Pacific Northwest Climate Analysis: A Parallelized, Provenance-Aware, and Extensible Framework for Climate Analysis and Beyond

Brett Yasutake
in partnership with Niko Simonson

University of Washington
Introduction

Analyzing Climate Data in 2013

Climate models consist of a vast array of real world physical measurements. These measurements, such as precipitation and temperature, are simulated over 100+ years. As climate scientists have become smarter over the past half century, climate models have become more sophisticated not only in their complexity, but also in the resolution at which we can model these physical phenomena. Our Pacific Northwest Climate Analysis (PNCA) framework facilitates climate researchers to gain insights from these increasingly complex climate models.

Over time these models have incorporated more sophisticated and realistic methods to model the oceans, air chemistry, and terrain vegetation, to name a few (Figure 1). Even as recently as the early 1990s, a 500 km by 500 km square on the earth contained one set of measurements. Compared with the most recent model from 2007, the same physical grid cell on the earth contains almost 25 sets of measurements.

Figure 1. Climate model advancements over time
These simulated physical phenomena are captured and stored in NetCDF formatted files, which are highly efficient, array based files that have become the standard file format in the climate science research community. Tools and software libraries have been developed to read, manipulate, and analyze the climate model data stored on NetCDF files, but many challenges are still encountered by climate researchers while attempting to perform their analysis on such files. Data analysis tools have not kept pace with our ability to capture and store data [Gray et al.]. Existing tools have been notoriously difficult to install, difficult to use, and provide no data provenance support. In addition, such tools offer only limited options and can only perform simple operations. Many times the use of such tools are abandoned in favor of implementing custom, one-off programs to perform the desired analyses. These custom programs are written for a single analysis and offer little to no reusability.

Climate researchers must often spend a lot of time and effort, not towards implementing their domain specific analyses, but towards periphery, non-climate science aspects of their analyses. These activities include file handling, file manipulation, writing output to results files, and collecting data provenance. The sheer size of these climate models (250MB per day, 36TB for a 100 year simulation) poses immediate practical issues with not only the transfer and storage, but more importantly with the time it takes to analyze the data due to hardware limitations. For this reason, it is desired to perform the analyses in a parallel fashion in order to produce results in a practical and reasonable amount of time. We would argue the effort put towards these periphery, non-climate science tasks should be of concern to computer scientists, and not climate researchers. We aim to fill the gap in usability, programmability, and functionality between existing toolsets and the climate science community’s need to easily implement complex data analysis algorithms on NetCDF climate models, while allowing climate researchers to take full advantage of the benefits of parallel processing.
Why MASS?

The mobile agent and spatial simulation (MASS) library has been an ongoing research project at the University of Washington. The MASS library addresses the semantic gap between analyzing algorithms and their actual implementation [Emau et al.]. The MASS library abstracts away many of the complex and technical details of parallel programming, such as inter-process communication and shared resource management. For the programmer, one simply defines how many computing nodes and resources are needed, and MASS manages the intricate parallel programming details under-the-hood.

Basing our framework on MASS allows us to further close the semantic gap between data analysis algorithms and their actual implementation, providing simple interface contracts and a simplified programming paradigm. This enables climate researchers to focus on implementing their domain specific algorithms, while taking full advantage of the power of parallel processing.

The MASS project has been, and will be, an ongoing and active area of research. This provided extra incentive to base our framework around MASS, ensuring that it will continue to be maintained and extended upon by future researchers as new advancements in MASS continue to be developed, such as the recent implementation of a dynamic load balancer.

Pacific Northwest Climate Analysis Project Goals

Our project contains several goals, not only immediate and short term, but also long term stretch goals. The first and foremost goal was to establish a framework for Pacific Northwest Climate Analysis, enabling climate researchers to code their own analytics modules and enable them to focus primarily on their data analysis algorithms of interest and at the same time allow them to transparently utilize the power of parallel computing. The initial release of our framework will provide immediate practical use by our stakeholders from the Climate Science and Physical Sciences departments.

Another immediate goal was to demonstrate a new, real world application of MASS in the
discipline of climate science to perform climate data analysis. Previous applications of MASS were simulation based, and our framework aims to be the first application of data analysis.

A short term goal is continued maintenance and extension of the framework, not only by MASS researchers but by climate scientists. Our design choices were heavily influenced by this goal, with the hopes that high usability and programmability lead to adoption by the climate science community, in not only an end-user perspective but also in developing new analytics modules for complex climate data analysis.

