infiniband)

✗ **Cons**

- Very high upfront + maintenance costs
- Only available to few companies + academic institutions
- Compromises regarding hardware (architecture/CPUs/GPUs/RAM)

Seismic inversion in the cloud

SLIM

Cloud computing

amazon web services™

Microsoft Azure

Google Cloud Platform

✓ Pros

- Theoretically unlimited scalability
- High flexibility (hardware, jobs)
- No upfront + maintenance costs: pay-as-you-go
- Available to anyone
- Latest hardware and architectures available (GPUs, ARM)

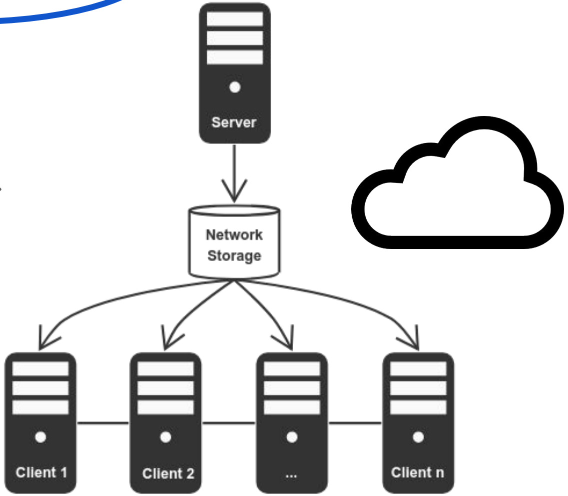
✗ Cons

- Slower inter-node connections (depending on platform)
- Oftentimes larger MTBF
- High costs if not used properly
- Need to transition software
- Steep learning curve

Moving to the cloud

```c
#include <inttypes.h>
#include <sys/time.h>
#include <math.h>
struct profiler
{
  double loop_stencils_a;
  double loop_body;
  double kernel;
} ;
struct flops
{
  long long loop_stencils_a;
  long long loop_body;
  long long kernel;
} ;
extern "C" int ForwardOperator(double *m_vec, double *u_vec, double *damp_vec, double *src_vec, float
*src_coords_vec, double *rec_vec, float *rec_coords_vec, long i1block, struct profiler *timings, struct flops *flops)
{
  double (*m)[280] = (double (*)[280]) m_vec;
  double (*u)[280][280] = (double (*)[280][280]) u_vec;
  double (*damp)[280] = (double (*)[280]) damp_vec;
  double (*src)[2] = (double (*)[2]) src_vec;
  float (*src_coords)[2] = (float (*)[2]) src_coords_vec;
  double (*rec)[101] = (double (*)[101]) rec_vec;
  float (*rec_coords)[2] = (float (*)[2]) rec_coords_vec;
  {
    struct timeval start_kernel, end_kernel;
    gettimeofday(&start_kernel, NULL);
    int t0;
    int t1;
    int t2;
    ;
    for (int i3 = 0; i3<3; i3+=1)
    {
      flops->kernel += 2.000000;
      {
        {
          t0 = (i3)%(3);
          t1 = (t0 + 1)%(3);
          t2 = (t1 + 1)%(3);
```


Lift and shift

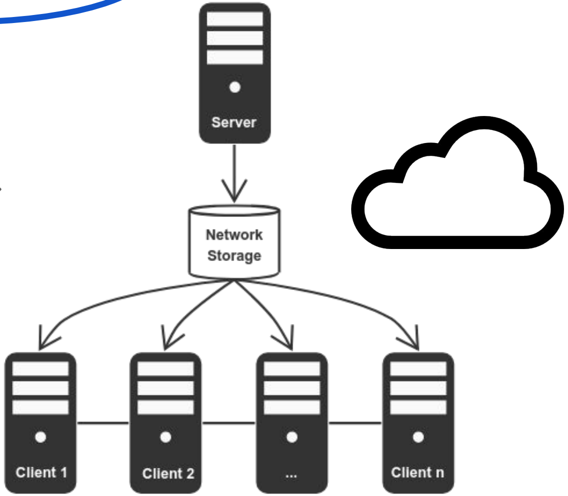


Moving to the cloud

```c
#include <inttypes.h>
#include <sys/time.h>
#include <math.h>
struct profiler
{
  double loop_stencils_a;
  double loop_body;
  double kernel;
} ;
struct flops
{
  long long loop_stencils_a;
  long long loop_body;
  long long kernel;
} ;
extern "C" int ForwardOperator(double *m_vec, double *u_vec, double *damp_vec, double *src_vec, float
*src_coords_vec, double *rec_vec, float *rec_coords_vec, long i1block, struct profiler *timings, struct flops *flops)
{
  double (*m)[280] = (double (*)[280]) m_vec;
  double (*u)[280][280] = (double (*)[280][280]) u_vec;
  double (*damp)[280] = (double (*)[280]) damp_vec;
  double (*src)[2] = (double (*)[2]) src_vec;
  float (*src_coords)[2] = (float (*)[2]) src_coords_vec;
  double (*rec)[101] = (double (*)[101]) rec_vec;
  float (*rec_coords)[2] = (float (*)[2]) rec_coords_vec;
  {
    struct timeval start_kernel, end_kernel;
    gettimeofday(&start_kernel, NULL);
    int t0;
    int t1;
    int t2;
    ;
    for (int i3 = 0; i3<3; i3+=1)
    {
      flops->kernel += 2.000000;
      {
        {
          t0 = (i3)%(3);
          t1 = (t0 + 1)%(3);
          t2 = (t1 + 1)%(3);
```

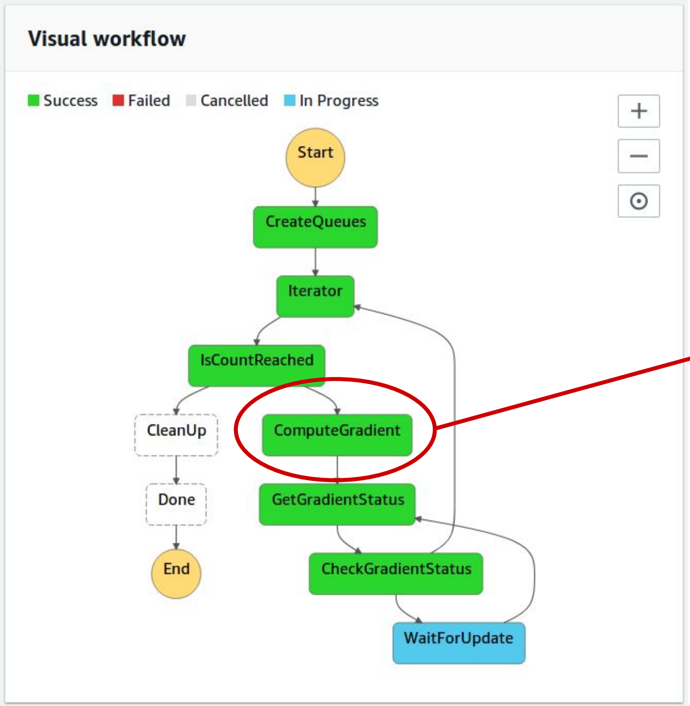
Legacy Fortran or C code

Lift and shift

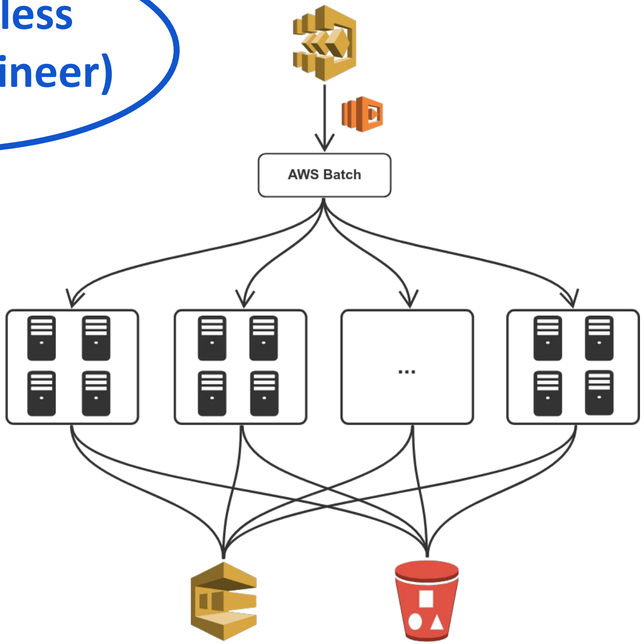


- Requires little to no work
- Long cluster start-up time and cost
- Idle instances/resilience/bandwidth/etc.
- Technically infeasible for industry scale

