Virtual laboratory for planetary materials: System service architecture overview

Cesar R.S. da Silva\textsuperscript{a,*}, Pedro R.C. da Silveira\textsuperscript{a}, Bijaya Karki\textsuperscript{b}, Renata M. Wentzcovitch\textsuperscript{a,c}, Paul A. Jensen\textsuperscript{a}, Evan F. Bollig\textsuperscript{d}, Marlon Pierce\textsuperscript{e}, Gordon Erlebacher\textsuperscript{d}, David A. Yuen\textsuperscript{a,f}

\textsuperscript{a} Minnesota Supercomputing Institute, University of Minnesota, Minneapolis, MN 55455, United States
\textsuperscript{b} Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803, United States
\textsuperscript{c} Department of Chem. Eng. & Mat. Sci., University of Minnesota, Minneapolis, MN 55455, United States
\textsuperscript{d} School of Computational Science, Florida State University, Tallahassee, FL 32306, United States
\textsuperscript{e} Community Grids Lab, Indiana University, Bloomington, IN 47404, United States
\textsuperscript{f} Department of Geology and Geophysics, University of Minnesota, Minneapolis, MN 55455, United States

Received 5 February 2007; received in revised form 26 April 2007; accepted 26 April 2007

Abstract

This paper brings an overall view of the service-oriented architecture (SOA) used in VLab, a system aimed to handle concurrent calculations of geo-materials participating in extensive workflows. We recap the algorithms of physical importance that underly the system requirements. The system architecture then emerges naturally. A usage view diagram is shown and thoroughly discussed. We also show how analysis tools are integrated in the SOA.

© 2007 Elsevier B.V. All rights reserved.

Keywords: First-principles molecular dynamics; Geo-materials computation; Distributed computing; Grids; Grid computing

1. Introduction

First principles methods based on the plane wave approach (Hohenberg and Kohn, 1964; Kohn and Sham, 1965) are the most common way to calculate theoretically the equations of state (EOS), \( P = P(V,T) \), or elastic constants, \( C_{ij} = C_{ij}(P,T) \) of earth forming materials. A particularly accurate method involves the use of damped variable cell shape molecular dynamics (VCS-MD) (Wentzcovitch, 1991; Wentzcovitch and Martins, 1991) with forces and stresses calculated from first principles (Nielsen and Martin, 1983, 1985). The results have been used to predict new phases (Tsuchiya et al., 2004), stability of phases (da Silva et al., 1999) and new temperature profile models (da Silva et al., 2000). Moreover, as experimental results for \( P(V,T) \) and \( C_{ij}(P,T) \) are scarce, most studies in physics of Earth and planetary interiors are directly or indirectly related to theoretical results.

In order to calculate a material’s EOS, the equilibrium configurations corresponding to a set of isotropic pressures must first be calculated using VCS-MD in damped mode. This is a pressure sampling phase, and deserves further considerations. This phase will be discussed in more detail in Section 2. If elastic constants are also required, a set of different Lagrangian strains must
be applied to each equilibrium configuration resulting from the first structural optimization step, and the resulting structure is relaxed again using fixed cell molecular dynamics (MD) in damped mode. This will produce a set of out of equilibrium internal stresses (Nielsen and Martin, 1983, 1985) that are used to compute the elastic constants at zero temperature. This approach does not take in consideration the atomic vibrations. In order to correct the EOS and $C_{ij}$ for zero point motion, or extend it to high temperatures, we must include the vibrational contribution. For this reason, the vibrational density of states (vDOS) must be calculated for every configuration (strained or not) resulting from structural optimizations described above (Anderson and Hama, 1999; Karki et al., 2000).

In the traditional approach, the calculation (MD, VCS-MD or vDOS) of any configuration is usually accomplished by one or more individual jobs. That, however, does not preclude concurrent execution of many jobs. The process consists in a systematic procedure, where the output of a given task is used to compose the input of the next one. In this process, the analysis of each output, elaboration of the next input, job submission and management of the enormous number of data files are performed manually. This is time consuming for the scientist and very prone to human errors. After the calculations are performed, the scientist must recollect a large amount of scattered information in properly formatted files to analyze the data. The analysis consists mostly in calculating the seismic velocities and related quantities to compare with seismographic available data.

When the kind of procedure described above is intended to be carried out automatically, it constitutes a workflow: “A Collection of actions, data, and its inter-relationships, that accomplish the automation of a work procedure, in which data is passed from action to action”. Here, action means any self-contained portion of work that produces modifications of data. The definition above applies to scientific workflow, which must not be confused with the concepts of workflow commonly used in business.

