Refactoring Reverse Time Migration with Apache Spark: 1. Towards In-Memory Coherence of Seismic-Reflection Wavefields

L. Ian Lumb
Division of Natural
Science
Faculty of Science
York University
Toronto, Ontario M3J 1P3, Canada
Email: ianlumb@yorku.ca

***Abstract*—Reverse-Time Migration (RTM) is widely acknowledged as the most-effective methodology for 'sharpening' the images generated by reflection seismology in support of petroleum exploration. Conventional RTM, however, is significantly impacted by two performance-related challenges. Although attention on RTM's compute-intensive modeling kernel is deferred to a subsequent effort, the need for significant disk I/O is addressed here. Broadly speaking, in-memory parallelism is introduced here through use of Apache Spark via the Thunder Project. In particular, a toy problem demonstrates that disk I/O can be effectively decoupled from computation during application of RTM's imaging condition - a condition that can now be applied as an in-memory cross-correlation between RTM's forward and reverse wavefields. Isolated prototyping on a laptop is shown to scale out to clusters, where Spark workloads can be managed alongside HPC and Big Data Analytics workloads. It is concluded that refactoring RTM through the progressive introduction of Spark is feasible in the case of addressing I/O bound challenges that impede application of the method in practice; Spark presents additional possibilities for approaching RTM that includes the prospect of completely refactored imaging conditions and forward modeling through Deep Learning.**

I. Introduction

Artifacts (i.e., unwanted signals) are the bane of any image-based approach for remotely sensing an object of interest. In the case of reflection seismology for petroleum exploration, these artifacts are the unavoidable and natural consequence of acoustic waves interacting with complex geological structures such as folds, faults, domes, as well as steeply dipping lithologies (aka, tilted-rock) interfaces (e.g., [1]). Ironically, it is precisely these same subsurface features that geophysicists seek to map as accurately as possible in imaged sections or volumes via reflection seismology, as petroleum reservoirs may be associated with one or more of these features. Fortunately, through the process of migration, the impact of these artifacts can be minimized, thus affording geophysicists the opportunity to focus on geologically relevant signal – as opposed to artifact-generated noise. Although there are a number of accepted methodologies for migrating data obtained through reflection seismology, there is little doubt that Reverse Time Migration (RTM) most-accurately depicts all reflectors in their actual locations in space and time [1], [2]. Of course, this accuracy typically carries with it a significant computational burden.

RTM's computational burden characterizes the past, present and (apparently) future of the method. Although it was originally conceived in the 1980s [2], it took almost two decades before RTM became computationally tractable. The mere existence of contemporary investigations indicates that performance remains an ongoing concern for those seeking to employ the method in routine processing workflows. In the following section, namely §II, the status quo in RTM performance is briefly reviewed to provide context for the present investigation; and towards the end of paper in §VII, additional context is provided – especially as it relates to how others are currently addressing the computational challenges imposed by RTM. The availability of ever-improving computational capabilities increases the appetite for higher-resolution modeling and simulation in science and engineering. In this regard, practitioners of RTM are no different in seeking higher-definition images of the subsurface – even though these images require wave-equation modeling that scales as f^4, where f refers to the frequency of the modeled waves [3], [4]. In other words, addressing performance challenges related to RTM has been of relevance, continues to be of relevance, and is anticipated to remain of relevance well into the future.

It is precisely this ongoing tension with RTM's computational burden that motivates the current undertaking. In a significant departure from previous approaches aimed at reducing the impact of RTM's known performances challenges (please see §II for a brief statement of the same), emphasis here is placed on the introduction of approaches typically employed in disciplines other than the sciences or engineering. Specifically, the current effort initiates a refactoring of RTM through the progressive introduction of Apache Spark [5] – a comprehensive and integrated platform for Big Data Analytics. After very briefly reviewing Apache Hadoop [6] in §III-A, the more-compelling case for Apache Spark is made

in the remainder of §III. A toy example (please see §IV-C) is contrived to illustrate how one of RTM's key performance challenges can be directly addressed through use of Apache Spark. More specifically, Apache Spark's propensity for in-memory computing is exploited to reduce I/O operations during time series analysis involving cross-correlation – i.e., during the application of RTM's imaging condition (please see §II first for additional details). Although the toy example is prototyped on a laptop (please see §IV-D), scaling to a computational cluster with significant data stores is a matter that also receives consideration (please see §V).

As an implementation, Apache Spark possesses the primitives for distributed-memory parallel computing. It would appear, therefore, that Spark could also be applied to the finite-difference kernel that is central to the wave-propagation computations that are required in the forward- and reverse-modeling components of RTM – components for which additional context is provided in §II. The introduction of other possibilities for Apache Spark, from alternate imaging conditions to modeling kernels, are alluded to in §V. Conclusions are drawn in §VI, and related work is summarized briefly in §VII.

II. Performance Challenges in Conventional RTM

Whereas interferometry exploits interference phenomenon between pairs of signals to expose *differences* [8], RTM migrates seismic-reflection data by making use of *coherence* to seek similarities (e.g., [1]). Seismic traces, the observations recorded by an array of geophones (receivers sensitive to seismic waves) subsequent to triggering by some event, represent one of the two signals required for RTM. The other is generated by forward modeling. In principle then, with the two wavefields, the requisites for RTM are met. Of course, in practice, there are details requiring consideration; and, with a bias towards performance challenges, it is aspects of these details elaborated next.

In RTM, use is made *twice* of the three-dimensional wave equation (WE3D) to propagate seismic waves. First in the forward problem, assumptions are made about the characteristics of the seismic source as well as variations in subsurface velocity, so that seismic waves can be propagated forward in time from their point of origin (at the source) into the subsurface (i.e., areas/volumes of geological interest); this results in the forward or source wavefields in the upper-branch of Figure 1. Using seismic traces recorded by arrays of geophones, as well as an assumed subsurface-velocity model, these observations are reversed-in-time (and hence the name, RTM), and then backwards propagated using the same WE3D; this results in the receivers wavefields in the lower-branch of Figure 1.

It is standard practice in RTM to make use of the Finite Difference Method (FDM) to numerically propagate all wavefields in space and time using computational resources. In order to ensure meaningful results (i.e., stable and non-dispersive from the perspective of numerical analysis) from the application of the FDM to the WE3D, however, both time and 3D space need to be discretized into small steps and grid intervals, respectively (e.g., [7]). Because the WE3D is a Partial Differential Equation in time and space, the FDM estimates future values using approximations for all partial derivatives. And in practice, it has been determined that RTM requires high-order approximations for all spatial derivatives if reliable results are to be optimally obtained (e.g., [7]). In other words, the RTM modeling kernel is inherently and unavoidably compute intensive.