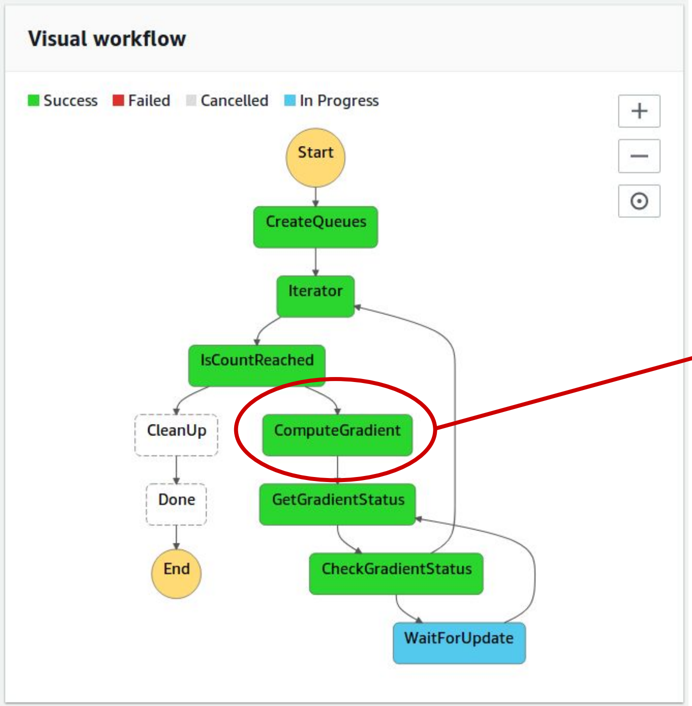
Moving to the cloud



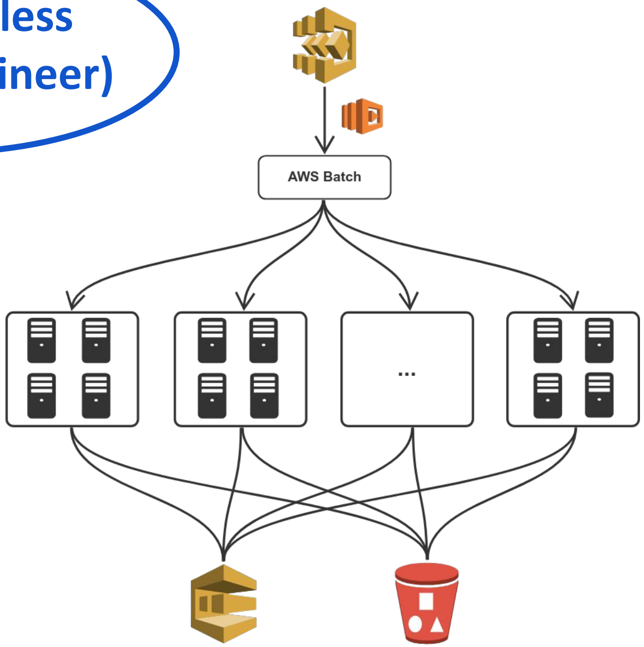
Go serverless
(and re-engineer)



Moving to the cloud



Go serverless
(and re-engineer)



- Save cost (up to **10x**): no idle instances, lower start-up time
- Resilience managed by cloud platform
- Requires re-engineering of software

* e.g. Valenciano, 2008; Dong et al., 2012; Zeng et al. 2014

SLIM

Serverless LS-RTM in the cloud

Typical components of LS-RTM*:

$$\underset{\delta \mathbf{m}}{\text{minimize}} \quad \sum_{i=1}^{n_s} \frac{1}{2} \left\| \mathbf{J}(\mathbf{m}, \mathbf{q}_i) \, \delta \mathbf{m} - \mathbf{d}_i^{\text{obs}} \right\|_2^2$$

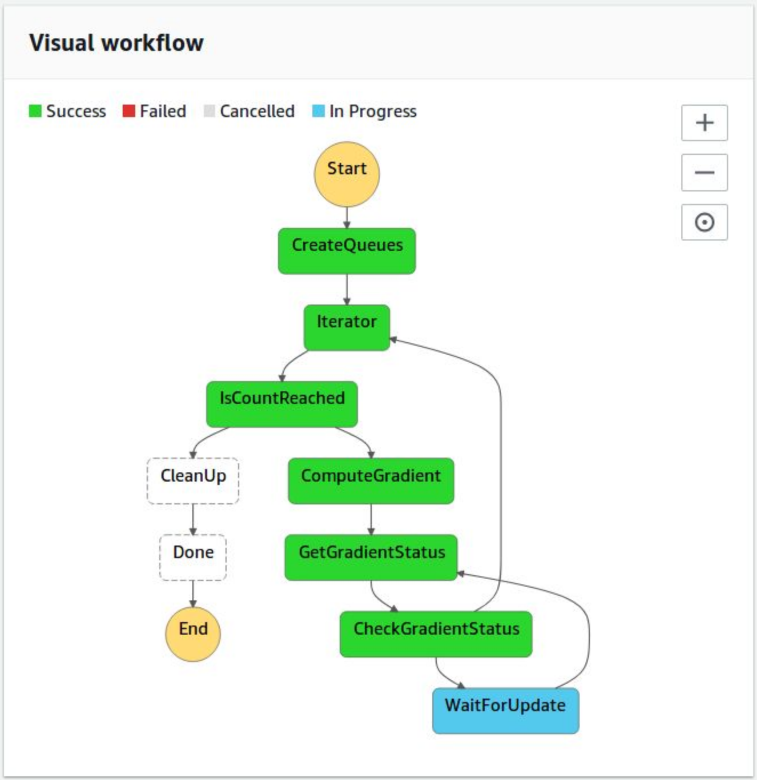
- 1. Compute gradient for all/subset of source locations: $\mathbf{g}_i = \mathbf{J}^\top \left(\mathbf{J} \, \delta \mathbf{m} - \mathbf{d}_i^{\text{obs}} \right)$
- 2. Sum gradients: $\mathbf{g} = \sum_{i=1}^{n_b} \mathbf{g}_i$
- 3. Update image based on optimization algorithm (SGD, CG, etc.):

$$\delta \mathbf{m} = \delta \mathbf{m} - \alpha \mathbf{g}$$

Serverless LS-RTM in the cloud

Serverless workflow with Step Functions:

- Algorithm as collection of *states**
- No compute instances required to execute workflow (i.e. *serverless*)
- States invoke AWS Lambda functions to run Python code
- Lambda functions: upload + run code w/o resource allocation