The VLab project aims to develop a service-oriented architecture (SOA) cyberinfrastructure designed to be a facilitator in performing this kind of workflows. The physical calculations are carried out by already available software packages. Among them, VLab has chosen the quantum ESPRESSO package (http://www.pwscf.org/) for the following reasons: ESPRESSO package is open source and available for free. It has a suitable, well integrated set of features, including variable cell shape molecular dynamics and calculation of vibrational density of states. Many participants in VLab project are also contributors to ESPRESSO package promoting synergy between the two projects.

In summary, performing a state of the art study in geomaterials poses several challenges to both the human and the computer system:

1. Preparing and submitting $10^2$–$10^3$ jobs.
2. Handling the huge workload imposed to the system.
4. Gathering relevant information scattered throughout hundreds or even thousands of output files from independent runs for further analysis.
5. Performing the analysis preferably without having to transfer data back to the user workstation.

Another important issue in performing this kind of calculation is performance. Plane wave first principles calculations involve a large number of Fast-Fourier transforms in three dimensions. The difficulties to achieve high performance and scalability throughout a large number of processors are well known. Fortunately, the workflow allows for a substantial number of jobs to be executed concurrently. Therefore, we can take advantage of a high throughput computing (HTC) model, in addition to the traditional high performance computing (HPC) approach. In our approach, each job will require only a moderate number of processors, allowing for better performance/processor ratio. A substantial number of jobs, however, are submitted for concurrent execution, relying substantially on the system throughput. Faster completion of the workflow, although expected, cannot be guaranteed. This approach, however, uses the system resources in a much more cost effective way than a purely HPC one.

The VLab project was proposed to fulfill three main goals: (1) provide tools to execute workflows as the one depicted above; (2) take advantage of the inherent workflow level parallelism to execute concurrently as many jobs as possible in a distributed computing environment; (3) provide a set of tools, interfaced to the user through a portal, to manage workflow execution and monitoring, data analysis and visualization in a collaborative way. The system is implemented as a service-oriented architecture (SOA) (Gannon et al., 2005). This paper brings an overall description of the current set of web services that implements VLab underlying SOA infrastructure. Although, a brief description of an early implementation is already given elsewhere (Nacar et al., 2006), this paper brings more accurate, and up to date information. The algorithm used to sample pressure parameter space was designed in a way to avoid coupling between pressure
points, enabling concurrent execution. This algorithm, which imposes several requirements on the system, as well as the complete $C_{ij}$ calculation workflow, is discussed in Section 2. Section 3 gives a block description of the service infrastructure. A closing summary is given in Section 4.

2. Workflows

Two programs from the quantum ESPRESSO package, PWscf ($pw.x$) for first principles electronic, structural and molecular dynamics calculations, and phonon ($ph.x$) for phonons calculations, are overwhelmingly responsible for the workload imposed on VLab by thermal and elastic properties calculations. MD, VCS-MD and eventually fixed configuration self-consistent field calculation are performed using $pw.x$. The program $ph.x$ calculates the dynamical matrix for a given atomic configuration at a given $q$-point. In most cases, input parameters are initial values of physical variables. Some, like cell vectors in MD calculations, pressure in VCS-MD, or cell vectors and $q$-point in phonons calculations, do not change during the execution cycle of the program, and are the ones frequently sampled in extensive studies.

In an extensive workflow such as high $P$, $T$, $C_{ij}$ calculations, most of work is due to the parameter space point sampling. Typical parameters to be sampled are pressure, Lagrangian strains and $q$-points. In our system, each parameter point corresponds to the execution of one task. As these parameter points are defined prior to the task starting, its execution cycles are decoupled from each other. This decoupling is what opens the opportunity for several tasks to be executed concurrently in a distributed environment.

Decoupled sampling of strains and $q$-points is straightforward. Sampling pressure points, however, is more complicated. This happens because starting from a configuration very different from the equilibrium is inefficient, as structural optimization will take much longer to converge. The usual approach to minimize this problem is to optimize structures in increasing order of pressures where the structural parameters obtained as output of $P_n$ are used as input for $P_{n+1}$. Unfortunately, this approach introduces an undesirable coupling between pressure points preventing them from running concurrently.