Our long term goal was to establish a framework, from which any scientific data analysis can be performed. NetCDF data is not only used by the climate science community, but also in astronomical and geophysical data.

**Methods**

*Our Project Management Approach*

The Pacific Northwest Climate Analysis framework was developed according to a six month project plan. This was a team project, in which my focus was on data provenance and usability and my partner’s focus was on parallel programming and algorithm development. The development lifecycle was organized around iterative two week sprint cycles, in which our focus and efforts for each sprint were driven by addressing our highest risks.

Each planned two week sprint involved initial risk analysis, evaluating any alternatives, development of deliverables and software, and testing. Review checkpoints were established to coincide with the end of each two week sprint cycle. Deliverables and targeted functionality were defined for each review checkpoint, planned from the beginning of the project. This iterative, agile approach allowed us the flexibility to adapt to changing requirements and unknown constraints that came up during the software development lifecycle.
Three major prototype versions were established: alpha, beta, and release candidates. The alpha development phase ran from late August of 2013 and concluded mid-October of 2013. The beta development phase ensued for another month and a half, and we arrived at our feature complete release candidate at the beginning of December of 2013.

Risk analysis was performed at the beginning of each sprint cycle to address and mitigate our highest potential risks that could potentially block us from achieving our project goals. Some of our highest risks included a NetCDF I/O bottleneck, components taking too long to build, insufficient architecture analysis, and poor programmability for climate researchers.

These risks were mitigated by the use of evolutionary prototyping, a scope advisory in which we declared features as potentially moving out of scope, peer reviews, seeking architectural design expertise from several of our faculty advisors, and gathering usability feedback from a Physical Sciences department faculty member.

**Ensuring Overall Software Quality**

Our quality management approach began with defining the quality attributes [Garvin, Perry, ISO 9126] most important for our framework to reach our goals. These goals were cross-referenced in a dependency matrix to highlight the trade-offs between quality attributes, positively or negatively.

To manage software quality, the practice of pair programming was utilized throughout the entire development of our framework. One person would be the “driver” and do the actual writing of the code. The other person would act as an “guide” or “navigator”. This proved to be very advantageous, in that the person doing the coding could focus on the task at hand such as the algorithm being implemented, syntax, and semantics. The other person would review all of the code being written and provide a higher level perspective, providing guidance on the overall software architectural design and software quality.

Switching roles often was another key aspect to our implementation of pair programming, which
allowed us to take advantage of complementary skills between the two of us. In the end, the development was roughly equally shared between both team members, and as a natural benefit, both of us ended up as intimately familiar with the code as the other, leaving no pockets unfamiliar code to the other.

Regular stakeholder meeting were just as important as development. Continual requirements gathering and refinement were of the utmost importance to achieve our goal of user adoption by usability and programmability by climate researchers. Weekly to bi-weekly meetings with our climate science stakeholders were conducted to obtain guidance on climate analysis algorithms and further refine our requirements to ensure a usable framework in the end was achieved.

We also participated in weekly meetings with the ongoing MASS research team to get support and continually review our approach to climate data analysis in the MASS programming paradigm.

Bi-weekly meetings were conducted with a computer science and software engineering faculty advisor to obtain architectural and data provenance feedback. Our extensive use of adapters to achieve high modularity and extensibility were heavily influenced by these faculty advisor meetings.

**Resources and Tools**

Many resources were utilized throughout the execution of this project. The Distributed Systems Laboratory at the University of Washington Bothell was the primary workspace of the MASS research team in which we participated in throughout the duration of this project. The NetBeans IDE was the sole development environment for our Java based framework.

For performance testing and storage of our climate model data, a quad-core, high performance, Linux based workstation was used. This system, known as “Hercules”, resides in the aforementioned Distributed Systems Laboratory.

The method of source control used was Git, combined with a Google Code repository to manage, store, and allow easy review and access to all versions of our code. Google documents, spreadsheets, and
presentations were used for all project management deliverables and artifacts produced throughout the planning and execution of our project. Google calendar, in combination with Trello, was used for schedule and task management.

**Weather Research & Forecasting (WRF) Model**

Global climate models are simulated and released once every five to seven years. Global climate models are then downsampled to produce smaller, more localized regional climate models. These regional climate models were the target data of our framework’s analysis, in particular the Weather Research & Forecasting (WRF) v3.1.1 model.

Climate model data are organized and stored within NetCDF formatted files. Each file represents a single time step in the simulation. The size of one time step in our climate model represents one quarter of a day, either the time step from midnight to 6am, 6am to noon, noon to 6pm, or from 6pm to midnight.

