Manuscript prepared for Geosci. Model Dev.
with version 2015/04/24 7.83 Copernicus papers of the LaTeX class copernicus.cls.
Date: 12 November 2015

Responses to reviewers and corresponding revisions of the manuscript

immediate

Correspondence to: K. Arthur Endsley (endsley@umich.edu)

1 Re: Anonymous Reviewer #1

> *"The paper struggles to make the case for why having the data directly in the user's browser is desirable."*

Authors' response: Having the data available directly in the user's browser allows for seamless rescaling of the visualization, e.g., changing the stretch "on-the-fly." We have seen performance issues in comparable approaches to this problem, e.g., with Web Mapping Services, which must request the data again from the server whenever scaling changes are desired. We believe scale changes, changes in the palette, and similar changes are in the purview of the client application; as they are merely changes in the application's state, they should be performed asynchronously in the client application without requiring interaction with the remote server.

Changes in manuscript: We have underscored the most important advantages of the Carbon Data Explorer in two places in the revised manuscript: in the "Concluding remarks" and in the last paragraph of the "Introduction." Quote from revised manuscript: "The ability to quickly load spatially explicit scientific data in a web browser allows for the online querying and comparison of measurement values at specific locations across datasets and the rapid, online filtering and aggregation of measurement data. These features are not currently available in Giovanni and alternative data management frameworks and web servers, such as TDS—particularly those that provide only rasterized data representations, such as WMS—are not capable of delivering this level of analysis or speed of interactivity...It contains all the tools necessary for online scientific data analysis in one package, including a non-blocking web server, an extensible, light-weight API, and a user-friendly web application. The text-based JSON format for storage and data interchange is not only fundamentally compatible with web browsers, it allows for scientific data to be manipulated in the web browser, providing asynchronous, rapid filtering and aggregation."

We have also revised section 2.4, "Provision of scientific data on the web," to read: "The RESTful design of the web server's API underscores an important point about having scientific data directly available in the user's web browser. We believe scale changes, changes in the palette, and similar changes are in the purview of the client application; as they are merely changes in the applica-

tion's state, they should be performed asynchronously in the client application without requiring interaction with the remote server. Keeping data on the server requires that new representations are generated even for relatively minor changes in application state. One example is the seamless rescaling of the visualization, e.g., changing the stretch "on-the-fly." We have seen performance issues in comparable approaches to this problem, e.g., with WMS, which must request the data again from the server whenever scaling changes are desired. While similar tools such as Giovanni have also enabled seamless changes to visualization parameters, they don't allow for map-based querying of measurement values or simultaneous comparison of measurement values across datasets, as the Carbon Data Explorer does."

> *"The paper also does not put the CDE's ability to "manage, aggregate, visualize, and share datasets" in the context of the THREDDS Data Server."*

Authors' response: It is possible that we haven't provided enough context for the THREDDS Data Server, as we're less familiar with its operation. We will address that in revising the text. Specifically, we will compare the CDE's features with those of the THREDDS Data Server.

Changes in manuscript: We have revised the manuscript throughout in order to put the Carbon Data Explorer in the context of the THREDDS Data Server and Ferret-THREDDS. Quote from revised manuscript's Introduction: "In its capacity as a data management and data access web server, the Carbon Data Explorer is similar to the THREDDS Data Server (TDS); its analytical capabilities make it similar to Ferret-THREDDS. The Carbon Data Explorer expands on both by providing an integrated front-end for visualization and analysis. While the THREDDS Client Catalog requires data to be registered with XML descriptors, the Carbon Data Explorer Python API has a user-friendly command-line interface that allows for faster, repeatable, one-time registration of data without the need to open a text editor and write XML. In data interchange, it also substitutes bulky XML for light-weight and more human-readable JavaScript Object Notation (JSON). In addition, it eschews the vulnerability-prone Java environment and Tomcat web server for a light-weight, non-blocking web server in Node.js that can be hidden behind a proxy server such as Apache. This design choice trades off the protocol interoperability of TDS, which was not identified as a requirement by our user community, for the ease of development and deployment of web services with newer, JavaScript-based technologies on the server...The ability to quickly load spatially explicit scientific data in a web browser allows for the online querying and comparison of measurement values at specific locations across datasets and the rapid, online filtering and aggregation of measurement data. These features are not currently available in Giovanni and alternative data management frameworks and web servers, such as TDS—particularly those that provide only rasterized data representations, such as WMS—are not capable of delivering this level of analysis or speed of interactivity."

Quote from the revised manuscript's "Other features" section (3.4): "This is similar to the 'virtual variables' of Ferret-THREDDS but provides not only access to an analysis but access to a client for visualizing and interacting with that analysis."

"The text claims the CDE and this approach results in high performance (page 4, line 22), but no performance metrics are given."

Authors' response: We will provide performance metrics for the API in the revised manuscript.

Changes in manuscript: We provided a table of request completion times for the API in the revised manuscript.

"I also cannot understand how a text representation of a number used in the CDE does not massively increase the size of the original data."

Authors' response: I believe this comment is related to a remark in the manuscript on page 4, line 24: "This text-based representation has the added benefits of compressing the data and enabling rapid filtering and aggregation." This remark is unclear and we will revise it in the manuscript. The text-based representation is not compressed relative to the original data, rather, we have eliminated redundancies that would come from a straightforward rendition of the data as text. Specifically, the spatial structure of the data have been separated from the measurement values so they can be transmitted to the client separately and without redundancy.