The forward and reverse propagation, described previously using the WE3D via the FDM, is carried out in two steps. After completion of the forward problem, the resulting source wavefields are written to disk in conventional RTM (please see Figure 1 for additional context). It is important to note that writing-to-disk is a *requirement*, as the data volumes involved in reflection seismology rapidly exhaust the capacity of physical memory. In a subsequent step, and for each source-receiver pair (aka. gather), source wavefields are read from disk so that they can be cross-correlated with the backwards-propagated (i.e., time-reversed) wavefields corresponding to the receivers – a step that again requires application of the FDM modeling kernel for the WE3D, on a per-timestep basis (e.g., [7]). Cross-correlation of the two wavefields is known as RTM's **imaging condition** [1] – i.e., the means through which RTM establishes coherence, and therefore similarities between the wavefields. Again, it is the similarities between the two wavefields that is interpreted as being of geological interest – i.e., the identification in space and time of geological reflectors like (steeply dipping) interfaces between different sedimentary lithologies, folds, faults, salt domes as well as (ultimately) reservoirs of even more complex geometrical structure. The inherent requirement for disk I/O involving multiple TB volumes of seismic-reflection data, during the application of RTM's imaging condition, results in a well-established performance bottleneck (e.g., [7]). Given that data volumes are expected only to increase, especially in the case of marine-based acquisition of seismic-reflection data (e.g., [4]), existing concerns relating to performance will be amplified.

To summarize, and in following [7], conventional RTM is burdened with two, primary challenges that are performance-related in nature:

1) Propagating seismic wavefields is computationally intensive, owing to appropriate use of the FDM on the WE3D. Because both forward and reverse wavefields require propagation, the FDM modeling kernel needs to be applied twice.
2) The application of RTM's imaging condition requires disk I/O in order for cross-correlation between forward and reverse wavefields. Current and anticipated data volumes are responsible for this situation.

Although it is anticipated (please refer to §V) that the approach taken here has at least the *potential* to address both of these performance-related challenges with conventional RTM, attention is focused here on the latter one. Specifically, through use of the approach detailed here, a significantly reduced

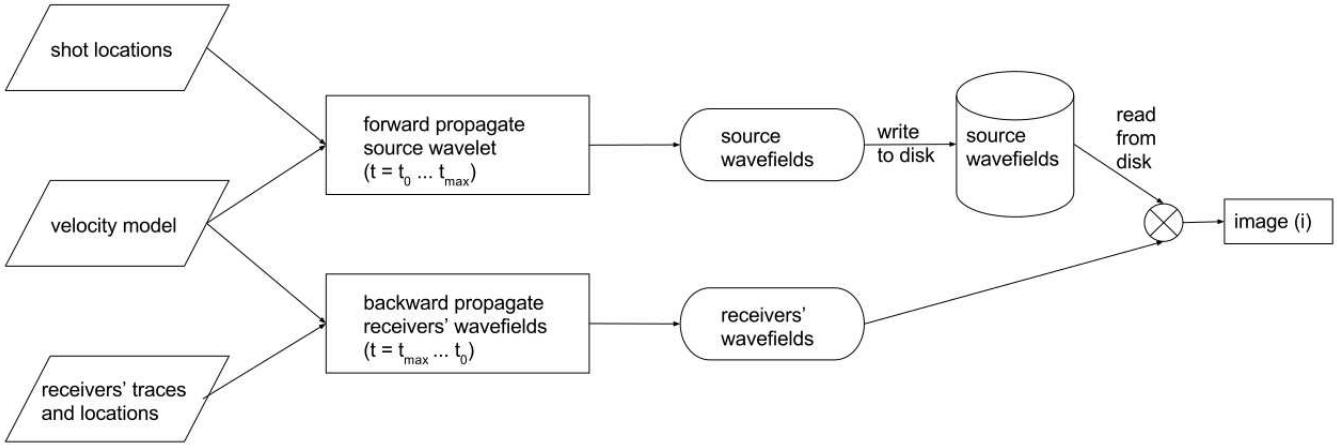

Fig. 1. Flowchart representation for conventional RTM; informed by the algorithmic description in [7].

reliance on disk I/O during application of RTM's imaging condition is sought.

III. Parallelism via Big Data Analytics

In confronting RTM's known performance challenges (II), the broad objective of this initiative was to seek improvements through the introduction parallelism by exploiting technologies available from Big Data Analytics. After briefly considering and dismissing Apache Hadoop in §III-A, attention focuses on in-memory distributed computing via Apache Spark in the remainder of the section.

A. Parallelism via Apache Hadoop

At the time this effort was initiated in circa 2015, Apache Hadoop presented as a fairly obvious starting point [9]; a choice that appeared consistent with comparable studies in the literature – e.g., processing large-scale seismic event data via waveform cross correlation [10]. Hadoop presented as extremely enticing to a discipline already awash in extremely large volumes of data: A distributed file system, known as the Hadoop File System (HDFS), that is effectively limitless in scale and built from commodity components (e.g., [6], [11]).

Of course if Hadoop were only about HDFS, a distributed, resilient, high-capacity file system, that would interesting but hardly remarkable as viable storage incumbents have a well-established record of addressing the requirements of reflection seismology in petroleum-exploration contexts. By leveraging capabilities inherent in HDFS, Hadoop facilitates data-local computing – in other words, workloads are proactively executed with innate awareness of topology (e.g., [6], [11]). Even in the simplest case involving Map-Reduce workloads, the locality of data and compute resources is a core competence [12]. Subsequent improvements (e.g., YARN [13]) enable additional capabilities.

Counterintuitive to existing practices in HPC, compute-data locality in Hadoop is achieved by treating storage as a resource to be proactively squandered [9]. In fact, Hadoop takes advantage of commodity storage resources to boost reliability by replicating data across multiple physical disks distributed across multiple physical systems across an entire infrastructure [6]. Thus Hadoop delivers a distributed, high-capacity, reliable, parallel file system *optimized* for compute-data locality.

Even though it is in itself a collection of services, HDFS represents only the underpinnings of the Hadoop data platform, as there are numerous additional and complimentary components [6], [14]. In addition to data-platform components, there are numerous analytics applications that allow for graph analytics, machine learning, stream processing, and much more [6], [14]. Particularly notable in the current context is Apache Flink, and its innate ability for in-memory processing [15].

Hadoop, and the ecosystem of components around it, clearly presents interesting opportunities for the introduction of parallelism via Big Data Analytics. And although it appears promising that aspects of RTM's known performance challenges could indeed be addressed by its introduction, the possibilities afforded by Apache Spark appear even more compelling; it is these possibilities that comprise the focus of the next section (§III-B).

