# MsPASS: A Parallel Processing Framework for Seismology

Yinzhi Wang[1] and Gary Pavlis[2]

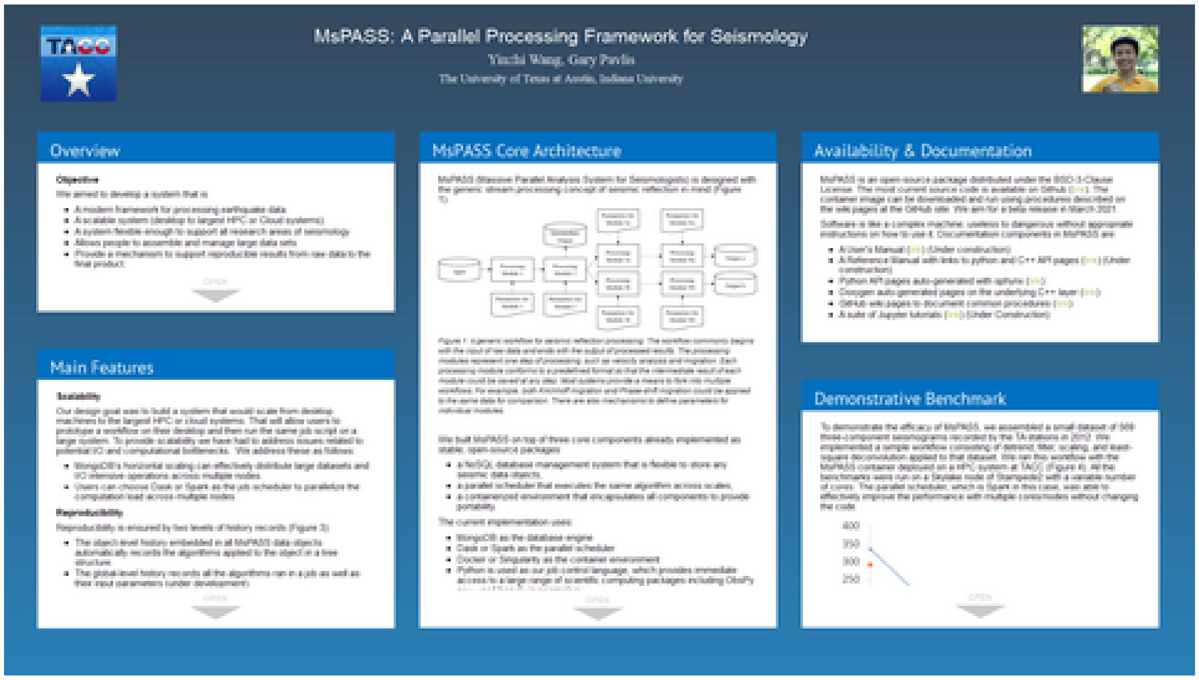
[1]University of Texas at Austin
[2]Indiana University Bloomington

November 24, 2022

## Abstract

Over the past decade, the huge success in many large-scale projects like the USArray component of Earthscope gave rise to a massive increase in the data volume available to the seismology community. We assert that the software infrastructure of the field has not kept up with parallel developments in 'big data' sciences. As a step towards enabling research at the extreme scale to more of the seismology community, we are developing a new framework for seismic data processing and management we call Massive Parallel Analysis System for Seismologists (MsPASS). MsPASS leverages several existing technologies: (1) Spark as the scalable parallel processing framework, (2) MongoDB as the flexible database system, and (3) Docker and Singularity as the containerized virtual environment. The core of the system builds on a rewrite of the SEISPP package to implement wrappers around the widely accepted ObsPy toolkit. The wrappers automate many database operations and provide a mechanism to automatically save the processing history and provide a mechanism for reproducibility. The synthesis of these components can provide flexibility to adapt to a wide range of data processing workflows. The use of containers enables the deployment to a wide range of computing platforms without requiring intervention by system administrators. We evaluate the effectiveness of the system with a deconvolution processing workflow applied to USArray data. Through extensive documentation and examples, we aim to make this system a sustainable, open-source framework for the community.