Fortunately, in most cases, what changes more substantially with pressure is the cell volume. This is the key to decouple pressure sampling. The calculation starts with a crude guess for the Vinet equation of state (EOS) ($Vinet et al., 1986$)

$$P = 3K_0 \frac{1 - f}{f^2} \exp[1.5(K' - 1)(1 - f)]$$

with $f = (V/V_0)^{1/3}$, and $V_0$ being the cell volume at zero pressure. Initial guesses for the cell volume from atomic radius, and $K' = 4$ are satisfactory. Bulk modulus can be guessed from any other material with similar composition and density. If this is not available, an estimate can be obtained by calculating internal stresses at two slightly different cell volumes, $V_a$ and $V_b$, around $V_0$. Variation in pressure, $\Delta P = P(V_b) - P(V_a)$, is calculated from the trace of the stress tensor at both volumes and $K_0$ is estimated as $K_0 = V_0(\Delta P/\Delta V)$, with $\Delta V = V_b - V_a$. This crude guess allows for a rough estimation of the cell volume at each pressure independently. Then a single self-consistent calculation, corresponding to just one MD step, is performed for each pressure. The virial pressures are calculated and used to obtain a better estimate for the EOS. This EOS, although still imprecise, is capable of producing configurations that are sufficiently near to equilibrium to allow for a fast convergence of the subsequent VCS-MD stage. We named this preliminary stage as fast refine (FR).

VLab workflows fall in one of six categories depending on extension:

(i) Simple execution of $pw.x$ or $ph.x$ codes. Those are usually preliminary or complementary calculations.

(ii) Pressure point sampling. Used mostly to determine a given materials equation of state (static EOS). This is usually done in two phases, each one consisting of a dozen or so of $pw.x$ runs. After the first phase, information must be collected from output of all $pw.x$ runs to compute the input for the second phase.

(iii) Determination of a given materials equation of state (finite temperature). Executed in three phases. The first two are identical to the case above. The third phase involves execution of the $ph.x$ (several runs for each $pw.x$ run from the previous phase). The $pw.x$ temporary binary outputs are used as inputs for $ph.x$.

(iv) Determination of elastic constants, $C_{ij}$ (static). Executed in three phases. The first two are identical to case (ii). The third phase executes several $pw.x$ runs for each run from the second phase. This case also involves collection of output files from all $pw.x$ runs of the previous phase.

(v) Determination of elastic constants (finite temperature). Besides the three phases involved in the previous case, it also involves several $ph.x$ runs for each $pw.x$ run in the second and third phases of case (iv).
(vi) Post-processing and visualization of results from (i) to (v). Although the categories (ii)–(v) generate a huge number of configurations, usually only a small number of those are selected for post-processing/visualization.

The general workflow can be written in terms of the following set of concrete operations:

(1) Using the portal, set up a general input containing a rough approximation to the EOS, and the pseudo-potential files.
(2) Call the appropriate service to transform that general input in several pw.x inputs. Each input is intended for a single MD step calculation with estimated cell volumes for the corresponding pressure based in the EOS from step 1.
(3) Run multiple distributed pw.x tasks using the input files from step 2 (Fast Refine).
(4) Gather pressure and cell volume data from the distributed pw.x standard outputs generated in the previous step.
(5) Fit the Vinet EOS.
(6) Automatically modify the general input with new values for EOS parameters from step 5.
(7) Call the appropriate service to transform that general input in several pw.x inputs for multi-step VCS-MD calculations using the new EOS. This will produce a new estimation for the initial cell volume.
(8) Run multiple distributed pw.x tasks using the input files from step 7. We call this step long refine.
(9) Execute again steps 4 and 5.
(10) Gather cell vector data from the respective pw.x standard outputs.
(11) Generate a user-defined set of Lagrangian strains for each cell collected in step 10, and generate corresponding pw.x inputs.
(12) Run multiple distributed pw.x tasks with fixed cell MD calculations from inputs generated in step 11.
(13) Collect the pw.x standard outputs from step 12, and extract: (a) the final stresses and (b) final positions.
(14) With a user-defined strains and stresses from step 13a, calculates a table of static elastic constants.
(15) With the final cells from steps 10 and 11, atomic positions from 13, and user-defined q-points, generate input files for the phonon jobs for every configuration resulting from steps 8 and 12. A phonon job comprises the execution of the following successive steps: (i) a pw.x run to perform a scf calculation of the final structure from steps 8 or 12; (ii) non-scf calculation if the q-point is not the gamma point; (iii) a ph.x run. The scf calculation is necessary to recreate the binary files used by the subsequent steps. All three calculations are to be carried in the same host.
(16) Run multiple distributed phonon jobs from input files generated in step 15.
(17) Collect all the dynamical matrices calculated by the ph.x runs in step 16 and calculate the constant force matrix for each configuration from steps 8 and 12. Then calculate the VDOS for those configurations.
(18) Using the VDOS from step 17, calculate the high temperature EOS, $C_{ij}$ and other thermal properties.