B. Parallelism via Apache Spark

1) Strategic Considerations: Although it is the implementation that deservedly receives attention, it is actually the underlying computer science that ultimately makes Apache Spark substantial from a strategic perspective [16]. At its core, Spark derives its capabilities from Resilient Distributed Datasets (RDDs). As a relatively recent abstraction for distributed computing, RDDs are fault-tolerant, parallel data structures ideally suited to in-memory cluster computing [17]. Furthermore, Zaharia et al. [17] architected RDDs to be persistent and partitionable across a cluster to ensure that data is optimally placed for computation – a capability it shares in common with Hadoop (§III-A). Finally, RDDs can be manipulated using a rich set of operators [17]. Originally conceived to address limitations inherent in MapReduce [17], RDDs are

so compelling in delivering the primitives for distributed-memory parallel computing, that some (e.g., [18], [19]) have enthusiastically opined them as a potential successor for MPI (the Message Passing Interface, [20]) – the de facto standard for distributed-memory parallel computing in HPC.

Of course, elegant computer science alone does not compel one to even partially refactor voluminous code, such as that involved in RTM; results, however, have the potential to up the strategic ante. In the original benchmarking studies [17], Spark outperformed Hadoop by a factor of 20 in a best-case scenario for Hadoop - involving use of binary data and an in-memory HDFS instance. Spark even outperformed Hadoop by a factor of 10 when memory is unavailable and it has to use disks [5]. RDDs aside, there is clearly substance to approaches that employ in-memory computing, as Apache Flink is reporting results comparable to those obtained by Spark in more-recent benchmarking studies (e.g., [21]). From open-source indicators such as Spark and Flink, to (refactored) offerings from for-profit juggernauts (e.g., [22], [23]), there is little doubt that in-memory computing is extremely efficient in delivering results with appealing speed. Because in-memory computing continues to generate compelling outcomes, benchmarking comparisons are only an Internet search query away.

As a final strategic consideration, uptake and support serve only to amplify the impetus for adopting Spark; from an actively engaged community of contributors and users, to supported platforms for Big Data Analytics, the value of Spark is increasingly evident (e.g., [5], [24], [25], [26], [16]).

Strategic considerations aside, the practicalities of refactoring applications such as RTM is the focus of the following section (§III-B2).

2) Practical Considerations: Although a sound strategic imperative may have been identified, the practicalities of systematically refactoring an application like RTM may circumvent any well-intended effort. Fortunately, in making use of Apache Spark for refactoring RTM, there is optimism regarding these practicalities. In approaching this refactoring from the perspective of RTM in an existing, traditional HPC context, infrastructural and development-centric considerations emerge.

From an infrastructural perspective, storage- and workload-management considerations predominate. Whereas Hadoop has an implicit dependency upon HDFS, Spark does not. In fact, Spark offers significant interoperability with alternatives to HDFS that include on-premise (e.g., Ceph, GPFS, Lustre) and cloud-based (e.g., AWS S2, OpenStack Swift) possibilities (e.g., [16]). Because the vast majority of organizations involved in processing exploration seismic data have an incumbent storage infrastructure, the prospect of re-use with simultaneous introduction of Spark, can be extremely enabling ([27], [18], [19]). In practice, this interoperability is often achieved via connectors or adapters – e.g., there exists a connector that allows HDFS to be effectively replaced by Lustre [28].

Although Spark includes a standalone capability for managing workload [29], use of YARN may be preferable in Hadoop clusters (e.g., [5], [24], [26]) – e.g., for integrated management of Spark and Hadoop workloads. Apache Mesos [30] is of particular interest in this context, as it can directly or indirectly allow for management of Spark workloads. In fact, it is by leveraging the Mesos-compliant framework, that Univa Universal Resource Broker [31] is empowered with the ability to manage Spark workloads in a cluster that is hybridized between Big Data Analytics and HPC; this is a subject that receives additional consideration (please see §V) in regard to early experiences in refactoring RTM via Spark. Additional possibilities for managing Spark workloads are alluded to elsewhere ([16]). Storage infrastructure notwithstanding, production deployments of Spark can be targeted for on-premise or cloud-based clusters. And within these clusters, Spark can be deployed directly on bare metal, within virtual machines, or even containerized. The highly dynamic nature of containerized deployments involving Docker [32] is addressed by some solutions for workload management (e.g., [33]), and warrants consideration beyond the current scope.

With the most-obvious practicalities for infrastructure set aside for the moment, attention can shift to developer-centric considerations for refactoring RTM through use of Apache Spark. That Spark offers the primitives for distributed-memory parallel computing is an extremely appealing capability, in principle (please see §III-B1). Pragmatically speaking, however, it is likely to be Spark's ability to leverage existing code (as shared libraries) written in C or C++ via Java Native Access (JNA, [34]) that is of more-immediate value in refactoring RTM. When it comes to completely refactoring code for RTM, however, Spark's native support for Java, Scala and Python ([5]) might be of increased value.

As summarized in §VII, GPU programmability via CUDA ([35]) has spurred the development of innovative algorithms that have been responsible for impressive improvements in the performance of RTM (e.g., [7]). Despite the possibilties that Spark presents, preserving existing investments in HPC via GPU technology is quite likely to remain a priority. Fortunately, HeteroSpark is already making this posssible in allowing Spark to selectively execute on GPUs as well as CPUs [36].

From strategic to practical considerations, the case for refactoring RTM via the possibilities presented via Apache Spark is compelling – more compelling than options available via Hadoop. And it is for these reasons that RTM's performance challenges are shortly addressed through use of Spark.

IV. Refactoring RTM via Apache Spark

A. Toy Problem for RTM: Introducing RDDs

Despite the acknowledged utility of RTM (§II), very few examples of open-source implementations appear to exist – even in support of textbooks on the analysis of seismic data (e.g., [1], [37]). Of course as implied in §II, and particularly as elucidated via Figure 1 along with the detailed algorithmic description provided by Liu et al. [7], a complete and functional implementation of RTM is quite involved (e.g., [38]) and definitely beyond the current purposes and intents. Instead, to