# MsPASS: A Parallel Processing Framework for Seismology



MsPASS: A Parallel Processing Framework for Seismology

Yinzhi Wang, Gary Pavlis

The University of Texas at Austin, Indiana University

## Overview

**Objective**

We aimed to develop a system that is:

- A modern framework for processing earthquake data
- A scalable system (desktop to largest HPC or Cloud systems)
- A system flexible enough to support all research areas of seismology
- Allows people to assemble and manage large data sets
- Provide a mechanism to support reproducible results from raw data to the final product

## Main Features

**Scalability**

Our design goal was to build a system that would scale from desktop machines to the largest HPC or cloud systems. That will allow users to prototype a workflow on their desktop and then run the same job script on a large system. To provide scalability we have had to address issues related to potential IO and computational bottlenecks. We address these as follows:

- MongoDB's horizontal scaling can effectively distribute large datasets and I/O intensive operations across multiple nodes.
- Users can choose Dask or Spark as the job scheduler to parallelize the computation load across multiple nodes.

**Reproducibility**

Reproducibility is ensured by two levels of history records (Figure 3):

- The object-level history embedded in all MsPASS data objects automatically records the algorithms applied to the object in a tree structure.
- The global-level history records all the algorithms ran in a job as well as their input parameters (under development).

## MsPASS Core Architecture

MsPASS (Massive Parallel Analysis System for Seismologists) is designed with the generic stream processing concept of seismic reflection in mind (Figure 1).

Figure 1. A generic workflow for seismic reflection processing. The workflow commonly begins with the input of raw data and ends with the output of processed results. The processing modules represent one step of processing, such as velocity analysis and migration. Each processing module conforms to a predefined format so that the intermediate result of each module could be saved at any step. Most systems provide a means to fork into multiple workflows. For example, both Kirchhoff migration and Phase-shift migration could be applied to the same data for comparison. There are also mechanisms to define parameters for individual modules.

We built MsPASS on top of three core components already implemented as stable, open-source packages:

- a NoSQL database management system that is flexible to store any seismic data objects,
- a parallel scheduler that executes the same algorithm across scales,
- a containerized environment that encapsulates all components to provide portability.

The current implementation uses:

- MongoDB as the database engine
- Dask or Spark as the parallel scheduler
- Docker or Singularity as the container environment
- Python is used as our job control language, which provides immediate access to a large range of scientific computing packages including ObsPy

## Availability & Documentation

MsPASS is an open-source package distributed under the BSD-3-Clause License. The most current source code is available on GitHub (link). The container image can be downloaded and run using procedures described on the wiki pages at the GitHub site. We aim for a beta release in March 2021.

Software is like a complex machine; useless to dangerous without appropriate instructions on how to use it. Documentation components in MsPASS are:

- A User's Manual (link) (Under construction)
- A Reference Manual with links to python and C++ API pages (link) (Under construction)
- Python API pages auto-generated with sphinx (link)
- Doxygen auto-generated pages on the underlying C++ layer (link)
- GitHub wiki pages to document common procedures (link)
- A suite of Jupyter tutorials (link) (Under Construction)

## Demonstrative Benchmark

To demonstrate the efficacy of MsPASS, we assembled a small dataset of 509 three-component seismograms recorded by the TA stations in 2012. We implemented a simple workflow consisting of detrend, filter, scaling, and least-square deconvolution applied to that dataset. We ran this workflow with the MsPASS container deployed on a HPC system at TACC (Figure 4). All the benchmarks were run on a Skylake node of Stampede2 with a variable number of cores. The parallel scheduler, which is Spark in this case, was able to effectively improve the performance with multiple cores/nodes without changing the code.

Yinzhi Wang, Gary Pavlis

The University of Texas at Austin, Indiana University

PRESENTED AT:

AGU FALL MEETING

Online Everywhere | 1–17 December 2020

## OVERVIEW

**Objective**

We aimed to develop a system that is

- A modern framework for processing earthquake data.
- A scalable system (desktop to largest HPC or Cloud systems).
- A system flexible enough to support all research areas of seismology
- Allows people to assemble and manage large data sets.
- Provide a mechanism to support reproducible results from raw data to the final product.