Fig. 1 shows a graphical representation of this workflow.

3. System description

The system is aimed to be strongly oriented towards the problem. In this case, a usage-oriented view (Laures, 2005) is more appropriate to describe its design. It is also more effective as a mean of elucidating the several design questions that appear during the solution devise phase of development. For these reasons, a usage-oriented view was primarily used for the development cycle. It will also be used throughout the rest of this paper to describe the architectural aspects of the VLab SOA.

The user interface is implemented in a portal. This includes the input preparation pages referred in step 1 of the workflow in the previous section. The portal is responsible for providing a consolidated single point interface to the main services. It also implements the user level abstraction of a single system, while providing access to the underlying resources. It aggregates the interfaces of project interactions, project executor, visualization services and analyses tools. It also allows for project management, input file preparation and uploading pseudo-potential files. The portal, and the technologies used there, are described in detail in a companion paper (Bollig et al., in press). All remaining steps, however, are executed by web services (Christensen et al., 2000). Steps 3, 7, 11 and 13 involve execution of tasks that take a long time to complete (long tasks or tasks for short), and are intermediated by steps that recollect data and execute short running tasks (stasks), like fittings, input transformations and calculation of strained input configurations. Extensive scattering of data, distribution of tasks and gathering of results are performed inside major blocks composed by steps 2–5, 6–8, 9–13(a), 13(b)–14. These blocks...
are successions of the following generic three steps sequence:

(i) Preparation of a set of inputs for *l*tasks, like *pw.x* or *ph.x* runs, and packing with appropriate companion files forming an *execution package*.

(ii) Distribution of the *execution packages* throughout the back-end computing nodes for execution.

(iii) Gathering results for analysis and set up of parameters to iterate steps (i)–(iii).

Moreover, in the overall process, input data comes primarily from the user and results must be, ultimately, sent back to the user.

Another key characteristic is that the data flows involved in our workflow are highly predictable and repetitive. This is in contrast with the LEAD project (Droegemeier et al., 2005), for example, that aims an implementation fully compliant with the Open Grid Service Architecture (OGSA) specification (Foster et al., 2004). In LEAD, data can come from a variety of sources.
in an unpredictable way, and must be sent to a multitude of destinations according to the particular application workflow that is collecting the data.

VLab, otherwise, is aimed to facilitate massive distributed computations. Workflow control and data transfers are responsible for an insignificant fraction of the overall workload. The communication scenario is much simpler. Most times a service is requested, the client sends input data to the server and receives back the service output data. Thanks to this reason, full OGSA compliance is not strictly necessary, and the requirement for a strong grid service layer can be relaxed. VLab presently implements only an essential subset of OGSA functionality. An increasing level of compliance with OGSA is slated for future versions. This choice allows for a much faster development cycle with limited resources, and the resulting system is still fully capable to fulfill its goals.

Massive number of **ltasks** are executed in blocks distributed throughout hosts that are interfaced to the rest of the system by the Compute WS web services. In many cases, execution cycles of **ltasks** can last for several days, or even weeks. Therefore, clients shall not be required to wait until the task is complete to get back the results. In this case, we would have to require the session to be persistent, which is a requirement quite difficult to fulfill. Consequently, it is far more convenient to run the **ltasks** unattended, under sole control of the host operating system. When executing a request, the Compute WS server closes the session as soon as the requested **ltask** is running, or otherwise submitted to a queuing system, in a submit and forget approach.

For sake of simplicity of design, services must be, as far as possible, purely transactional and stateless. This is particularly important for the Compute WS web services. Executions of **ltasks** are not transactions. Instead, starting or submitting **ltasks**, inquiring for status, or retrieving results, are typical Compute WS transaction. Therefore, after starting or submitting **ltasks**, they must send back the appropriate metadata to enable the client to locate the server, the files related to the task in the host file system, monitor execution status or further interact with the running task.