Each individual file for its time step includes simulated measurements for over 150 variables such as temperature, precipitation, and surface pressure. Each individual file contains these measurements over a region of the Pacific Northwest, partitioned into a 123x162 grid.

Each NetCDF file contains metadata identifying its climate model of origin, simulation date and time, geographical coordinates, and a host of other projection and model information. As part of data provenance collection, our framework automatically captures and writes all metadata from the source file and creates a data provenance metadata file. This data provenance file not only includes all source file metadata, but also any experimental parameters from our framework such as number of processes, threads, execution time, and any other user-defined data provenance variables. The metadata is also transferred to output files, enabling them to be read by any visualization tool, or even to be fed back into the framework for further analysis.
**System Architecture**

*Adapter Centric Design*

Our architectural design is centered around the use of adapters (*Figure 2*). This enables high modularity and extensibility because it minimizes the dependencies between all components of our framework. For instance, if the framework to be extended to enable more types of source data such as from a database or spreadsheet, only the input adapter needs to be modified. Similarly, if someone wishes to write the results files to another format other than NetCDF, only the output adapter needs to be updated.

*Figure 2. Final architecture dataflow diagram*
This adapter centric design allowed us to isolate the analytics module as much as possible. A climate researcher would implement their domain specific analyses by writing a new analytics module. The adapters provide a toolset for the analytics module writer, simplifying file handling, communication with the user interface, and data provenance collection. The analytics module writer simply needs to provide a few required functions and place the analytics module in the correct package, and the new analytics module is automatically integrated into the user interface.

**Extracting Climate Model Data for Analysis**

The input adapter provides simplified file handling to the analytics module writer for enhanced usability. The input adapter provides various read functions to extract source data from the NetCDF climate model for the various data types a NetCDF file can contain.

The climate researcher only needs to provide a source file name, the variable name, and its dimensions in order to receive an array filled with its values, ready to be manipulated. Several helper functions are also included in the input adapter. A getDay function, for example, returns the time step that a file represents.

**User Authored Analytics Modules**

The analytics modules are coded by climate researchers and can be thought of as “plug-ins” to the PNCA framework. Any number of user defined functions are written to carry out the analysis (*Figure 3*). The user defined functions are performed in parallel and are where the climate researcher utilizes our set of adapters to extract data from climate models, manipulate and share data, collect provenance, and write results to output NetCDF files. The climate researcher can focus their efforts on the data analysis algorithm with minimal focus on file handling, provenance collection, user interface communication, and parallel processing.
Figure 3. Basic framework for developing an analytics module

Writing Analysis Results to Output

When data analysis is completed, the output adapter facilitates easy writing of the results to NetCDF files. Similar to the input adapter, the climate researcher only needs to provide the array filled with the results data, a variable name, source, and target file names. The output adapter automatically writes the data to a NetCDF output file, all the while automatically capturing and transferring the data provenance and metadata to the results file.

Customizable and Automatic Data Provenance Capture

The provenance adapter performs several functions. A provenance adapter with no constructor parameters can be instantiated for exception logging. All exceptions are passed to the provenance adapter, which identifies the function and computing node in which it occurs, writing to a unique exception log for the particular experimental run.

Instantiating the provenance adapter with a source file and target file name automatically
produces a data provenance file in NetCDF format. This data provenance file contains all source model information as well as user customized data provenance information, such as all MASS parameters and the date and time of the experimental run. These are written to global variables within the data provenance NetCDF file. NetCDF files were chosen to store data provenance in a manner to support easy query and retrieval [Davis et al.] while also balancing the provenance capture with overhead and storage constraints [Reilly et al.].

**Automatic User Interface Integration**

The execution adapter generalizes user authored analytics module execution through the use of Java reflection. There are three functions that are called by the execution adapter: setFiles, initialize, and runAnalysis (*Figure 3*). As long as the analytics module writer includes these required functions and places the analytics module in the analytics package, integration with and execution via the user interface is automatic and transparent to the analytics module writer.

The analytics module writer can provide customized status messages to the user interface with a call to the status adapter’s reportMessage function. The status adapter employs a model view viewModel design pattern in which it holds the latest status message, allowing the user interface to query the adapter at regular intervals for new or updated statuses to display to the end user.