**Motivation**

- Main software packages in use are based on archaic technology
- Analysis of large data sets like USArray has been compromised by inadequate software infrastructure
- Innovation is limited by too much time spent on tedious tasks that could be solved with better software infrastructure.

**Data Management Issues in Seismology**

- Large volumes of data involved in any modern seismological research give rise to the need for effective local data management solutions.
- Existing relational database management systems for seismology lack flexibility (and performance) when adopted in a research data processing environment.
- Seismological research data today are commonly organized and managed as a hierarchy of files and directories indexed by their names. This creates scaling problems on HPC's I/O systems.

**Data Processing Issues in Seismology**

- The core processing concepts of seismology comes from seismic reflection processing, where data objects limited by a pre-defined format flow through a chain of algorithms.
- SAC is the most common tool for passive seismic data processing influenced by that concept. ObsPy and some similar toolkits in Matlab are modern alternatives to SAC that are less restrictive.
- Emerging HPC and Cloud resources have led to a range of advanced software packages that can exploit the parallel architecture of those systems. All, however, are specialized and not designed as a general processing framework

## MAIN FEATURES

**Scalability**

Our design goal was to build a system that would scale from desktop machines to the largest HPC or cloud systems. That will allow users to prototype a workflow on their desktop and then run the same job script on a large system. To provide scalability we have had to address issues related to potential I/O and computational bottlenecks. We address these as follows:

- MongoDB's horizontal scaling can effectively distribute large datasets and I/O intensive operations across multiple nodes.
- Users can choose Dask or Spark as the job scheduler to parallelize the computation load across multiple nodes.

**Reproducibility**

Reproducibility is ensured by two levels of history records (Figure 3):

- The object-level history embedded in all MsPASS data objects automatically records the algorithms applied to the object in a tree structure.
- The global-level history records all the algorithms ran in a job as well as their input parameters (under development).
- The object-level history is linked to the global-level history with unique algorithm IDs.
- Any processed waveform can be reproduced from the raw data with the history records.
- Error logs from any algorithms are also preserved with the data object as an auxiliary provenance information.