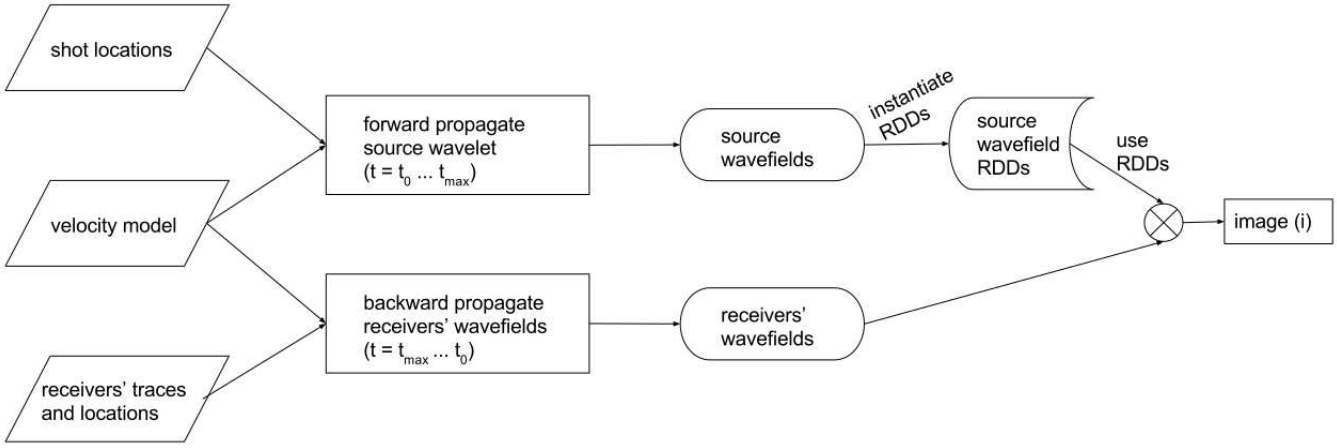

Fig. 2. Modified flowchart representation for RTM with conventional I/O replaced by RDDs (after [27]).

effectively problematize the performance challenges inherent in conventional RTM, the current effort places emphasis on a toy problem as the expository vehicle.

To fix ideas for the purpose of the toy problem, the RTM implementation flow charted previously in Figure 1 is seemingly modestly refactored through the introduction of RDDs in Figure 2. Most notable in this latter figure is the 'replacement' of disk I/O. To achieve this replacement, forward and reverse wavefields are simultaneously manipulated in-memory as RDDs. Although use of RDDs does not obviate the need for disk I/O completely, as wavefield data does need to be read from disk at some point to 'populate' the RDDs, the approach certainly does succeed in *decoupling* disk I/O from application of RTM's imaging condition – a condition that can now be applied solely in memory (§IV-C). Even in this toy problem then, RTM's imaging condition can be applied as an in-memory computation (i.e., a cross-correlation between RTM's forward and reverse wavefields) on data that resides entirely in memory as RDDs. In other words, the toy problem is designed to directly address the I/O bottleneck identified previously (§II) as one of the two key performance challenges associated with the conventional approach for RTM.

The ability to completely decouple wavefield disk I/O from the computation (Figure 3), as required in application of RTM's imaging condition, is one of the significant contributions of the present approach. Given the data volumes typically involved seismic exploration for potential petroleum reserves (§I), this decoupling is of inherent value. As an added-value consequence, this decoupling allows for construction of a dependent workflow. In other words the rate-determining step of populating the RDDs, with the forward and reverse wavefields, could be conveniently staged as a prerequisite for the latter step of the compute-itensive cross-correlation. In the simplest case, RDDs are populated by reading from one of file systems supported by Spark – a partial list was provided previously in §III-B2. Spark, however, interoperates with numerous data sources other than pure file systems [39]; and it is possible, of course, that this interoperability may improve upon the performance of populating RDDs. In §V this subject is revisited, as Spark now offers alternatives to RDDs as they were originally conceived [17] and implemented.

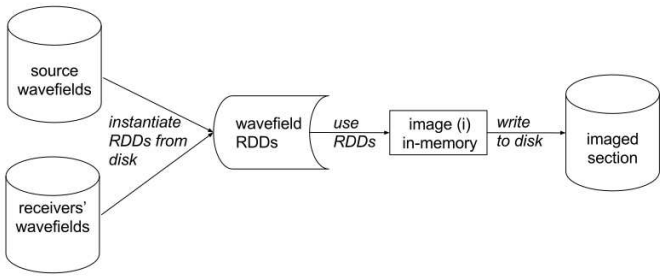

Fig. 3. RDD introduction results in decoupling disk I/O from computation of RTM's imaging condition.

B. Toy Problem for RTM: Introducing Spark via Thunder

As noted previously (§III-B2), Apache Spark provides native implementations in Java, Scala, Python and R. For rapid prototyping in the case of the toy problem introduced here (in §IV-A), it was decided that use of Spark's support for Python would prove effective and efficient. Moreover, this choice of Python allowed for use of the Thunder Project – an open-source library for image and time-series analysis originally developed for the neurosciences [40].

Thunder is written in Spark's Python API (PySpark, [41]) and makes use of well-established, science-enabling Python packages such as numpy [42], scipy [43], and others [44]. When used in tandem with Jupyter Notebooks [45], the resulting development environment is ideal for interactive prototyping with graphics support – please see §IV-D for the specific case involving refactoring RTM and [46] for generic examples relating to Thunder.

After initializing packages (e.g., to enable graphics and scientific computing) for use as necessary, and establishing a PySpark context [41] via `ThunderContext` [47], a development platform for refactoring RTM with Thunder via Spark is rapidly established.

One of the primary reasons Thunder proved extremely appealing, in enabling refactoring of RTM via Spark, was its existing availability of support for time-series analysis ([18], [19]). In particular, the `thunder.TimeSeries` class [48] includes `crossCorr`, a method for cross-correlation [49]. Unfortunately, this class (and method) are limited in their current implementation, as collectively they only support the cross-correlation of time series against a single target [50]. The next-major release of Thunder will also not address this limitation [51]. Fortunately, Bennett states [52]: "... a general rule is that if [T]hunder doesn't have some [required] functionality ... it's usually possible to get that functionality easily with the interface to the underlying [RDD.]" In the following section (§IV-C), Thunder is 'extended' to include a cross-correlation capability that can handle two time series – i.e., in order to ensure it is directly applicable to this mode of comparison between RTM's forward and reverse wavefields.

It was noted previously (§III-B2) that Spark can access C or C++ shared libraries via JNA. Thunder, in drawing upon its Python lineage, can interface with code written in C/C++ as well as Fortran in various ways (e.g., [53], [54]). Because situations may arise where the ability to interface with external libraries, binaries, etc. is required, it is reassuring to note that this is indeed a possibility with Thunder – a capability not lost on those who intend to refactor code through the systematic introduction of RDDs and Spark.

C. Toy Problem for RTM: Spark Implementation via Thunder

Because Thunder's existing capability for cross-correlation does not allow for application to an arbitrary pair of time series, as is needed here to cross-correlate RTM's forward and reverse wavefields, it is necessary that Thunder itself be 'extended'. As indicated previously (please refer to §IV-B), extensions to Thunder are possible by making use of the interface to the underlying RDD.