*Friedmann and Pizarro, AWS Compute Blog, 2017

Serverless LS-RTM in the cloud

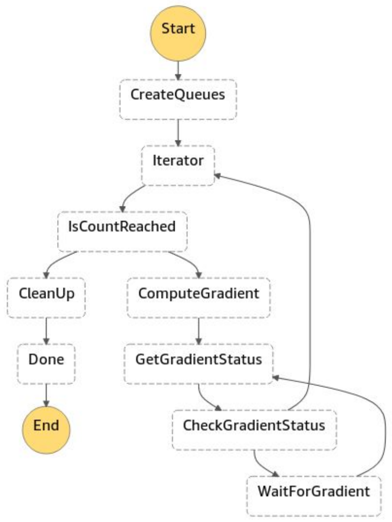
State machine defined as json file

Definition

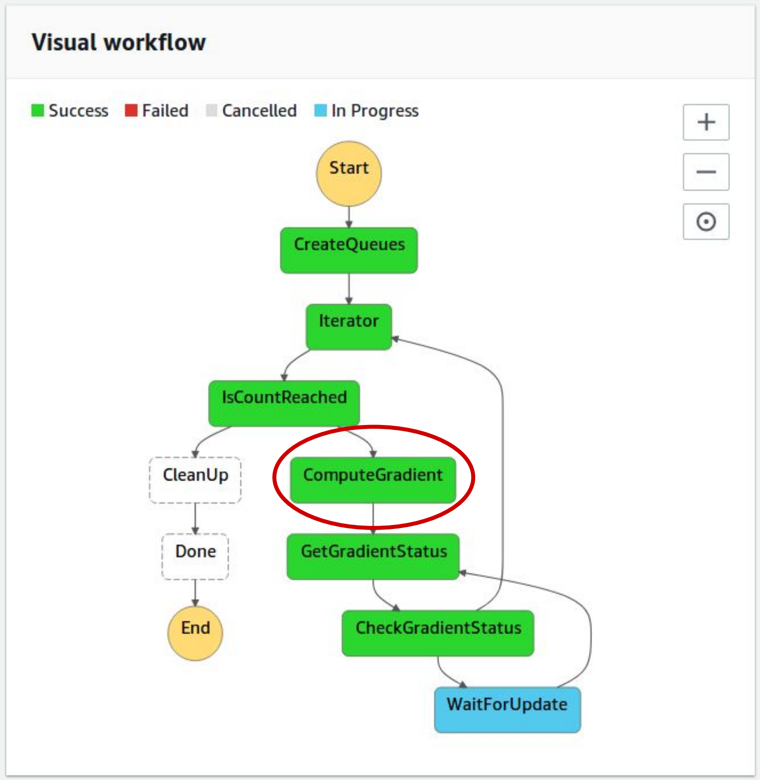
Generate code snippet

Learn more

```
{
    "Comment": "Iterator State Machine Example",
    "StartAt": "CreateQueues",
    "States": {
        "CreateQueues": {
            "Comment": "Create SQS queues and lambda triggers for the gradient reduction",
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-1:851065145468:function:CreateQueues",
            "ResultPath": "$",
            "Next": "Iterator"
        },
        "Iterator": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-1:851065145468:function:IteratorStochastic",
            "ResultPath": "$",
            "Next": "IsCountReached"
        },
        "IsCountReached": {
            "Type": "Choice",
            "Choices": [
                {
                    "Variable": "$.iterator.continue",
                    "BooleanEquals": true,
                    "Next": "ComputeGradient"
                }
            ],
            "Default": "CleanUp"
        },
        "ComputeGradient": {
            "Comment": "Your application logic, to run a specific number of times",
```



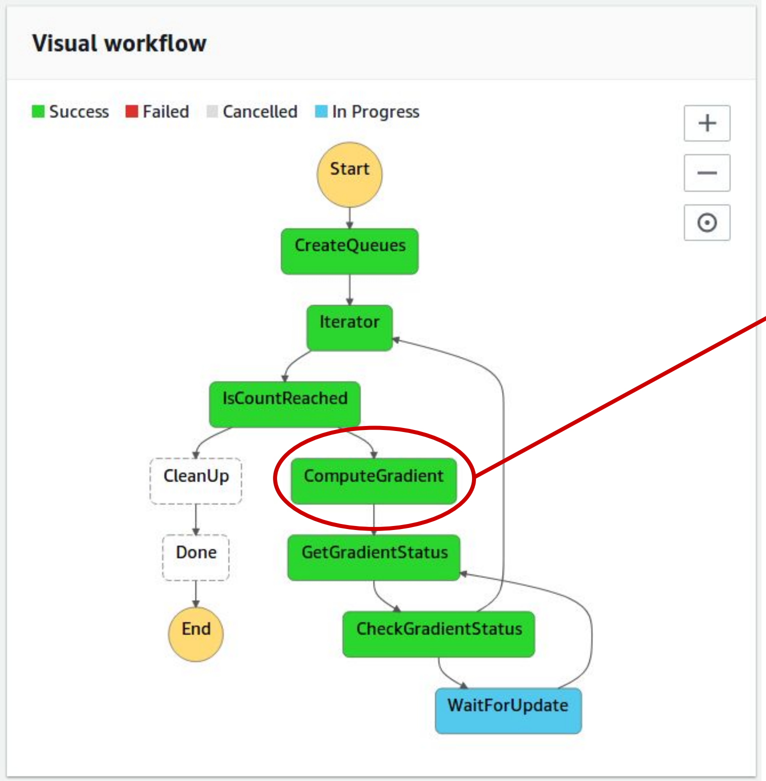
Gradient computations

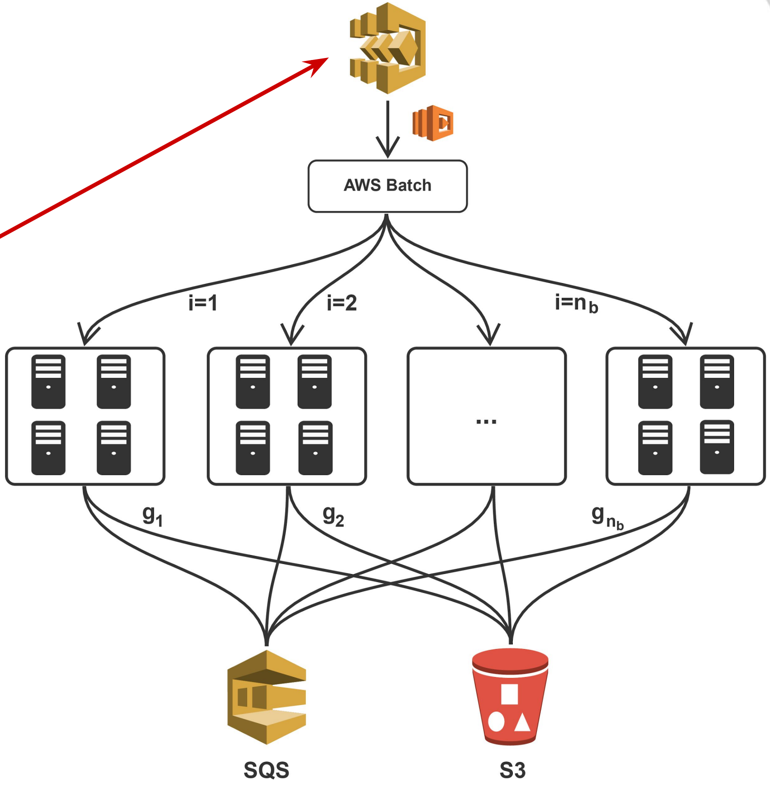


Compute gradients of the LS-RTM objective function:

- embarrassingly parallel
- model predicted data + backpropagate residual + imaging condition
- compute/memory heavy process (store/recompute wavefields)