The piece of metadata used in VLab to track **ltasks** is called receipt. Every time a server is requested to run an **ltask**, it gives back a receipt to the client. When the client needs to retrieve any information, or request any further action (restarting the task, for example), it files a complaint with a code for that information or action requested, and send the receipt back to the server, that uses the information stored in the receipt to accomplish the request. In order to achieve persistency, the receipt is stored in a central database with other metadata. The complete description of VLab metadata can be found in reference (da Silveira et al., in press).

In most cases, **ltasks** are **pw.x**, **ph.x** or post-processing tools runs. These programs deal with two very distinctive kinds of files: (i) small input, standard output or intermediate text files. Due to their small to moderate sizes these files can be transferred multiple times between servers without generating significant network traffic. (ii) Huge binary files that contain wave functions, charge density, electrostatic potential, etc. In espresso, these files are intended to support resuming incomplete or interrupted calculations. They are also used as input for the post-processing tools and for the **ph.x**. Usually, they can be re-created from data present in the standard output, avoiding large file transfers. Moreover, being FORTRAN binaries, their format is system dependent. Since the system is heterogeneous, the transferred files would be useless in most cases.

For the specific case of charge density visualizations, however, re-creating the files is not the best strategy, since it is not reasonable to run PWscf to recreate those files in the visualization server. The solution for this case is to make sure to run the post-processing tool responsible for generating the visualization input data in the same server as the **pw.x** **ltask** has ran. Although post-processing runs are **ltasks**, they impose a much smaller workload than **pw.x** or **ph.x**, so they can always run in the same server as **pw.x** independently of its current load conditions. Moreover, post-processing utilities generates text files that, although large, are still much smaller than the binary files used as input. Consequently, they can be transferred without generating excessive network traffic or long delays.

### 3.1. Main requirements

Based on the preceding discussion, we can recap the main requirements:

(i) Services must be structured as a tree, with workflow control at the root, to match the generic three steps above.

(ii) Services must be, as far as possible, transactional and stateless, specially the Compute WS and the services that run the complementary **stasks**.

(iii) Services must return to the client, within the transaction, their results, or appropriate metadata in the case the requested service regards to an **ltask**.

**VLab**: The piece of metadata used in VLab is called receipt. Every time a server is requested to run an **ltask**, it gives back a receipt to the client. When the client needs to retrieve any information, or request any further action (restarting the task, for example), it files a complaint with a code for that information or action requested, and send the receipt back to the server, that uses the information stored in the receipt to accomplish the request. In order to achieve persistency, the receipt is stored in a central database with other metadata. The complete description of VLab metadata can be found in reference (da Silveira et al., in press).

In most cases, **ltasks** are **pw.x**, **ph.x** or post-processing tools runs. These programs deal with two very distinctive kinds of files: (i) small input, standard output or intermediate text files. Due to their small to moderate sizes these files can be transferred multiple times between servers without generating significant network traffic. (ii) Huge binary files that contain wave functions, charge density, electrostatic potential, etc. In espresso, these files are intended to support resuming incomplete or interrupted calculations. They are also used as input for the post-processing tools and for the **ph.x**. Usually, they can be re-created from data present in the standard output, avoiding large file transfers. Moreover, being FORTRAN binaries, their format is system dependent. Since the system is heterogeneous, the transferred files would be useless in most cases.

For the specific case of charge density visualizations, however, re-creating the files is not the best strategy, since it is not reasonable to run PWscf to recreate those files in the visualization server. The solution for this case is to make sure to run the post-processing tool responsible for generating the visualization input data in the same server as the **pw.x** **ltask** has ran. Although post-processing runs are **ltasks**, they impose a much smaller workload than **pw.x** or **ph.x**, so they can always run in the same server as **pw.x** independently of its current load conditions. Moreover, post-processing utilities generates text files that, although large, are still much smaller than the binary files used as input. Consequently, they can be transferred without generating excessive network traffic or long delays.

### 3.1. Main requirements

Based on the preceding discussion, we can recap the main requirements:

(i) Services must be structured as a tree, with workflow control at the root, to match the generic three steps above.

(ii) Services must be, as far as possible, transactional and stateless, specially the Compute WS and the services that run the complementary **stasks**.

(iii) Services must return to the client, within the transaction, their results, or appropriate metadata in the case the requested service regards to an **ltask**.
3.2. Overall structure

The overall structure is shown in Fig. 2. The main computing services are in the bottom box. They comprise Compute WS, Volumetric Data Visualizer and ElasViz Server.