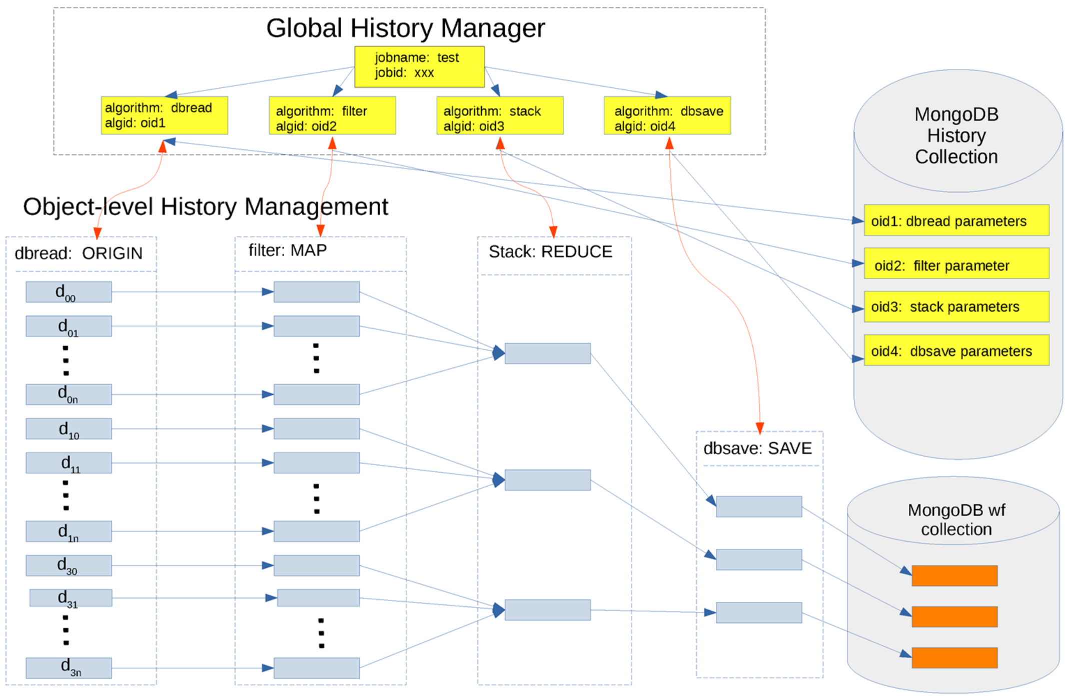


*Figure 3. Wiring diagram illustrating concepts used to build our processing history mechanism. The overall workflow is defined by distinct processing modules illustrated at the top of the figure with yellow boxes. Each algorithm is assigned a unique name and an id defined by the MongoDB ObjectID of a document defining input parameters for a unique instance of that algorithm. Parameters can be as simple as a set of command line arguments or as elaborate as a tree defined by xml, yaml, or an Antelope parameter file format. The example here shows four processing steps defined by a reader (dbread), a map operator (filter), a reduce operator (stack), and a save operator (dbsave). The topmost box defines an instance of a workflow run defined by the unique combination of a jobname and jobid. The lower part of the figure shows a second, novel element of our framework we call an object-level history. The definitions of the history (i.e. the algorithms being run) are coordinated by the global history manager (top of the figure). Each processing module imprints it's name and instance (algid) as data passes through the system. That communication is illustrated by red arrows between the top and bottom parts of the figure. The example processing flow shown has the four steps defined in the yellow boxes, but illustrated here as a G-tree used as the conceptual way we preserve the object-level history. The example shows a set*

*of data objects that define the origin. The box on the left is the distributed inputs defined by reading data by some mechanism (normally under management by MongoDB). The second stage is a map operator defined by a standard time-domain filter operator. The third stage in this illustration is a reduce operator, stack (average), that in this example takes input from 3 groups of seismograms and outputs 3 stacks. The last stage is a save.*

**Portability**

- The containers make it easy to deploy MsPASS on machines from a laptop to the largest HPC and Cloud computing systems.
- The containers isolate the system environment to allow distribution of the system as a complete package that requires no auxiliary package installations.
- Both the MongoDB and the Dask/Spark components within the container can be configured to utilize the multiple nodes on HPC or Cloud.
- The same script of a workflow can run with any configuration without change.

**Flexibility for Research Prototypes**

- Python as the job control language is familiar to many.
- Python is more flexible than traditional seismology dependence on unix shell or specialized interpreters (e.g. SAC).
- The Metadata object used in MsPASS can be thought of as an infinitely extensible header – adaptable to any problem in contrast to rigid formats like SEGY or SAC.
- MongoDB's flexible namespace permits extensions to new problems without starting from scratch.
    - Auxiliary "collections" are used at present for importing variable formats.
    - Simple procedure to register new attributes to be formally managed by MongoDB or left unregistered to prototyping.
    - Flexible alias mechanism to map different namespaces – e.g. SAC aliases.

## MSPASS CORE ARCHITECTURE

MsPASS (Massive Parallel Analysis System for Seismologists) is designed with the generic stream processing concept of seismic reflection in mind (Figure 1).