Listing 18 provides a concrete implementation for the toy problem. First, RDDs are instantiated into Thunder Context `tsc` through use of the `loadSeries` method from Thunder's `ThunderContext` class; thus, `series1` and `series2` are in-memory representations of the forward and receivers' wavefields. Next, function `rtm_ic` is employed to define the RTM imaging condition via cross-correlation by making use of the existing FFT (Fast Fourier Transform) convolution capability in scipy [55]. The RDD-enabled version of this 'extended' capability, namely `rtm_it_rdd`, is subsequently defined by simultaneously making use of the interface to the underlying RDD via PySpark RDD [41], as well as the `join()` and `mapValues()` methods available from this same class. The net result is a capability for cross-correlating RTM's two time series in memory through interactive use of a Jupyter Notebook that has invoked Thunder.

Even though this is only a toy problem, the intended purpose has been demonstrated – i.e., an *in-memory* assessment of the coherence between two seismic wavefields. Moreover, conventional RTM's innate requirement to employ disk I/O *during* the application of its imaging condition (i.e., the estimation of coherence between wavefields via cross-correlation) has been removed completely (Figure 1), and replaced by an in-memory computation (Figure 2). Of course, disk I/O is required to initially populate the RDDs with data; once populated, however, there is no ongoing requirement for disk I/O – and certainly not during the computation of the imaging condition (Figure 3). Although this concludes the successful demonstration of the toy problem, remarks regarding the environment used in prototyping are summarized in the next section (§IV-D); potential next steps in the progressive refactoring of RTM are considered in §V.

```
# instantiate RDDs from disk
import os.path as pth
seriespath = pth.
  join(pth.dirname(pth.realpath(thunder.__file__)),
    '/<srcdir>/')
series1 = tsc.loadSeries(seriespath + 'trace1.dat',
  nkeys=1, nvalues=1, inputFormat='text')
series2 = tsc.loadSeries(seriespath + 'trace2.dat',
  nkeys=1, nvalues=1, inputFormat='text')
# define imaging condition
def rtm_ic(a,b):
  from scipy.signal import fftconvolve
  return fftconvolve(a, b[::-1])
# apply imaging condition in memory
rtm_ic_rdd = series1.rdd.join(series2.rdd).
  mapValues(fun(series1.values().take(data.nrecords),
    series2.values().take(data.nrecords)))
```

Listing 1. Python implementation via Thunder of the toy problem described in §IV-C. After instantiating RDDs from disk, RTM's imaging condition is first defined, and then applied as an in-memory computation.

D. Prototyping Environment

Although Apache Spark is intended for clusters [5], for prototyping purposes, a complete, highly portable development environment was established on a laptop in the following three steps [56]:

1) **Spark** A recent, pre-built version of Apache Spark was downloaded, the contents of the tarball extracted, and placed in a convenient location (`/opt`). Setting and exporting the `SPARK_HOME` environment variable is all that is required for Thunder to make use of Spark.
2) **Anaconda** Anaconda is an open platform for Data Science that is built with Python [57]. Even if a Python distribution is already available, it is definitely worth considering Anaconda, as it will take care of dependencies, for instance. Support for Jupyter Notebook is included with Anaconda.
3) **Thunder** Using Python's package manager, pip [58], via Anaconda or a pre-existing Python deployment, Thunder is easily installed, and its prerequisites met.

Once installed, Thunder can be invoked at the command line by simply typing `thunder` at an interactive-shell prompt. Of course, this (and the following GUI offering) assumes that the `SPARK_HOME` environment variable has already been set. As noted previously (§IV-B), however, an interactive notebook capability is provided by Jupyter Notebook [45]. When `IPYTHON_OPTS="notebook --profile=nbserver"`

`thunder` is issued at an interactive-shell prompt, the Jupyter Notebook service starts in a Web browser, and communicates with a locally hosted Web server. From the instantiated Web page, pre-existing notebooks can be opened, or new ones created. Although the emphasis here has been Python, Jupyter Notebook supports over 40 programming languages [45].

From the introduction of RDDs, to Spark implementation (through use of Thunder and PySpark), to notebook GUI development environment (via Jupyter Notebook), a toy problem has demonstrated that conventional RTM's performance challenge with disk I/O can be refactored. In the following section, §V, this demonstration is considered in a broader context for RTM.

V. Discussion

In essence the purpose of this section is to review the current investigation from a critical perspective, and in so doing, indicate where additional effort needs to be placed.

The most-obvious place to start is with the toy-problem implementation presented here in §IV-C. Although there is likely to be some merit in further pursuing studies with synthetic data, especially if benchmarking is made a point of emphasis, of considerably greater value would be the introduction of realistic data. Once acquired, reflection-seismic data is written to an industry-standard format, known as "SEG Y rev 1", established by the Society of Exploration Geophysicists (SEG) in 2002 [59]. Yan et al. [60] have successfully represented sample reflection-seismic datasets in-memory as RDDs through use of a Spark cluster. In the current context, their work is encouraging, as it suggests the toy model can be successfully extended to incorporate realistic data. It is fair to note, however, that even Yan et al.'s effort serves only as a proof-of-concept demonstration; additional effort will be required to validate this approach for the TB-scale-and-greater data volumes that typify the petroleum industry at the current time (as alluded to in §II).

By design, the toy problem completely avoided the modeling kernel employed to propagate each of the forward and reverse wavefields (§II provides additional context). This was an admittedly significant oversight, as it is well known that implementations making use of the FDM are acknowledged as performance-challenged [7]. It is likely an even greater oversight in the context of applying Spark to a problem currently factored for HPC, and thus an more-stringent requirement than that posed here by the toy problem. Although addressing this oversight (for the RTM case) is planned, there is existing enthusiasm for the feasibility of such an undertaking. In particular, in applying Spark to the one-dimensional diffusion equation, Dursi [61] noted that the PySpark implementation requires about half as many lines of code, and one-tenth of the boilerplate when compared with an implementation in MPI. Not only is the PySpark implementation fault-tolerant, an inherent characteristic of RDDs (§III-B1), it is more-easily coded – as Spark developers are enabled to focus at a higher level of abstraction [61].

It might be argued that development on a laptop (i.e., §IV-D) is also a significant concern with the implementation of the toy problem (§IV-C). By installing Thunder into a pre-existing Spark cluster, however, the development environment described previously scales out seamlessly. In fact, as demonstrated previously by Lumb [18], it is the deployment of the Spark cluster itself that is the considerable undertaking; to significantly reduce the infrastructural burden, he made use of Bright Cluster Manager for Big Data [62] to first provision the physical cluster, then deploy Hadoop and Spark, and finally to monitor and manage the entire cluster on an ongoing basis. Although Lumb [18] only made use of HDFS as the underlying file system for Spark, he did experiment with different modalities for managing Spark workloads, as Spark's built-in workload manager as well as YARN were both available in his setup.