**User Interface Approach**

The user interface contains a single frame from which all user actions are performed (*Figure 4*). A file selector is used to capture a list of input files selected by the user. A dynamic algorithm selection drop down list is used, populating it with any analytics modules found in the analytics package. Upon execution by clicking the Start Analysis button, a new thread is instantiated to run the execution adapter’s runAnalysis function, passing it the selected input files and analytics module.
A third party visualization tool, named Panoply, was integrated into the framework for a couple of reasons. The first reason being that it is a widely available and free tool to inspect and visualize NetCDF formatted climate data. The second reason is that the climate science department was already using the tool and familiar with its use and abilities.

![Figure 4. User interface design](image)

**Results**

*Our Findings*

Our final release candidate was the result of five major prototypes, 44 commits, and over 6,000 source lines of code. Out of that 6,000 source lines of code, the average analytics module contains only 400 source lines of code. Comparable Fortran code that performs a similar analysis contains over 1600 source lines of code.

The recommended module size for minimum defect density if 200-750 source lines of code [Larid et al.]. This is very important with respect to our goal of usability and programmability by climate researchers because the size and effort to write an average analytics module (to perform quite complex analyses) in our framework ensures minimal defect density and debugging.
We chose to install and perform the same analysis as our moisture flux analytics module using an existing tool. Through this process, we gained first hand experience with the difficulty in using an existing tool and its limitations. In the end, we ran a moisture flux analysis on one month of NetCDF climate data. Our framework completed the analyses in 13 seconds, compared to the existing tool taking 2 minutes and 36 seconds to perform the same analysis. From a usability perspective, it also produced twice as many output files (without experimental run data provenance) because it was limited to only writing one variable to one output file, whereas our framework was able to combine both output variables into a single results file.

Hardware memory limitations were encountered throughout the project. To mitigate this issue, we enabled MASS to perform multiple calculations. The data analyses are performed in “chunks”, to avoid an out of memory heap issue from loading all the source data at once.

Usability feedback is currently being sought from a Physical Sciences professor. An initial training session was conducted, in which we were able to create a simple analytics module to demonstrate the programmability of our framework in under a half hour. We hope to evaluate these results based on perceived benefits, speed-up, and programmability while ensuring we evaluate these results with respect to the user’s programming background [Teijeiro et al.]. The architectural design and programmability enabled us to relate the programming paradigm as a “plug-in” feature, which greatly helped the user relate to how the framework will use newly created analytics modules.

Pacific Northwest Climate Analysis’ Impact and Future Potential

The most immediate impact of Pacific Northwest Climate Analysis is the immediate practical use of the ten analytics modules already implemented, including moisture flux, wind gradient, and extreme indices (Figure 5). The extreme indices modules in particular, are a set of very common, standard calculations used to quickly profile and gain insights from a set of climate data. Simple precipitation index
and counting the number of icing days are some examples already implemented in our framework.

Along with the already implemented modules, the framework can be put to immediate practical use by the continued development of more analytics modules and adoption by the climate science community.

![Select Analytics Module](image)

**Figure 5. Climate analysis algorithms implemented (analytics modules)**

NetCDF formatted files are used not only in climate science but also in astronomical and geophysical data. Our architectural design allows easy extension to utilize other data sources for not only input but also output. With these two reasons in mind, it is our hope that the Pacific Northwest Climate Analysis framework will spread its breadth to other disciplines and types of data analysis.

Mobile agents are a major feature and benefit of MASS, but we have not utilized mobile agents in our framework. Now that an established starting base for data analyses within MASS has been created, mobile agents can be employed over the computing space to perform complex pattern detection. Mobile agents can be deployed to search for storms, and correlate the data with atmospheric conditions elsewhere in the climate model, for example. As mentioned earlier, MASS is an ongoing and active research project. Our immediate next steps are to not only perform proper hand-off to the MASS research team for continued maintenance, but to enable future researchers to extend upon the framework.
References


Noirhomme-Fraiture, M. Processing of large data sets: Evolution, opportunities and challenges.


Appendices

Appendix I: Faculty Advisors
- Dr. Munehiro Fukuda, *Computer Science and Software Engineering*
  - Domain expertise in parallel and distributed computing and architectural design
- Dr. Erik Salathe, *Climate Science and Policy*
  - Domain expertise in climate science, climate model analysis, and NetCDF
- Dr. Hazeline Asuncion, *Computer Science and Software Engineering*
  - Domain expertise in data provenance and architectural design