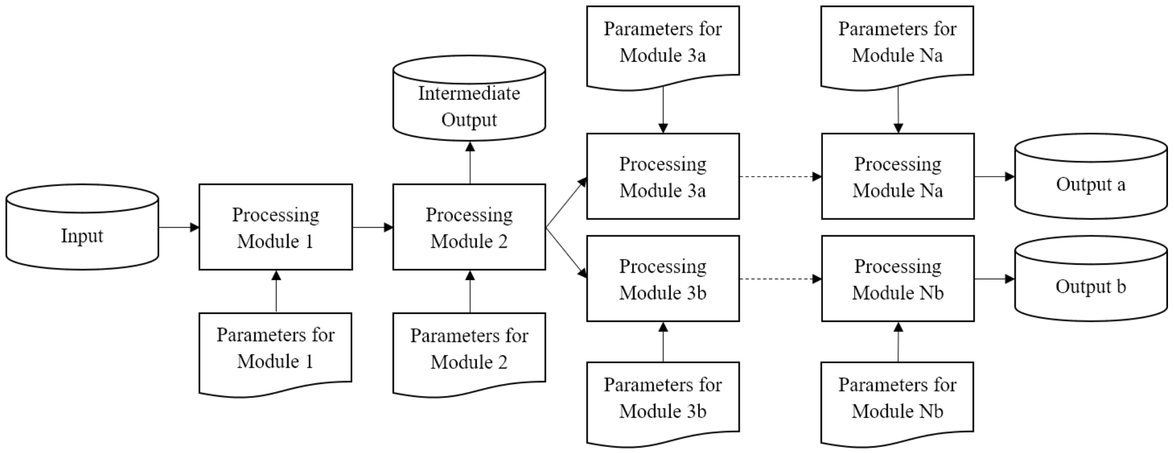


*Figure 1. A generic workflow for seismic reflection processing. The workflow commonly begins with the input of raw data and ends with the output of processed results. The processing modules represent one step of processing, such as velocity analysis and migration. Each processing module conforms to a predefined format so that the intermediate result of each module could be saved at any step. Most systems provide a means to fork into multiple workflows. For example, both Kirchhoff migration and Phase-shift migration could be applied to the same data for comparison. There are also mechanisms to define parameters for individual modules.*

We built MsPASS on top of three core components already implemented as stable, open-source packages:

- a NoSQL database management system that is flexible to store any seismic data objects,
- a parallel scheduler that executes the same algorithm across scales,
- a containerized environment that encapsulates all components to provide portability.

The current implementation uses:

- MongoDB as the database engine
- Dask or Spark as the parallel scheduler
- Docker or Singularity as the container environment
- Python is used as our job control language, which provides immediate access to a large range of scientific computing packages including ObsPy now used heavily in seismology.