Gradient computations

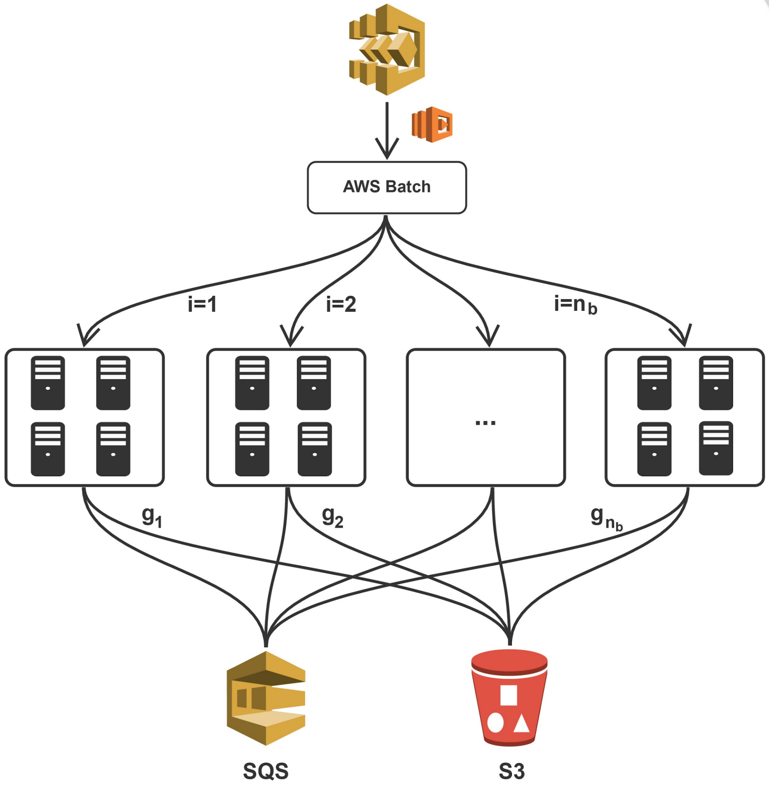




Gradient computations

Nested levels of parallelization:

- Parallelize shot records (AWS Batch)
- Domain decomposition (MPI)
- Multithreading (OpenMP)
- Each gradient computed on individual instance **or** cluster of instances (cluster of clusters)

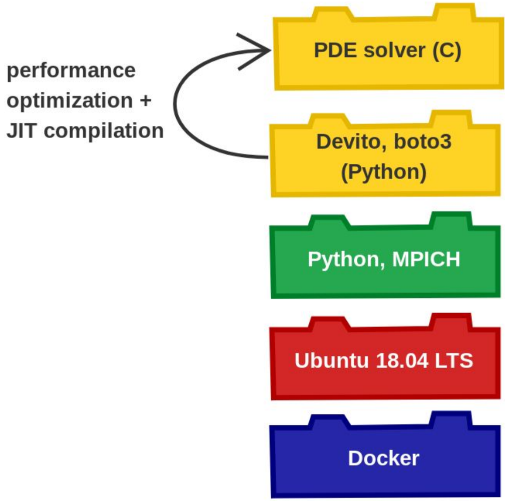


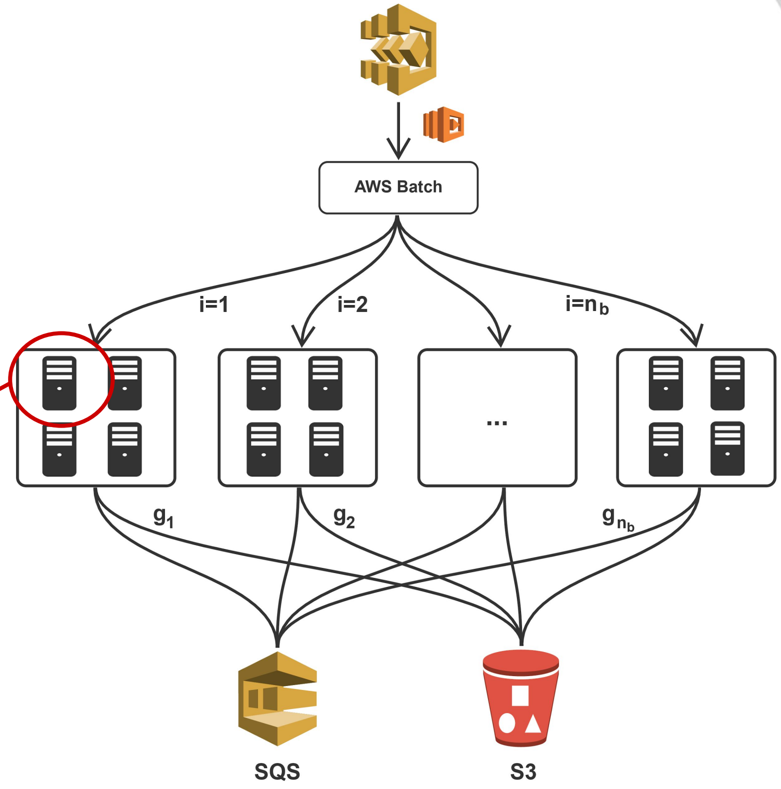
Gradient computations

* Luporini et al., 2018; Louboutin et al., 2019

Software to compute gradients:

- Batch runs docker containers
- Solve wave equations using Devito*
- Automated performance optimizations (loop blocking, vectorization, refactoring, OMP, MPI, etc.)