Compute WS web service is responsible for running pw.x, ph.x and the post-processing tools from ESPRESSO package. It fulfills requests to start tasks in the underlying operating system of the server, check status, interrupt tasks and exchange files with the client. When starting pw.x or ph.x, Compute WS receives as input a compressed file called execution package. It contains a directory with one or more pw.x or ph.x input files, the pseudo-potential files requested in the input file and an empty sub-directory called tmp, which is the necessary environment to run the task. Once received, the execution package is decompressed, and the task is queued or started inside the decompressed directory. Output files, like the standard pw.x output, or the dynamical matrix produced by the ph.x code will also be stored in that directory. Submitting an execution package for execution is a single transaction that is acknowledged with a receipt. Once the implied task is started or queued and the receipt sent back to the client, the transaction is complete. At this point, Compute WS closes the session and disregards any information about it. Any further transaction regarding that task must be accompanied by the receipt. Possible further actions are: requests for status update, task interruption, task restart, starting several post-processing utilities, download input or output files, upload replacements for files in the execution package and upload input files for the post-processing utilities.

Compute WS must also have the ability to make results from the post-processing tool directly available to the volumetric data visualizer server to avoid unnecessary network traffic. This is easily done within the receipt protocol. When the visualization post-processing
starts, it sends back a receipt containing, among other things, the URL of the visualization data file that will be generated. When the completion of the post-processing \textit{ltask} is detected, the visualization server is notified and downloads the data file from the compute server.

The topmost service is the project executor. In \textit{VLab}, we use the concept of \textit{projects} as user level abstraction for the workflow. Project executor duty is to control the execution of the workflow at higher level. It calls the auxiliary services that execute several \textit{stasks}, sends \textit{ltasks} for execution, gathers results to be fed in the auxiliary services, processes output files from one phase producing input files for the next one, controls the sequence of phases and takes automated decisions to controlling the overall workflow execution.

On entry, the project executor receives the project descriptor and a compressed directory called \textit{basic package}. The project descriptor contains general information about the project. The \textit{basic package} contains the general input file for the whole project, and a subdirectory that is a template for the \textit{execution packages}. From the \textit{basic package} it creates a directory, called \textit{main package}, aimed to keep copies of the \textit{execution packages} with up to date output files generated by the corresponding \textit{ltask}. Consistency with the files in the backend is guaranteed at the end of the \textit{ltask} execution and at every user interaction.

The execution of the entire project and the phases that compose it are also considered \textit{ltasks}, and therefore generate receipts. The receipt is a recursive hierarchical data structure. Project receipts contain phase receipts that, in their turn, contain task receipts. Project executor stores project receipt in a database, within the project descriptor, and takes care of maintaining these metadata up to date.

Compute WS is, by itself, unable to detect changes in status of its tasks. Therefore, it cannot notify the project executor upon those changes. To circumvent this problem, project executor frequently polls Compute WS about status of tasks to detect changes of state. In order to do that, it first parses the project receipt for outstanding \textit{ltasks}, i.e. \textit{ltasks} that have been sent to task executor but not yet returned final results. It then uses the attached \textit{ltask} receipts to locate the computing host, job (or process) ID, working directory, etc., ...

The task dispatcher is responsible for electing back-end nodes and dispatching \textit{ltasks} for execution. It gathers information on current load conditions in each compute server prior to dispatch for execution heavy \textit{ltasks} \textit{(pw.x} and \textit{ph.x}). After dispatching an \textit{ltask} it gets the \textit{receipt} and forwards it back to the project executor.

Task interaction is currently implemented as a library shared by both project executor and project interaction. Its function is to retrieve data and task status from the compute nodes and keep consistency of the main package. It also supports the features necessary for user intervention in running projects, and provides interfaces for accessing the semaphores used to control shared access to task files.

There are also several auxiliary services. Their function is to execute tasks that are usually intermediate between major phases in the workflow. They are characterized by fast execution cycles and return results almost instantly, within a single transaction. Consequently, they do not need to generate any persistent metadata. The Eulerian and Lagrangian strain generators are auxiliary services responsible for generating the several input files corresponding to parameter sampling in pressure and cell vectors. There are also services for fitting the Vinet EOS, preparing \textit{ph.x} input files and post-processing of \textit{ph.x} output files (labeled as high T post-processing in Fig. 2).

Project interaction is a web service that supports the project monitoring user level functionality, the monitoring portlet, in the portal. This module allows for user interactions with projects and all tasks that compose it. In \textit{VLab}, collaboration is established by means of shared access to projects, including file handling and tasks management. Users can intervene in a project by modifying input files, and canceling or restarting tasks. Project interaction is responsible for supporting this kind of shared access by precluding race conditions that could arise from it, and potentially compromise data integrity.