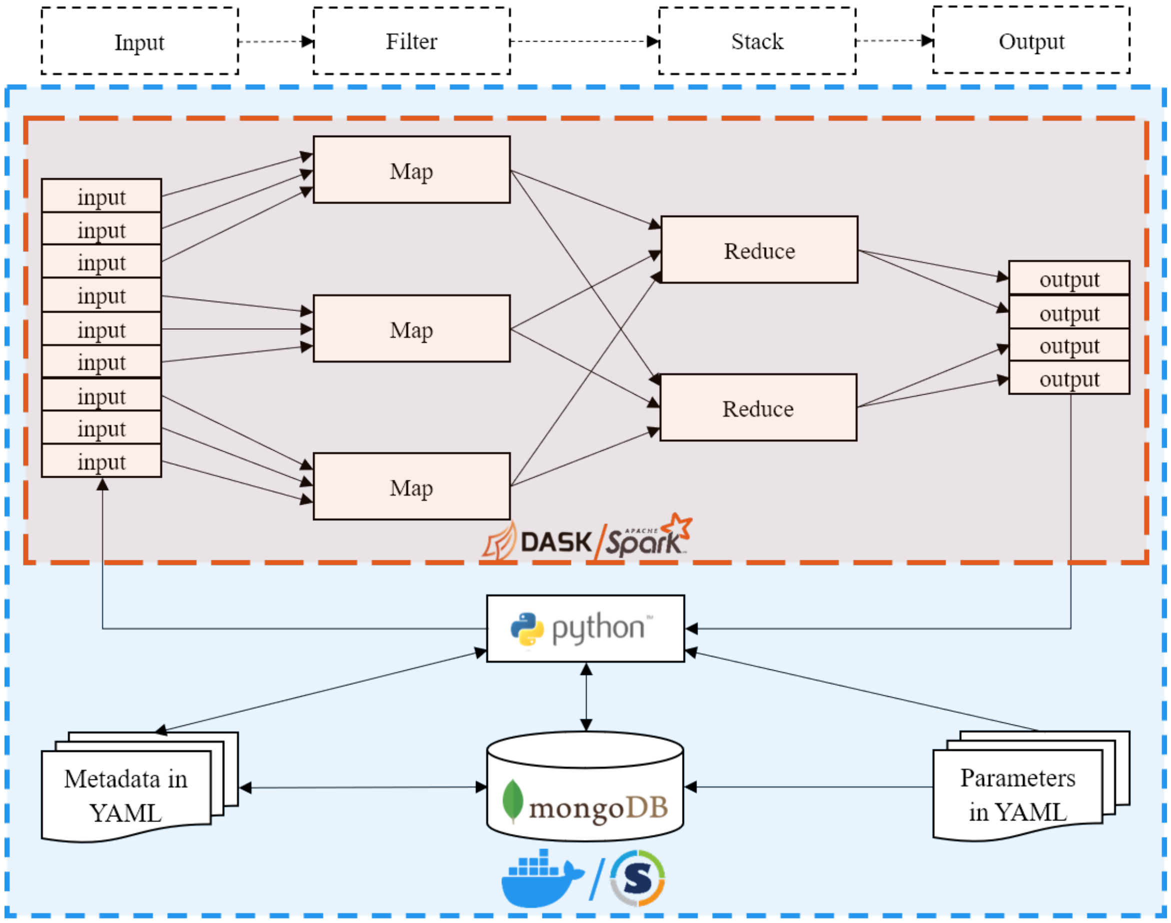


*Figure 2. An example processing workflow to filter and stack an ensemble of waveforms. The top is the user view of data flow, and the bottom is the actual execution graph. A Python script drives the workflow by selecting metadata from MongoDB, linking them with waveforms, and setting up a distributed input dataset. Each map operation applies the filter to its local data and passes on the outputs for the next step. The reducer operations then apply stacking and produce outputs to be stored in the database. The map and reduce operations are orchestrated by Dask or Spark in parallel. All the components are containerized in an image that is compatible with Docker and Singularity.*

## AVAILABILITY & DOCUMENTATION

MsPASS is an open-source package distributed under the BSD-3-Clause License. The most current source code is available on Github (link (https://github.com/wangyinz/mspass)). The container image can be downloaded and run using procedures described on the wiki pages at the GitHub site. We aim for a beta release in March 2021.

Software is like a complex machine; useless to dangerous without appropriate instructions on how to use it. Documentation components in MsPASS are:

- A User's Manual (link (https://wangyinz.github.io/mspass/index.html)) (Under construction)
- A Reference Manual with links to python and C++ API pages (link (https://wangyinz.github.io/mspass/reference_manual.html)) (Under construction)
- Python API pages auto-generated with sphynx (link (https://wangyinz.github.io/mspass/py-modindex.html))
- Doxygen auto-generated pages on the underlying C++ layer (link (https://wangyinz.github.io/mspass/cxx_api/mspass.html#mspass-namespace))
- GitHub wiki pages to document common procedures (link (https://github.com/wangyinz/mspass/wiki))
- A suite of Jupyter tutorials (link (https://github.com/wangyinz/mspass_tutorial/tree/master/notebooks)) (Under Construction)

## DEMONSTRATIVE BENCHMARK

To demonstrate the efficacy of MsPASS, we assembled a small dataset of 569 three-component seismograms recorded by the TA stations in 2012. We implemented a simple workflow consisting of detrend, filter, scaling, and least-square deconvolution applied to that dataset. We ran this workflow with the MsPASS container deployed on a HPC system at TACC (Figure 4). All the benchmarks were run on a Skylake node of Stampede2 with a variable number of cores. The parallel scheduler, which is Spark in this case, was able to effectively improve the performance with multiple cores/nodes without changing the code.