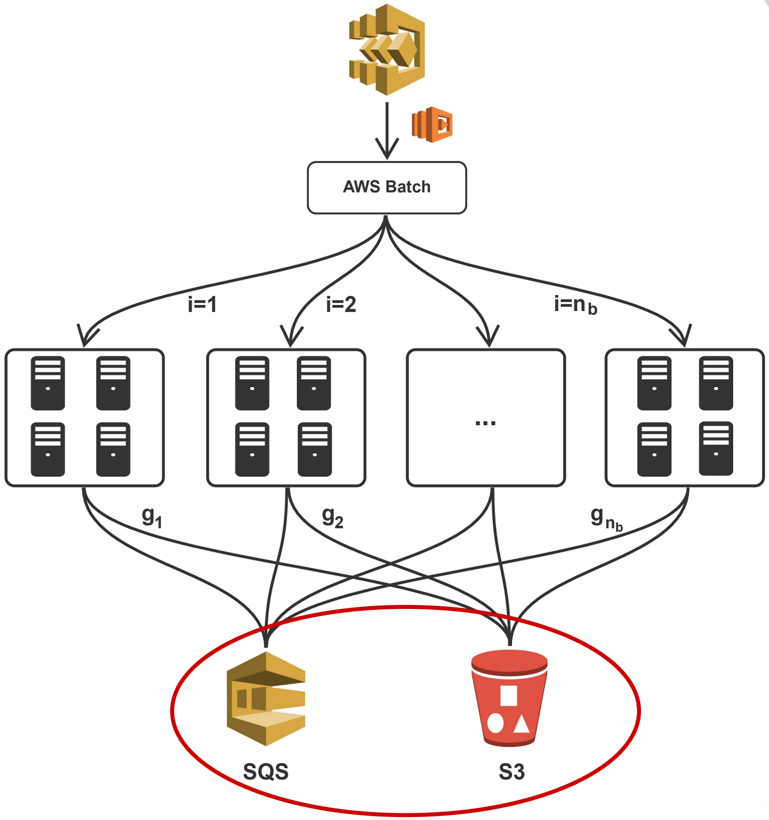




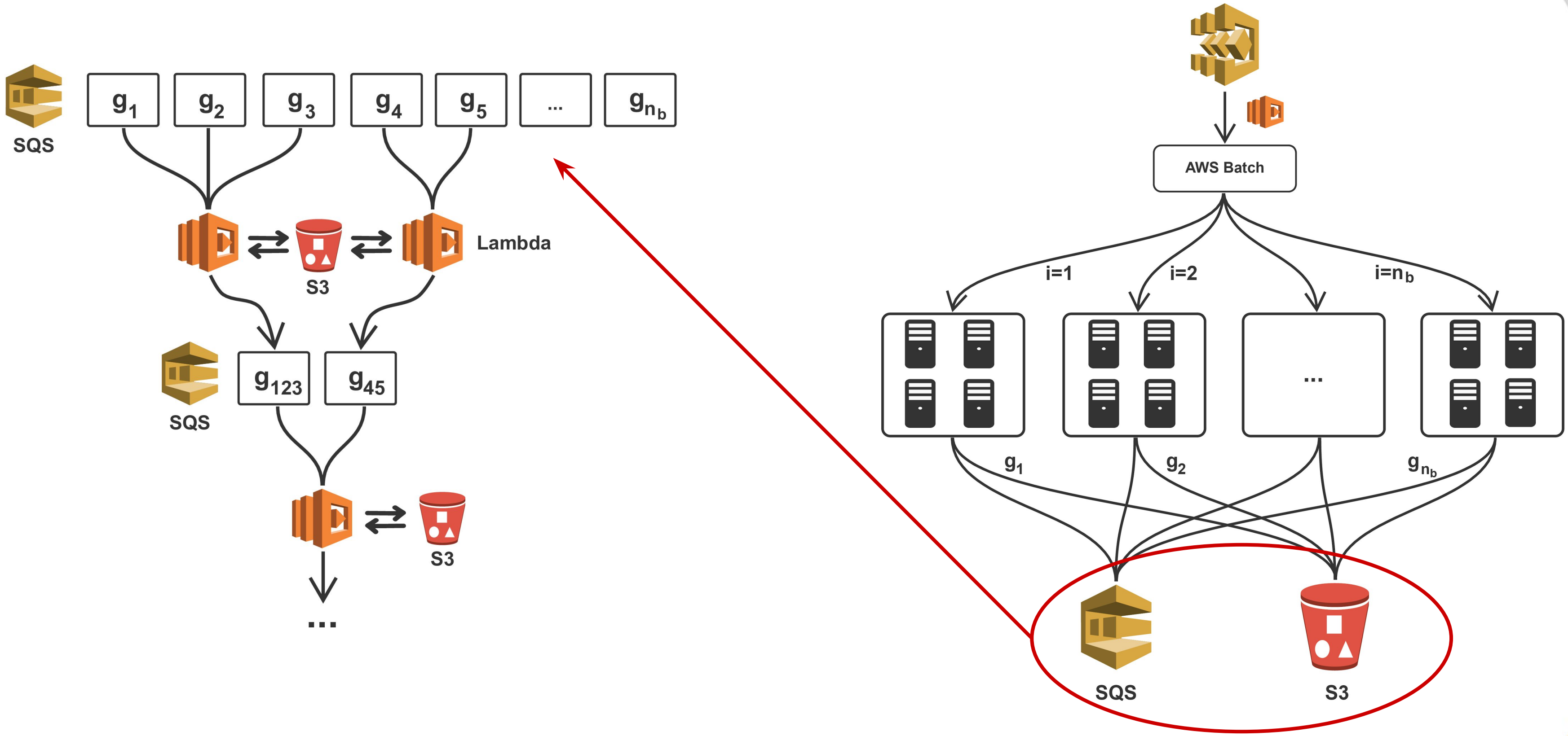
Gradient computations

Summation of gradients

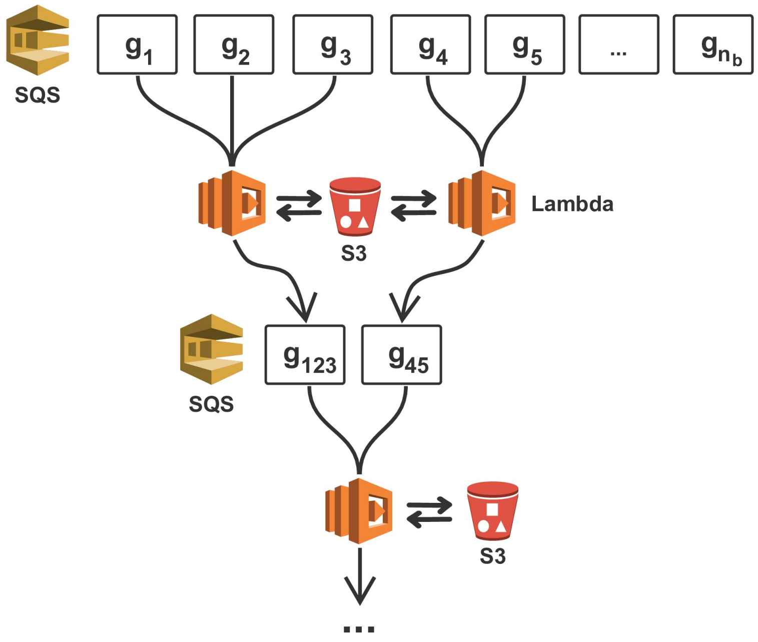
- Gradients stored in object storage (S3)
- Virtually unlimited I/O scalability
- Send object IDs to message queue
- Event-driven gradient summation using Lambda functions



Gradient computations



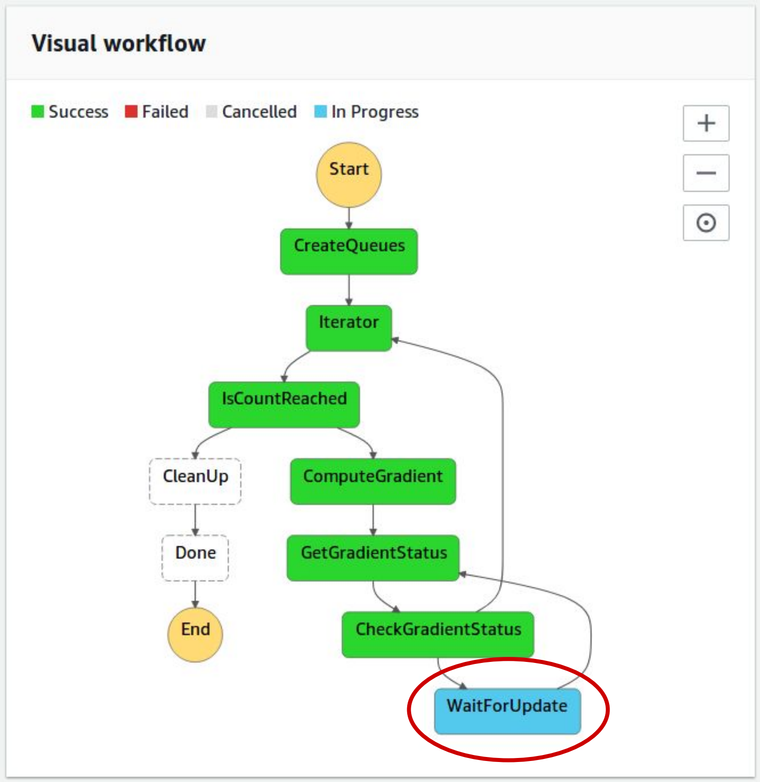
Gradient computations



Event-driven gradient reduction

- AWS Lambda functions
- Cheaper than compute nodes
- Asynchronous and parallel
- Invoked as soon as at least 2 gradients are available
- Stream gradients from S3 -> sum -> write back
- Update image after final summation