Appendix II: Condensed Project Plan (2013)
- Week of 7/21 - development of QA documents, architectural design
- Week of 8/4 - first prototype and unit tests; simple parallel computation utilizing the MASS library
- Week of 8/18 - second prototype; formal status review; integration and initial validation tests are available; read NetCDF in a parallelized fashion
- Week of 9/1 - third prototype, aim for alpha (feature complete) candidate one; climate analysis algorithm implementation complete
- Week of 9/15 - fourth prototype, alpha candidate two; moisture flux divergence output data set is achieved; user interface displays task status
- Week of 9/29 - fifth prototype, beta (can be used/demoed to stakeholders) candidate one, acceptance testing; user interface displays results (text, summary statistics)
- Week of 10/13 - sixth prototype, beta candidate two; user interface displays graphical results (charts, graphs, maps)
- Week of 10/27 - seventh prototype, beta candidate three; user interface allows user defined algorithms to be created and executed
- Week of 11/10 - eighth prototype, code complete release candidate one; user acceptance testing
- Week of 11/24 - ninth prototype, release candidate two to address acceptance test issues; user acceptance testing
- Week of 12/8 - complete and present project; code complete system is achieved

Appendix III: Project Activity Grid

<table>
<thead>
<tr>
<th>General Activity</th>
<th>Proposed Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1) Project Management/Communications (PMC)</strong></td>
<td><strong>Project Plan</strong></td>
</tr>
<tr>
<td>- Complete Project Proposal</td>
<td></td>
</tr>
<tr>
<td>- Expand schedule by integrating Proposal’s review checkpoints with activity grid</td>
<td></td>
</tr>
<tr>
<td>- Include formatting such that the schedule’s development history is</td>
<td></td>
</tr>
</tbody>
</table>
preserved, but the schedule itself can be modified to take into account changing conditions

|● Elicit requirements through interviews with stakeholders: Professors Asuncion, Fukuda, and Salathe
● Develop preferred functional requirement format and cataloguing scheme
● Design specification
● Derive further requirements from Architectural Design Document |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional Requirements Specification</td>
</tr>
</tbody>
</table>

|● Brainstorm risks
● Identify further risks through stakeholder interviews
● Identify further risks through relevant literature review |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Risk Analysis</td>
</tr>
</tbody>
</table>

|● Develop communication approach through formal knowledge (i.e. information from past courses)
● Maintain communication via email and Trello to establish joint approach |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Stakeholder Communication Management</td>
</tr>
</tbody>
</table>

2) Technical Approach (TA)

|● Develop architecture diagram using a hybrid architectural approach:
  ○ data flow
  ○ model-view-controller (especially for UI development)
  ○ independent components (communicating processes)
  ○ layered systems
● Decide on and implement evaluation method
  ○ Either way, the evaluation method must be abbreviated due to time constraints
  ○ Fundamental question is: scenario-based or model-based? (Answer: scenario-based)
  ○ Incorporate best practices for distributed and HPC evaluation
● Design components, flow of data, and flow of control in a document using the Software Architectural Visual Notation (SAVN)
● From the Architectural Diagram, focus on data flow
● Determine data distribution and collection scheme, relationship with algorithm development |
|---|
|Architecture Diagrams
  ● SAVN Diagram
  ● Data Flow Diagram
  ● Package Dependency Diagram
  ● Conceptual Network Project Backbone Diagram|

|● Identify resources required from Functional Requirements and Risk Analysis
● Develop acquisition approach; track results |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Computing Resource Document</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>● Develop algorithms from consultation with Professor Salatthe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm Specifications</td>
</tr>
<tr>
<td>● Familiarization of NetCDF from literature review</td>
</tr>
<tr>
<td>● Study provided data sets</td>
</tr>
<tr>
<td>● Identify non-functional requirements from requirements elicitation</td>
</tr>
<tr>
<td>● devise means to measure and track</td>
</tr>
<tr>
<td>● Sketch out UI based on stakeholder interviews and functional requirements</td>
</tr>
<tr>
<td>● Design interfaces based on architectural documents</td>
</tr>
</tbody>
</table>

### 3) Testing Strategy/Results (TSR)

|● Develop quality management plan from Project Proposal Document | Quality Management Plan |
|● Set up bug tracking database | |
|● Generate test case checklist using the Functional Requirements, Use Cases, and Metrics Documents, guided by the QA Plan | Project Testing Document Test Cases |
|● Expand test case checklist to more detailed test cases | |
|● Maintain an expandable set of standard unit tests for regression testing | |
|● Provenance - track the necessary steps of data transformation and processing from Data Flow Diagram and Algorithm Specification | Data Provenance Document |
|● Traceability - devise a means to keep track of historical changes to our body of documentation | |
|● Record what is being tracked | |

### 4) Lessons Learned/Next Steps (LL)

|● Conduct post mortem by comparing results to documented requirements | Lessons Learned Report |