As noted previously in §III-B2, Univa Universal Resource Broker [31] possesses the ability to manage Spark workloads in a cluster that is hybridized between Big Data Analytics and HPC; it is notable that this interoperability is achieved through a broker that interfaces UURB as a Mesos-compliant framework. By virtue of this interoperability, the compilation of mature policies supported by Univa Grid Engine (UGE, [63]) are immediately available for scheduling Spark workloads, and the limited policies implicit in use of YARN or Spark itself are no longer a restriction. Because use of UURB is particularly well aligned with efforts that seek to introduce refactored code through the progressive introduction of RDDs via Spark, scale-out efforts under development will employ UURB in HPC clusters with some degree of enablement for Big Data Analytics. Because Spark minimizes the need for disk I/O, as the toy example well illustrated (§IV-C), HDFS (or any another Spark-compliant data source, as described in §III-B2) is largely decoupled during in-memory computations – a decoupling deemed here as appealing in the case of RTM, and most likely in numerous other use cases.

As indicated here (§II), in RTM cross-correlation serves as the canonical imaging condition for establishing coherence between forward and reverse wavefields. The introduction here of Spark presents an *analytics upside* – i.e., an almost frictionless opportunity to refactor RTM's imaging condition through use of Big Data Analytics. Although statements in this context rapidly degrade to the level of speculation (e.g., [64]), it is worth mentioning possibilities for Deep Learning (via Spark's built in Machine Learning library, MLlib, [65]) in establishing alternate imaging conditions, and even for improvements in the velocity model required for forward modeling with the WE3D.

Finally, Spark offers a DataFrames API as well as (most recently) Datasets [5]. With these abstractions that are distinct from, and yet build upon the fundamental abstraction of RDDs, serious refactoring has additional possibilities to consider – possibilities that may afford, for instance, improved performance in certain use cases (e.g., [66]).

With Spark continuing to evolve at a rapid pace, this is a snapshot of considerations worth discussing in the present

context at the present time. There is little doubt that the dialogue will be ongoing. Before briefly summarizing related work in §VII, conclusions are drawn in the following section (§VI).

VI. Conclusions

RTM has a past, present, and almost certainly, a future of computational challenge and opportunity in support of petroleum exploration (§I). Whereas there is clearly an impetus to employ the method that most-effectively removes artifacts from seismic-reflection data, contemporary use via conventional RTM suffers from two performance-related challenges (§II). With current and anticipated data volumes, and a pressing need for increased degrees of resolution, these challenges are exacerbated. In an effort to confront the performance-related challenges imposed by use of the method in realistic petroleum-exploration scenarios, GPU-based technologies have been applied successfully in confronting RTM's compute and I/O bound nature (§VII).

In seeking to address RTM's performance-related challenges through the introduction of parallelism available from Big Data Analytics, the present effort employed a radically different approach. The almost-natural tendency to gravitate towards Apache Hadoop was resisted, and instead attention focused on use of Apache Spark (§III-A). As an implementation, Spark is based on a compelling abstraction known as RDDs, and these RDDs have extremely appealing characteristics (§III-B). Spark delivers the primitives for distributed-memory parallel computing that are ideally suited to clusters. And because computations are performed in-memory, Spark has an established reputation for efficiently delivering results.

With emphasis on conventional RTM's performance being gated by disk I/O, Spark was introduced in theory (§IV-A) and in practice. A toy problem, that made use of synthetic data, demonstrated that in decoupling disk I/O, in-memory cross-correlation of wavefields proceeds efficiently with Spark (§IV-C). The choice to make use of PySpark via Thunder proved extremely appropriate in rapidly enabling an effective development environment and producing results (§IV-B and §IV-D). Extension of the toy problem to use cases more realistic of current practice in petroleum exploration was considered (§V). Because it has already been established that the industry-standard format seismic data can indeed populate RDDs in a Spark-based cluster ([60]), the ability to address conventional RTM's extant issue with disk I/O can be regarded as essentially validated through use of Spark.

Thus the most-pressing challenge for further validating uptake of Spark, in addressing conventional RTM's known performance-related challenges, involves the modeling kernel that is employed to forward and reverse propagate seismic wavefields. Although there is cause for optimism in principle, Spark's ability to be of use in this computationally intensive context needs to be validated; a matter that is currently receiving attention.

Beyond conventional RTM's known performance-related challenges, the utility of Spark could be surprisingly impactful. Already validated is Spark's ability for prototyping on an isolated laptop and relatively seamless scaling out to clusters (§V). In addition to on-premise deployments involving use of UURB, extension into public and private clouds warrants consideration. Considerably less clear is Spark's ability to introduce imagining conditions other than ones based purely on cross-correlation; while Spark renders such opportunities surprisingly accessible, use of Deep Learning for instance, requires considerable investigation.

In summary, and even at this relatively early stage, the prospects for Spark in refactoring RTM are nothing short of encouraging; there is clearly motivation and merit for further investigation. And of course the utility of the present and future investigations, though focused on RTM, is anticipated to provide insight for additional use case which are today being serviced solely by conventional HPC.

VII. Related Work

In this final section related work is summarized briefly.

Although the emphasis was not always RTM, the use of GPUs in processing seismic data dates back to circa 2008 in the private sector (e.g., [67], [68], [69]). And as various subsequent efforts demonstrate (e.g., [70], [7], [71]), there remains significant interest in use of this technology in the industry. Broadly speaking, in addressing RTM, algorithms have made effective and efficient use of both the memory hierarchy as well as parallel-processing capabilities inherent in GPGPUs (e.g., [7]). Interestingly, at least one private-sector organization bypassed the use of GPUs in their processing workflow, as their efforts lead them to address other priorities first in a broad-scale refactoring of their code and even their infrastructure [4]; this organization, however, has not ruled out a potential role for GPU-enabled seismic processing in the future. Although it is beyond the current scope to review the performance improvements achieved through the introduction of GPU-enabled seismic processing, even anecdotal indicators point to the potential for double-digit speedups in comparisons to more-conventional approaches involving purely CPUs – with some of these gains being specific to RTM.