As an example, Fig. 3 is a snapshot of the monitoring portlet showing the status of a small project named “\textit{EOS-MgO}”. At that moment, the project was in the long refine phase. This is the calculation of the equation of state of cubic MgO using a two atom cell. Equilibrium cell volumes were calculated for a set of 14 pressures. Forces and stresses were obtained from realistic electronic structure calculations. The tasks are distributed throughout servers belonging to two different domains, University of Minnesota (umn.edu) and Indiana University (indiana.edu).

Integrating analysis tools, with specialized visualization features is a primary goal of \textit{VLab}. There are currently two kinds of visualization implemented in \textit{VLab}: (a) volumetric data visualization, used for analyzing charge density distributions and (b) elasticity visualization.

The volumetric (3D) data visualizer is used in \textit{VLab} to analyze spacial distribution of charge density of selected atomic configurations. For this reason, it is also referred as \textit{charge density visualizer}. Its use is usually associ-
ated with verification of correctness of individual first principle calculations. Therefore, we elected to hook its interface through the monitoring area of the VLab portal that interfaces with the Project Interaction web service.

This visualizer takes as input charge density data point files produced by post-processing jobs and renders it as isosurfaces. Additionally, it can render cutting planes, allowing for the analysis of a two-dimensional projection of data along any plane chosen by the user. It also features three-dimensional interactivity, allowing for image rotation and zooming. The images are high quality, print-ready, allowing for production of publication quality figures.

The visualizer is composed of two parts: a high performance visualization server, on which the datasets are stored and images are generated; a client that runs in the user workstation.

The visualization service relies heavily on the Visualization Toolkit (VTK) (http://www.vtk.org/doc/release/5.0/html/) to perform visual modelling, geometry and offscreen rendering. It operates as a web service, using SOAP messaging to communicate between an interactive client and a high performance visualization server, on which the datasets are stored. Serialization of the web service is accomplished with the help of the gSOAP (http://www.cs.fsu.edu/engelen/soapdoc2.html) toolkit.

The client is responsible for user interactivity and display. It is written entirely in Java and deployed with Java Web Start, allowing it to be run on any platform with automatic installation. Three-dimensional interaction with the image model is simulated on the client. A wireframe bounding box showing the orientation of the dataset is rotated or scaled by the user until the desired positioning is obtained (the same approach is also used to position the cutting plane). The client then sends a SOAP request containing the new orientation to the visualization server, where a new rendering of the data is generated offscreen. A bitmap image of the data in the new orientation is captured and encapsulated for transmission back to the client, where the fresh image provides the user with the desired new view of the data.

Because three-dimensional interactivity is only simulated, a small latency while new images are transferred can be observed, depending on the network conditions. Substantial effort is being devoted to develop new encapsulation methods to optimize the client/server communication and enable the application to provide closer to real-time responses.
The elasticity visualization functionality, by its turn, is the most important analysis tool from the geoscientist’s perspective. It is the \textit{VLab} component that bridges the newly calculated elastic constants to geophysical data, including experimentally determined elastic properties. Due to its importance, a more detailed description will be given.

\textit{ElasVis} has been recently developed to be an efficient visualization system to facilitate understanding of various aspects of the mineral elasticity datasets produced by an increasing number of both theoretical calculations and experiments (Karki and Chennamsetty, 2004; Ananthuni et al., 2006). It allows us to visualize and analyze the elastic constant tensors and the additional data (for wave-velocities and anisotropy), which are generated on the fly.

While the original implementation (Karki and Chennamsetty, 2004) was done in C with support of OpenGL (Schroeder et al., 2004) and GLUI, the current implementation is done in JAVA with JOGL (https://jogl.dev.java.net/) to support the remote extension and interaction (Ananthuni et al., 2006).

The \textit{ElasVis} architecture consists of several components including data management, visualization and interactivity modules, arranged in a client–server architecture. The server side components act as a repository for web application files, Java standard application classes, external libraries, Java Network Launching Protocol (JNLP) (http://java.sun.com/products/javawebstart/reference/api/index.html) files deployed with each client request and other scripts, and elasticity data, that are kept in a dedicated database. Application classes and other dependencies are deployed in the user machine using Java Web Start technology. For each client request, a JNLP file will be saved on a web server. Based on the JNLP descriptor, application classes and external libraries are signed and dynamically deployed.