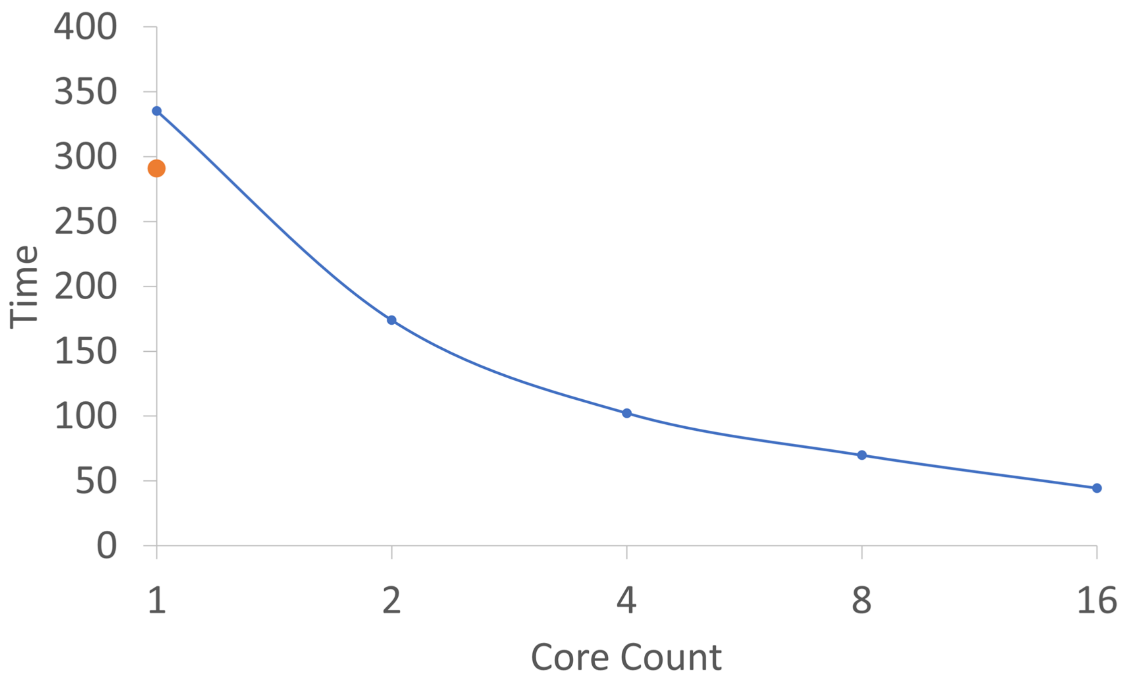


*Figure 4. Scaling of the demonstrative workflow of applying detrend, filter, scaling, and least-square deconvolution to 569 three-component seismograms. The blue line shows the wall time of the workflow executed with different numbers of cores within Spark. The red dot is the wall time of the same workflow running as a sequential Python script. The difference in single core performance is from the overhead of the Spark scheduler.*

We also ran the workflow on multiple nodes but we found this example did not scale further due to the size of the test dataset. A larger dataset is needed to take full advantage of Spark along with the parallel reader with MongoDB that is currently under construction. The parallel reader is expected to be functional by early 2021.

## ABSTRACT

Over the past decade, the huge success in many large-scale projects like the USArray component of Earthscope gave rise to a massive increase in the data volume available to the seismology community. We assert that the software infrastructure of the field has not kept up with parallel developments in ‘big data’ sciences. As a step towards enabling research at the extreme scale to more of the seismology community, we are developing a new framework for seismic data processing and management we call Massive Parallel Analysis System for Seismologists (MsPASS). MsPASS leverages several existing technologies: (1) Spark as the scalable parallel processing framework, (2) MongoDB as the flexible database system, and (3) Docker and Singularity as the containerized virtual environment. The core of the system builds on a rewrite of the SEISPP package to implement wrappers around the widely accepted ObsPy toolkit. The wrappers automate many database operations and provide a mechanism to automatically save the processing history and provide a mechanism for reproducibility. The synthesis of these components can provide flexibility to adapt to a wide range of data processing workflows. The use of containers enables the deployment to a wide range of computing platforms without requiring intervention by system administrators. We evaluate the effectiveness of the system with a deconvolution processing workflow applied to USArray data. Through extensive documentation and examples, we aim to make this system a sustainable, open-source framework for the community.