Pioneering efforts to apply Big Data Analytics to the processing of seismic data is, by comparison, understandably a much-more recent undertaking. For instance, Addair et al. [10] made use of Apache Hadoop in global seismology; of particular interest in the current context (§III-A), was their requirement to process large-scale seismic event data via waveform cross correlation. In effectively exploiting the compute-data locality available from the combination of MapReduce and HDFS, a speedup factor of 19 was reported, though effort was required to refactor their use case for optimal use of Hadoop [10]. Although use of Hadoop was considered and dismissed here in §III-A, anecdotal evidence suggests that there are indeed organizations currently considering its use in various seismic-processing contexts.

Even more understandable is the absence of literature on uptake of Apache Spark in use cases for processing seismic data. Whereas 2015 might be regarded as the watershed year

in which Spark began receiving consideration for seismic processing in general, at least one effort pre-dates this interest [72]. Although the overall intention of the effort is to develop a comprehensive platform for seismic analytics (e.g., [72]), Yan et al. [60] demonstrated an ability to populate RDDs with SEG Y format seismic data using this Spark-based platform – a demonstration regarded here in §Vto be of significant value in the RTM context. In a more recent effort, the same platform allowed for successful detection of geological faults in seismic data via the support for Machine Learning in Spark's MLlib [73]. After initially considering and dismissing Hadoop, as reviewed here in §III-A and §III-B, respectively, Lumb [27] initiated efforts to refactor RTM through the progressive introduction of RDDs via Spark. Although subsequent efforts (e.g., [18], [19] and the present one) have focused on only RTM's known performance challenge with disk I/O (§II), there is an intention to consider RTM's compute-intensive modeling kernel (§II) in a subsequent effort.

ACKNOWLEDGMENTS

The author would like to thank the communities behind Apache Spark and the Thunder Project, as any refactoring of RTM is achieved through use of the implementations they have developed and made available.

REFERENCES

[1] H.-W. Zhou, *Practical Seismic Data Analysis*. Padstow, Cornwall, UK: Cambridge University Press, 2014.

[2] J. B. Bednar, "A brief history of seismic migration," *Geophysics*, vol. 70, no. 3, pp. 3MJ–20MJ, 2005.

[3] F. Alabert, "How is High Performance Computing reshaping O&G exploration and production," presented at the Rice University Oil & Gas HPC Conference, 2016. [Online]. Available: http://sched.co/5sUz

[4] S. Brandsberg-Dahl, "Exploration Seismology and the Return of the Supercomputer," presented at the Rice University Oil & Gas HPC Conference, 2016. [Online]. Available: http://sched.co/5w5o

[5] Apache Spark. [Online]. Available: http://spark.apache.org/

[6] Apache Hadoop. [Online]. Available: http://hadoop.apache.org/

[7] G. Liu, Y. Liu, L. Ren, and X. Meng, "3D seismic reverse time migration on GPGPU," *Computers & Geosciences*, vol. 59, pp. 17 – 23, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0098300413001519

[8] A. Curtis, P. Gerstoft, H. Sato, R. Snieder, and K. Wapenaar, "Seismic interferometry - turning noise into signal," *The Leading Edge*, vol. 25, no. 9, pp. 1082–1092, 2006.

[9] I. Lumb. (2015) Seismic migration using Hadoop: How did i get here? blog post. [Online]. Available: http://www.brightcomputing.com/blog/seismic-migration-using-hadoop-how-did-i-get-here

[10] T. Addair, D. Dodge, W. Walter, and S. Ruppert, "Large-scale seismic signal analysis with Hadoop," *Computers & Geosciences*, vol. 66, pp. 145 – 154, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0098300414000259

[11] T. White, *Hadoop: The Definitive Guide*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2009.

[12] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[13] Apache YARN. [Online]. Available: https://hadoop.apache.org/docs/r2.7.1/hadoop-yarn/hadoop-yarn-site/YARN.html

[14] The Hadoop Ecosystem Table. [Online]. Available: https://hadoopecosystemtable.github.io/

[15] Apache Flink. [Online]. Available: https://flink.apache.org/

[16] I. Lumb. (2015, Mar.) 8 Reasons Apache Spark is So Hot. insideBIGDATA. [Online]. Available: http://insidebigdata.com/2015/03/06/8-reasons-apache-spark-hot/

[17] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 2–2.

[18] I. Lumb, "Reverse Time Migration via Resilient Distributed Datasets: Towards In-Memory Coherence of Seismic-Reflection Wavefields using Apache Spark," presented at the High Performance Computing Symposium (HPCS), 2015. [Online]. Available: https://2015.hpcs.ca/wp-content/uploads/2015/06/Book-of-Contributed-talks-abstracts.pdf

[19] ——, "Reverse Time Migration via Resilient Distributed Datasets: Towards In-Memory Coherence of Seismic-Reflection Wavefields Using Thunder via Apache Spark," presented at the Rice University Oil & Gas HPC Conference, 2016. [Online]. Available: http://sched.co/5sVC

[20] The Message Passing Interface (mpi) standard. [Online]. Available: http://www.mcs.anl.gov/research/projects/mpi/

[21] S. Ewen, "Introducing Apache Flink: Fast and reliable data analytics in clusters," presented at the Strata + Hadoop World, 2015. [Online]. Available: http://conferences.oreilly.com/strata/big-data-conference-uk-2015/public/schedule/detail/39703

[22] Oracle Database In-Memory. [Online]. Available: http://www.oracle.com/us/products/database/options/database-in-memory/overview/index.html

[23] OSAP HANA On Premise Deployment: In-Memory Database Platform. [Online]. Available: http://go.sap.com/canada/product/technology-platform/hana-on-premise.html

[24] Apache Spark. [Online]. Available: https://www.cloudera.com/products/apache-hadoop/apache-spark.html

[25] Databricks. [Online]. Available: https://databricks.com/product/databricks

[26] Apache Spark & Hadoop. [Online]. Available: http://hortonworks.com/hadoop/spark/

[27] I. Lumb, "RTM Using Hadoop: Is There a Case for Migration?" presented at the Rice University Oil & Gas HPC Conference, 2015. [Online]. Available: http://sched.co/2DhB

[28] Intel Enterprise Edition for Lustre software. [Online]. Available: http://www.intel.com/content/www/us/en/software/intel-enterprise-edition-for-lustre-software.html

[29] Spark Documentation - Latest Version. [Online]. Available: http://spark.apache.org/docs/latest/

[30] Apache Mesos. [Online]. Available: http://mesos.apache.org/

[31] Univa Universal Resource Broker. [Online]. Available: http://www.univa.com/resources/files/urb.pdf

[32] Docker. [Online]. Available: https://www.docker.com/

[33] Univa Grid Engine - Container Edition. [Online]. Available: http://www.univa.com/resources/files/gridengine_container_edition.pdf

[34] Java Native Access. [Online]. Available: https://github.com/java-native-access/jna

[35] CUDA Parallel Computing Platform. [Online]. Available: http://www.nvidia.ca/object/cuda_home_new.html

[36] P. Li, Y. Luo, N. Zhang, and Y. Cao, "HeteroSpark: A heterogeneous CPU/GPU Spark platform for machine learning algorithms," in *Networking, Architecture and Storage (NAS), 2015 IEEE International Conference on*. IEEE, 2015, pp. 347–348.