The \textit{ElasVis} data management component encompasses the mechanisms, which import data from external sources into visualization system and manage the data internally with generation of additional data. The original input data are the calculated or measured values of elastic moduli, $C_{ij}$ with $i, j = 1, 2, \ldots, 6$.

The $C_{ij}$ values are used to generate additional data such as velocities and anisotropy factors, which are needed for the understanding of wave propagation in an anisotropic material. Elastic wave velocities are obtained as a function of propagation direction by solving the Christoffel equation (Musgrave, 1972):

\begin{equation}
|C_{αβγσ}n_αn_γ - ρV^2δ_{αγ}| = 0
\end{equation}

where $n$ is the propagation direction, $ρ$ the density, $V$ the velocity and $δ$ is the Kronecker delta function. The calculated eigenvalues and vectors define three unique waves for each propagation direction: (a) one longitudinal ($P$) wave, (b) two shear ($S1$ and $S2$) waves. Thus, generated velocity data (eigenvalues) are then used to generate different types of anisotropy data such as the azimuthal, polarization and transverse anisotropy factors.

The visualization module is a set of software components that perform rendering of different sets of data. It deals with manipulation of the geometric primitives (points, lines or polygons) required for rendering and determines where and when objects are displayed. It supports animation, selective, global and multiple viewpoints. \textit{ElasVis} adopts the parallel coordinates and star plot techniques to visualize the $C_{ij}$ data. In both techniques, the number of axes is equal to the number of independent elastic constants, which is determined by the symmetry of the material under consideration; for example, an orthorhombic crystal will have nine axes. The computed velocity-direction data are graphically displayed using the polygon-based surface rendering technique. Three velocity surfaces are represented by different colors: green for longitudinal wave, red for shear wave 1, and blue for shear wave 2. Thus, the shape and brightness of the closed 3D surface together represent the velocity-direction distribution. Other data are displayed using line or scatter plots.

Finally, the user interface module deals with the exploration of the data in a flexible and interactive mode. The list of features include: detachable toolbars, personalized color and thickness options for selected datasets, single and multiple dataset modes, zooming options, among others. It also allows to visualize the velocity distribution waves in 3D mode or 2D mode (i.e., $xy$, $yz$ and $xz$ planes).

\textit{Elasviz} was not originally intended for \textit{VLab}. It was, however, easily integrated, since it relies on a client–server architecture, a database to store its data and on Java Web Start to deploy the user side client.

When the $C_{ij}$ workflow ends, all the basic data necessary to calculate the elastic constants are already in place. At this point, the user needs to fill out some additional information, like the crystal type and the sum of atomic masses inside the calculation cell. When the user actually requests the visualization, the portal sends a request to project executor to calculate the appropriate set of elastic constant according to the crystal symmetry, and inserts the constants into the database. It first creates a new entry in the table \textit{tblmanualdatasets} (Fig. 4), and sets the field \textit{projectname}. Here, \textit{manualdatasetid} is the primary key to the table \textit{Tblmanualdatasetval-}
ues, where the newly calculated constants are actually inserted. Each line of Tblmanualdatasetvalues holds the elastic constants corresponding to one pressure.

After the data is inserted into the database, project executor notifies the portal that starts the client side application using JNLP protocol. The projectname, manualdatasetid and the necessary information to make a connection to a specific database are inserted in the corresponding JNLP file. The client side application then selects the data inserted in the previous step, and renders the images.

4. Summary

We have recapped the conceivably most extensive workflow in geo-materials elasticity research, which is prototypical for a number of other applications, and established the main requirements for a SOA infrastructure capable of supporting it. We have shown, in a usage-oriented view, the SOA necessary to accomplish those requirements, the services specifications, and explained how to integrate remote data analysis tools into the SOA.

Acknowledgments

Research was supported by NSF grant ITR-0426757 (VLab). Minnesota Supercomputing Institute provided the main computational facilities.

References


Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., 2000. Web services description language (WSDL), Version 1.1 (see also http://www.w3.org/TR/wsdl).


constraints from the elasticity of silicate perovskite. Phys. Earth Planet. Interiors 118, 103.

da Silveira, P.R.C., da Silva, C.R.S., Wentzcovitch, R.M. Metadata management for distributed first principles calculations in VLab—a collaborative grid/portal system for geo-materials computation, in press.


Jogl home page: https://jogl.dev.java.net/.


VTK online manual at http://www.vtk.org/doc/release/5.0/html/.