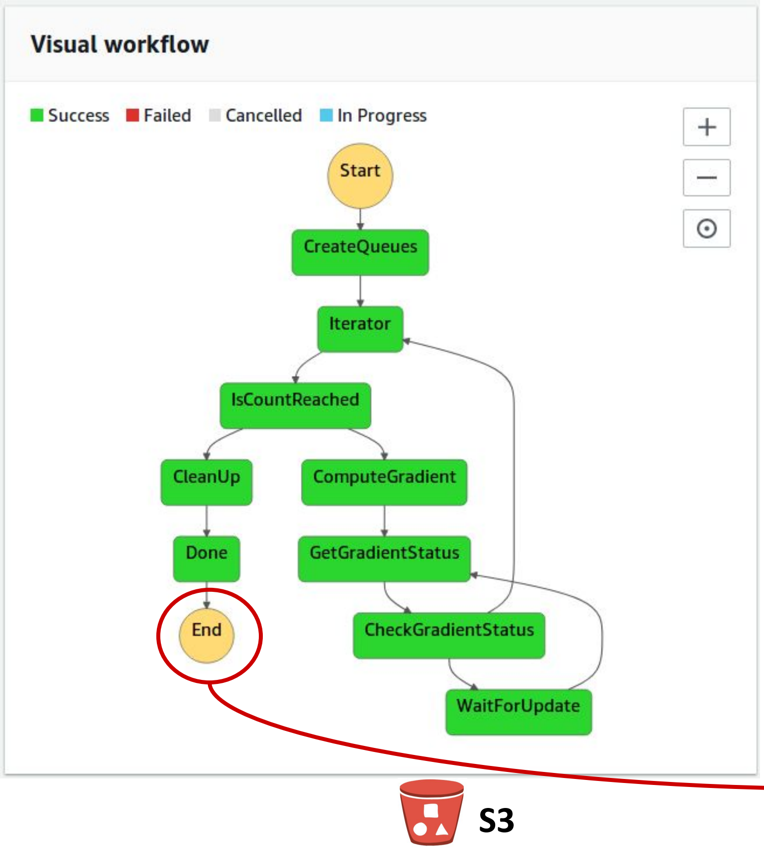
Gradient computations



Serverless workflow:

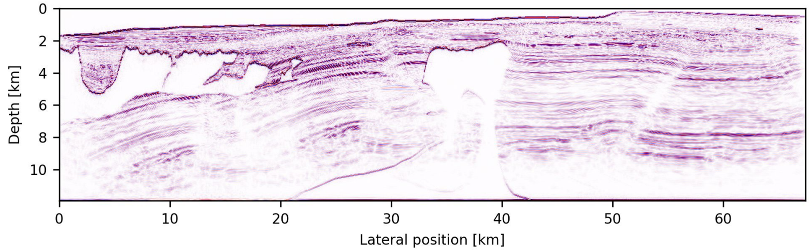
- No additional EC2 instances during gradient computation
- State machine waits for updated image
- Automatic progression to next iteration

Gradient computations



Serverless workflow:

- No additional EC2 instances during gradient computation
- State machine waits for updated image
- Automatic progression to next iteration
- Clean up resources after final iteration

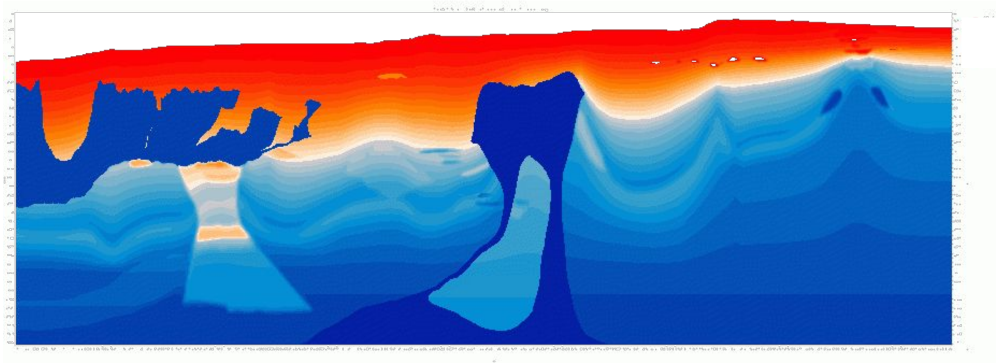


Billette and Brandsberg-Dahl, 2004

Numerical examples

Sparsity-promoting LS-RTM of the BP Synthetic 2004 model:

- 1348 shot records
- Velocity model: 67.4 x 11.9 km (10,789 x 1,911 grid points)
- 20 iterations of linearized Bregman method
- Batchsize of 200 shot records per iteration
- Curvelet-based sparsity promotion



BP Synthetic 2004

Numerical examples

Sparsity-promoting LS-RTM on the BP Synthetic 2004 model

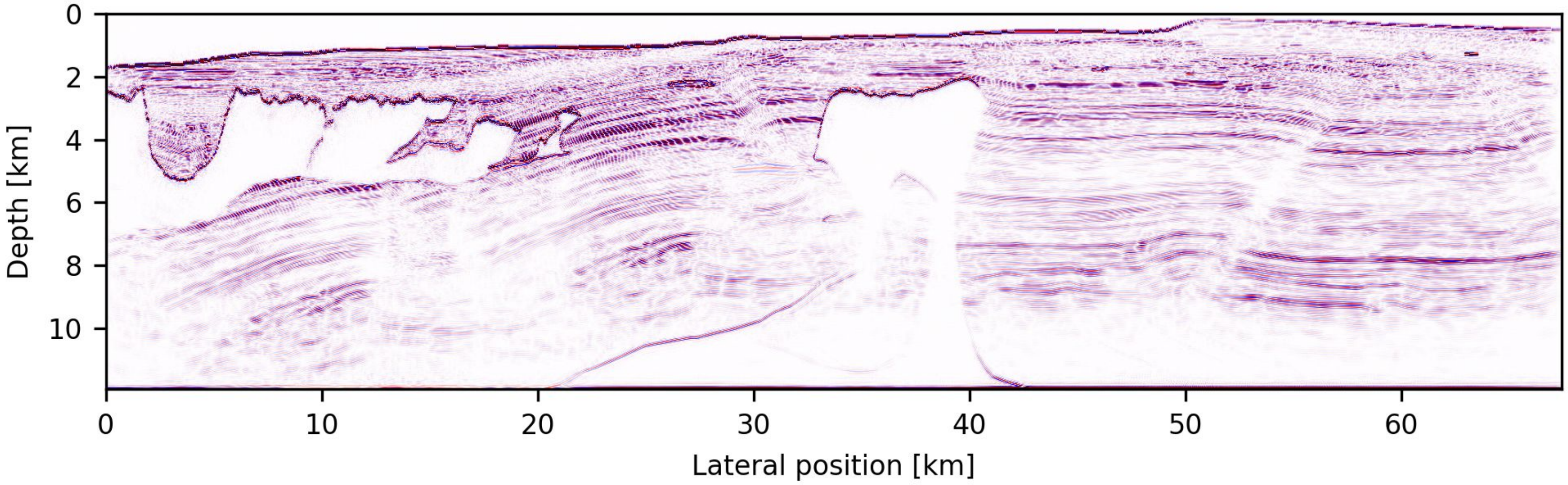
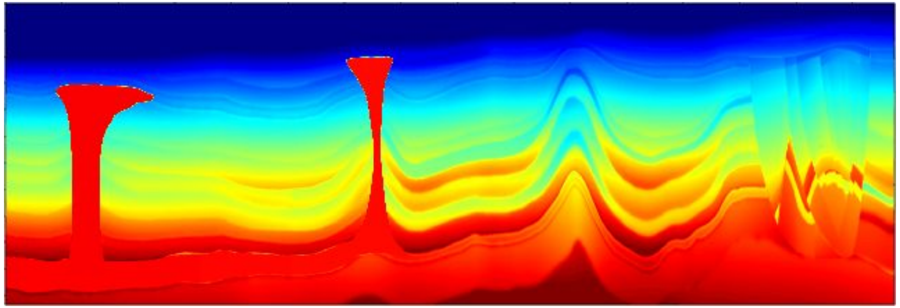


Image after ~3 data passes (total cost of < 120 $)

Numerical examples

Reverse-time migration of the BP TTI model:

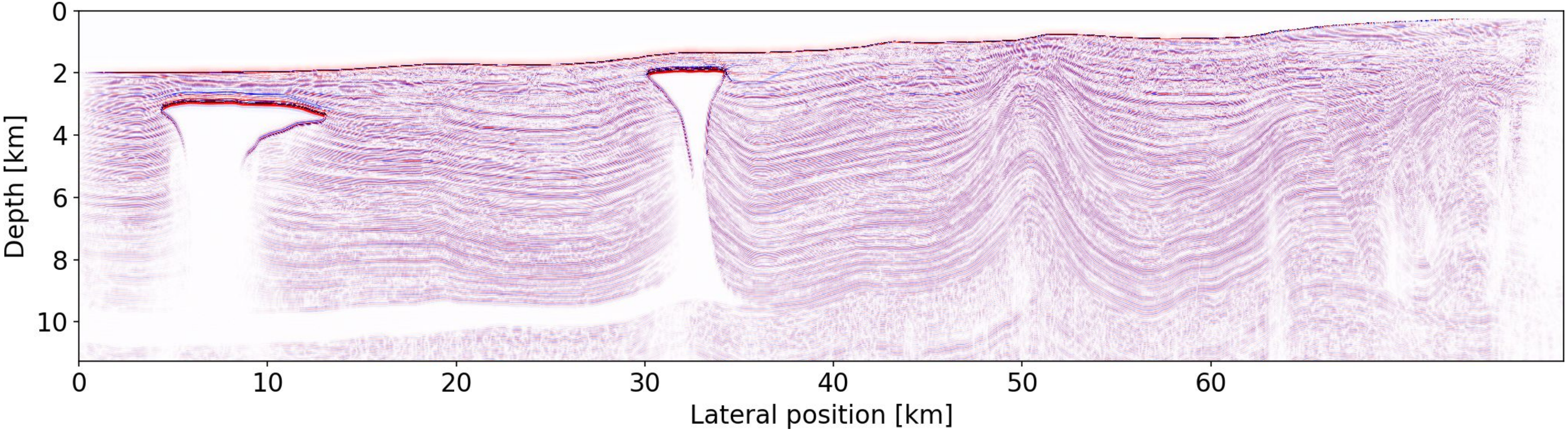
- 1641 shot records
- Velocity model: 78.7 x 11.3 km (12,596 x 1,801 grid points)
- Anisotropic modeling using pseudo-acoustic TTI equations*
- True adjoints of linearized Born scattering operator
- Domain-decomposition to compute gradients
- Each gradient computed on MPI cluster of 6 instances (no spot instances)



BP TTI 2007

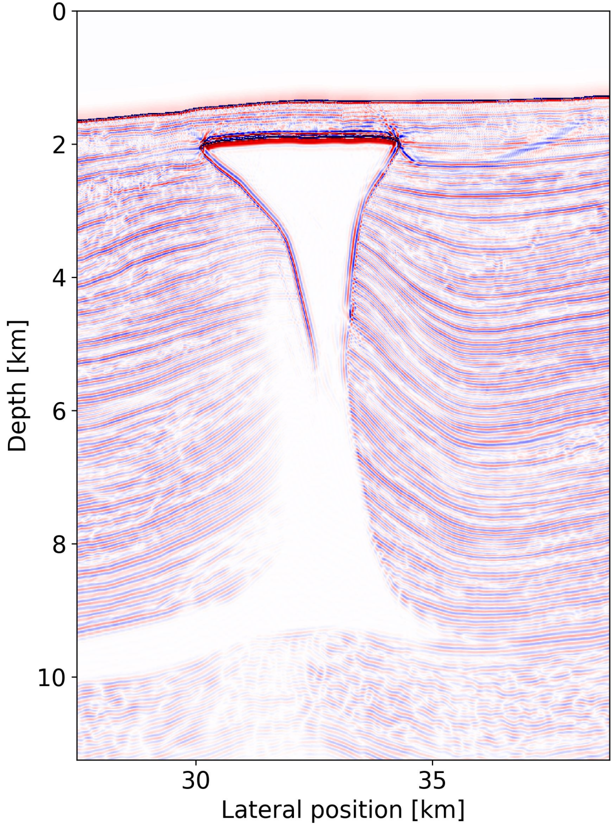
Numerical examples

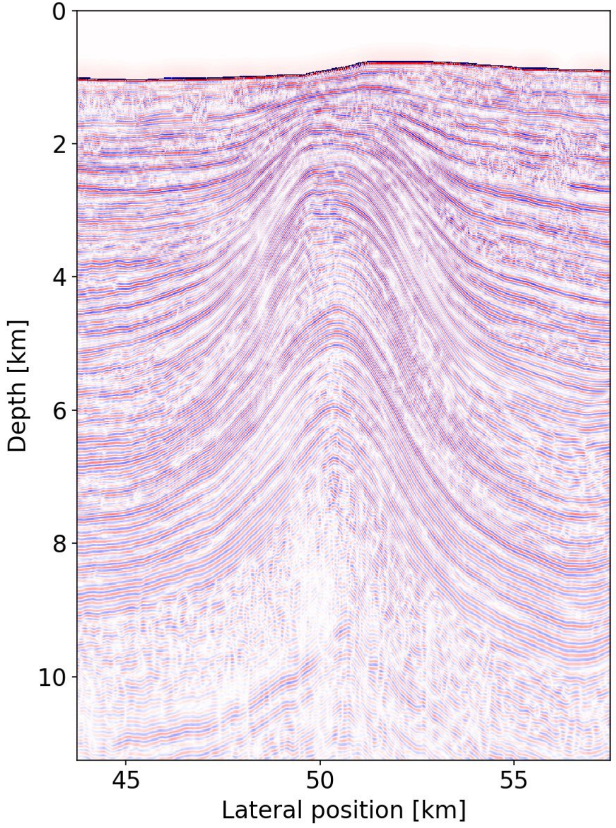
Reverse-time migration of the BP TTI model



RTM image (total cost of approx. 420 $)

Numerical examples





RTM image (total cost of approx. 420 $)

Numerical examples

	BP TTI 2007	BP 2004
No. of shots	1641	1348
Instances/gradient	6	1
Instance type	m5.xlarge	r5.large
Runtime/gradient	13.5 minutes	45 minutes
On-demand price/ gradient	0.26 $	0.0945 $
Spot price/gradient	N/A	0.027 $
On-demand price/ data pass	425.35 $	127.39 $
Spot price/data pass	N/A	35.99 $

Numerical examples

	BP TTI 2007	BP 2004
No. of shots	1641	1348
Instances/gradient	6	1
Instance type	m5.xlarge	r5.large
Runtime/gradient	13.5 minutes	45 minutes
On-demand price/ gradient	0.26 $	0.0945 $
Spot price/gradient	N/A	0.027 $
On-demand price/ data pass	425.35 $	127.39 $
Spot price/data pass	N/A	35.99 $

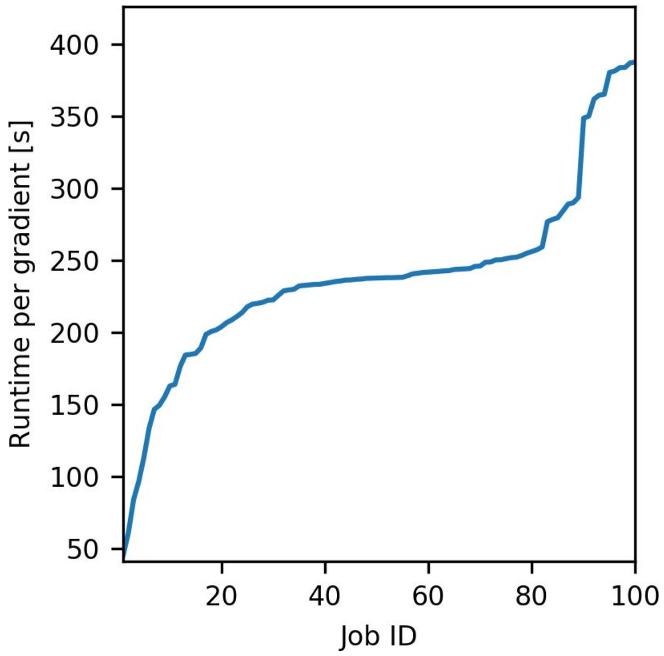
Serverless approach:

- Full flexibility
- On-demand + spot instances or combination of both
- Large number of instance types (memory, compute, GPU, HPC nodes)
- Adjust resources according to priority: cost, turn-around time, importance, etc.