[37] B. Biondi, *3D seismic imaging*. Society of Exploration Geophysicists Tulsa, 2006, no. 14.

[38] N. Moussa, "Seismic imaging using GPGPU accelerated reverse time migration," *CS 315A Parallel Computer Architecture and Programming*, 2009.

[39] Spark SQL, DataFrames and Datasets Guide. [Online]. Available: http://spark.apache.org/docs/latest/sql-programming-guide.html#data-sources

[40] Thunder. [Online]. Available: http://thunder-project.org/

[41] Spark Python API Docs. [Online]. Available: http://spark.apache.org/docs/latest/api/python/

[42] NumPy. [Online]. Available: http://www.numpy.org/

[43] SciPy. [Online]. Available: http://www.scipy.org/

[44] Thunder documentation. [Online]. Available: http://thunder-project.org/thunder/docs/

[45] Project Jupyter. [Online]. Available: http://jupyter.org/

[46] Thunder live notebooks. [Online]. Available: hhttp://w2.notebooks.codeneuro.org:8000/user/zOH3PaGViTPn/notebooks/thunder/tsmg.ipynb

[47] Thunder context. [Online]. Available: http://thunder-project.org/thunder/docs/tutorials/thunder_context.html

[48] Thunder API - thunder.TimeSeries. [Online]. Available: http://thunder-project.org/thunder/docs/generated/thunder.TimeSeries.html#thunder.TimeSeries

[49] Thunder API - thunder.TimeSeries.crossCorr. [Online]. Available: http://thunder-project.org/thunder/docs/generated/thunder.TimeSeries.html#thunder.TimeSeries.crossCorr

[50] J. Freeman. Nature of thunder.TimeSeries.crossCorr. [Online]. Available: https://gitter.im/thunder-project/thunder?at=57040c82f504b375561b70e9

[51] Thunder Project - series.py. [Online]. Available: https://github.com/thunder-project/thunder/blob/1.0.0/thunder/series/series.py#L896

[52] D. Bennett. Missing functionality in Thunder. [Online]. Available: https://gitter.im/thunder-project/thunder?at=57040c82f504b375561b70e9

[53] Extending Python with C or C++. [Online]. Available: https://docs.python.org/2/extending/extending.html

[54] Using Python as glue. [Online]. Available: http://docs.scipy.org/doc/numpy-1.10.1/user/c-info.python-as-glue.html

[55] scipy.signal.fftconvolve. [Online]. Available: http://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.signal.fftconvolve.html

[56] Thunder installation. [Online]. Available: http://thunder-project.org/thunder/docs/install_local.html

[57] Why Anaconda? [Online]. Available: https://www.continuum.io/why-anaconda

[58] pip 8.1.1 docucumentation. [Online]. Available: https://pip.pypa.io/en/stable/

[59] (2002) SEG Y rev 1 Data Exchange format. [Online]. Available: http://www.seg.org/documents/10161/77915/seg_y_rev1.pdf

[60] Y. Yan *et al.*, "A Spark-based Seismic Data Analytics Cloud," presented at the Rice University Oil & Gas HPC Conference, 2015. [Online]. Available: http://sched.co/2NFw

[61] J. Dursi. (2015) HPC is dying, and MPI is killing it. blog post. [Online]. Available: http://www.dursi.ca/hpc-is-dying-and-mpi-is-killing-it/

[62] Bright Cluster Manager for Big Data. [Online]. Available: http://www.brightcomputing.com/product-offerings/bright-cluster-manager-for-big-data

[63] Univa Grid Engine. [Online]. Available: http://www.univa.com/resources/files/gridengine.pdf

[64] I. Lumb. (2015) Possibilities for Reverse-Time Seismic Migration (RTM) using Apache Spark. blog post. [Online]. Available: https://ianlumb.wordpress.com/2015/04/01/possibilities-for-reverse-time-seismic-migration-rtm-using-apache-spark/

[65] Apache Spark MLlib. [Online]. Available: http://spark.apache.org/mllib/

[66] A. Grove. (2016) Apache Spark: RDD, DataFrame or Dataset? blog post. [Online]. Available: http://www.agildata.com/apache-spark-rdd-vs-dataframe-vs-dataset/

[67] B. Deschizeaux and J.-Y. Blanc, "Imaging Earth's Subsurface Using CUDA," in *GPU Gems 3*, H. Nguyen, Ed. Boston, MA, USA: Pearson Education, Inc., 2008, ch. 38. [Online]. Available: https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch38.html

[68] S. Morton, "Experiences with Seismic Algorithms on GPUs," presented at the Rice University Oil & Gas HPC Workshop, 2008. [Online]. Available: http://citi2.rice.edu/OG-HPC-WS/Scott%20Morton-Hess.pdf

[69] J. Panetta, T. Teixeira, P. R. P. Souza Filho, C. A. Cunha Filho, D. Sotto, F. M. Roxo da Motta, S. S. Pinheiro, A. L. Romanelli Rosa, L. R. Monnerat, L. T. Carneiro, and C. H. B. Albrecht, "Accelerating Time and Depth Seismic Migration by CPU and GPU Cooperation," *International Journal of Parallel Programming*, vol. 40, no. 3, pp. 290–312, 2011. [Online]. Available: http://dx.doi.org/10.1007/s10766-011-0185-2

[70] T. Cullison, R. Gandham, and S. Morton, "Revisiting Kirchhoff Migration on GPUs," presented at the Rice University Oil & Gas HPC Conference, 2015. [Online]. Available: http://sched.co/2DWr

[71] A. Modave, A. St-Cyr, and T. Warburton, "Performance of DGTD Finite Element Methods for the RTM Procedure on GPU Clusters," presented at the Rice University Oil & Gas HPC Conference, 2016. [Online]. Available: http://sched.co/5sVF

[72] L. Huang, "Spark-Based Seismic Data Analytics Cloud," presented at the The Society of HPC Professionals (SHPCP), 2014. [Online]. Available: http://www.hpcsociety.org/event-921306

[73] C. Chen, T. Clee, L. Huang, and Y. Yan, "Applying Big Data Analytics to Seismic Interpretation," presented at the Rice University Oil & Gas HPC Conference, 2016. [Online]. Available: http://sched.co/5sVA