Cost comparison

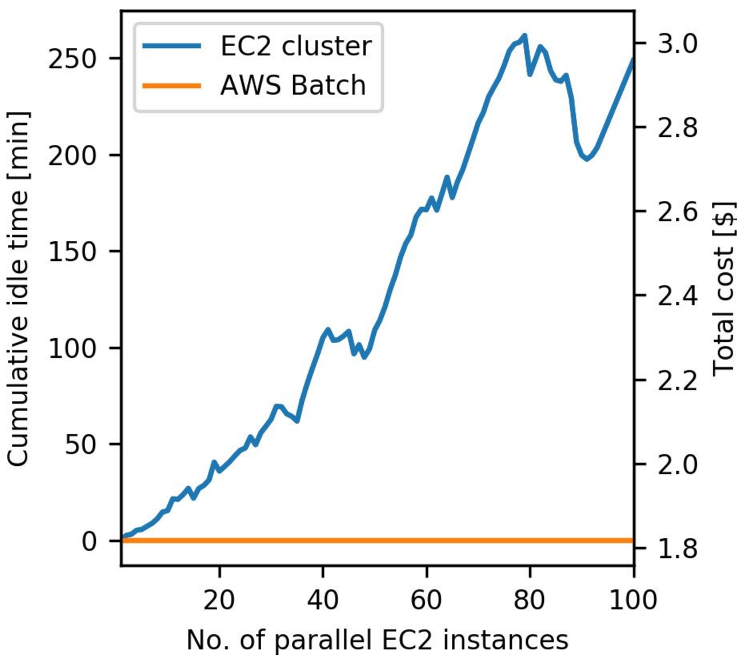
Compute 100 gradients for BP model:

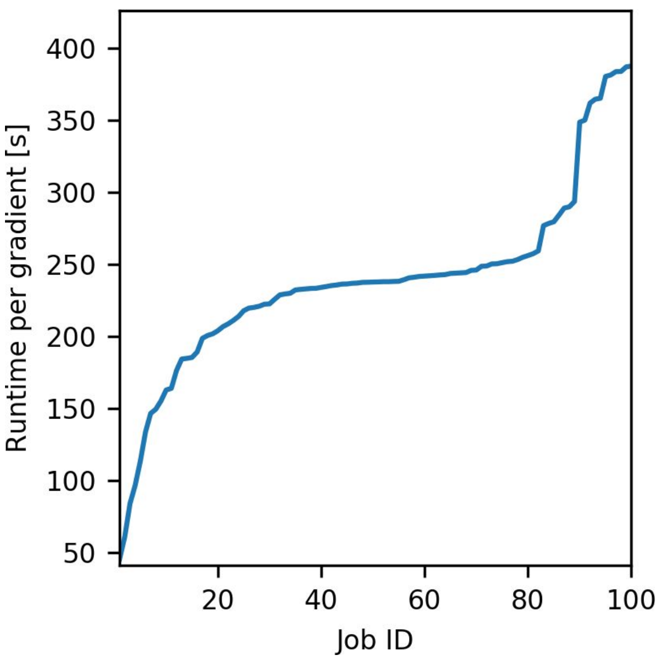
- Runtime varies for each gradient (EC2 related, varying max. offset, etc.)
- Fixed cluster: nodes have to wait until last gradient is computed
- Batch: each instance runs only as long computations last
- No cost during wait time for other gradients



Sorted runtimes of 100 gradients

Cost comparison

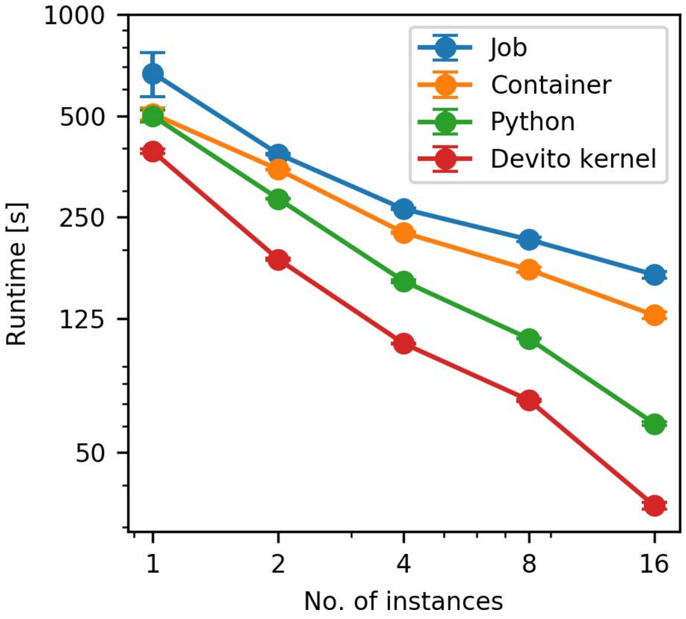


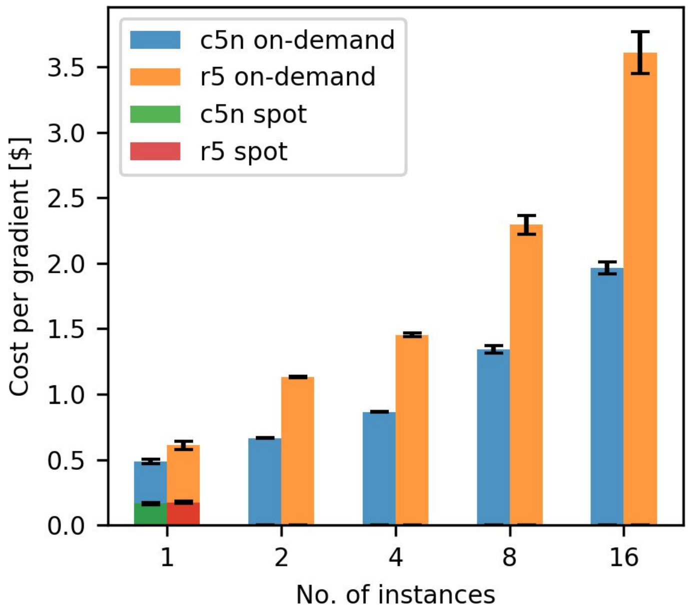


Sorted runtimes of 100 gradients

Strong scaling - MPI

- Fixed workload: 1 gradient
- Runtime as function of no. of instances (per gradient)
- Good speed-up **but** significant cost increase





Conclusions

Seismic imaging in the cloud:

- Need to rethink how to bring software to the cloud
- Lift and shift approach not ideal (complexity, resilience, cost)
- Instead: take advantage of new cloud technologies
- High-throughput batch computing, serverless/event-driven computations, object storage, spot instances
- Only pay what you use: up to 10x cost reduction
- **Software based on separation of concerns + abstractions is prerequisite to go serverless**

SLIM

Future directions

Go large:

- Collaboration with cloud providers to run at industry-scale
- 3D TTI RTM and LS-RTM
- SEAM model: long offset data acquisition w/ 3D elastic modeling
- Keynote speech at 4th **EAGE workshop on HPC for Upstream** (Dubai, Oct. 8, presented by F. J. Herrmann)

Check for updates on our website and on Researchgate:

https://slim.gatech.edu/

https://www.researchgate.net/lab/SLIM-Felix-J-Herrmann

Acknowledgments

This research was funded by the Georgia Research Alliance and the Georgia Institute of Technology.

Arxiv preprint: *https://arxiv.org/abs/1909.01279*