Changes in manuscript: In the revised manuscript, "This text-based representation..." was revised to read: "This text-based representation is not only compatible with web browsers, it allows for the data to be manipulated directly in the website, providing asynchronous rapid filtering and aggregation. Only the OGC Web Coverage Service (WCS), a protocol also based in a data cube metaphor, allows for this level of interaction and online analysis of data. However, in our experience, stand-alone WCS implementations are usually undocumented or the documentation is relegated to the WCS standard."

"The authors need to be clear what core goals the CDE addresses that these existing tools [THREDDS and Ferret-THREDDS] do not capture. In the context of data sharing, what does CDE do that OpenDAP does not? Is the distinction that OpenDAP generally requires desktop applications, whereas the goal here is to provide web browser access? Again, the case needs to be made for why client-side data provides superior capabilities to the current server-side approach."

Authors' response: We will revise the manuscript to elaborate on the differences between the CDE and Ferret-THREDDS and between the CDE and OpenDAP. As you intuited, a core difference is that the CDE can be run in a web browser on any computer without requiring the installation of dependency libraries (e.g., Java). In addition, the provision of the data directly in the web browser allows for a level of interaction and immediacy with the data we feel are not available with alternatives.

Changes in manuscript: We revised the Introduction of the manuscript to state: "In its capacity as a data management and data access web server, the Carbon Data Explorer is similar to the THREDDS Data Server (TDS); its analytical capabilities make it similar to Ferret-THREDDS. The

Carbon Data Explorer expands on both by providing an integrated front-end for visualization and analysis. While the THREDDS Client Catalog requires data to be registered with XML descriptors, the Carbon Data Explorer Python API has a user-friendly command-line interface that allows for faster, repeatable, one-time registration of data without the need to open a text editor and write XML. In data interchange, it also substitutes bulky XML for light-weight and more human-readable JavaScript Object Notation (JSON). In addition, it eschews the vulnerability-prone Java environment and Tomcat web server for a light-weight, non-blocking web server in Node.js that can be hidden behind a proxy server such as Apache. This design choice trades off the protocol interoperability of TDS, which was not identified as a requirement by our user community, for the ease of development and deployment of web services with newer, JavaScript-based technologies on the server." As stated earlier, we have also revised section 2.4, "Provision of scientific data on the web," in order to "[make the case] for why client-side data provides superior capabilities to the current server-side approach."

> *"Could the concept of JSON data slice access be integrated into the THREDDS software stack as opposed to using MongoDB? ...When datasets can be rather large, I don't practically see modeling centers hosting two copies of their datasets, one for THREDDS (which they will do for CMIPs) and a text version for CDE."*

Authors' response: That is a compelling idea and one worth exploring in future work. I don't see any reason why a module for THREDDS could not be developed to support this. Fast JSON serialization of global-scale climate data is one of the things we've demonstrated with CDE, as many data representations are transformed inside the database on-demand, so we're confident that THREDDS could accomplish this "on the fly" in a similar manner without storing a redundant JSON format.

Changes in manuscript: We added a line to the "Concluding remarks" section: "Also, as a prototype, it is hoped that the software's seamless, interactive visualization and comparison features will inspire the expansion of existing data management and data access frameworks such as TDS and Ferret-THREDDS to support more rich, JavaScript-based visualization libraries."

> *"It is not clear from the introduction that the CDE is software that is installed at a modelling data center as opposed to an online service or API."*

Authors' response: CDE could be installed anywhere but we anticipate CDE would be installed by the people who "own" the data. We will strive to make this more clear in our revisions.

Changes in manuscript: We revised the "Implementation" section to read: "This suite of software could be run on a single computer or separately on multiple computers, each running any UNIX-like operating system (Mac OS X or a GNU/Linux system). The suite is designed to be installed on the data provider's (modeler's) network, with the server API optionally facing the public web."

> *"Likewise, since the software is installed locally, should the name Carbon Data Explorer be generalized?"*

Authors' response: Other reviewers have suggested the name could be changed as the software is not restricted to visualizing carbon science datasets. However, as the software we developed with the aim of visualizing carbon science datasets, and as its features are likely best suited to that purpose, we feel the name is appropriate. As for installation, the CDE could be used only locally (that is, "in-house" by the data providers) but they are still "exploring" their data, as far as that metaphor can carry them. It is also our hope that the serialization to ASCII Grids and GeoTIFFs would allow non-experts to access climate data through the CDE and, thus, it would truly allow scientists in other domains, not just climate experts, to "explore" such data.

Changes in manuscript: We do not feel that a change to the name of the software is required.

"Can large-scale meaningful analysis (beyond mapping) practically be done in real-time on the client browser with current technologies?"

Authors' response: We are not domain experts ourselves, so we collaborated with climate scientists in the development of this software. With the exception of visualizing covariance structure, the analytical capabilities they requested are fully implemented in the CDE, including side-by-side measurement and uncertainty visualization, for global-scale gridded data on the order of 1-degree. We will provide performance metrics for this use case. For datasets with significantly higher spatial resolution, CDE might not be useful.

Changes in manuscript: We provided performance metrics that demonstrate the Carbon Data Explorer's features are feasible in real time on a remote web client.

"What are the 'rich analytical capabilities' mentioned? Are they JavaScript math libraries? Is the text referring to temporal and spatial averages or more complex forms of analysis?"

Authors' response: Our use of the word "rich" may be subjective here. JavaScript math libraries enable the computations but they are, specifically, the aggregation of multiple data slices, the differencing of two data slices, the calculation and visualization of anomalies, and the computation of summary statistics over arbitrary regions. In working with our domain experts, we identified these at the key analytical capabilities they needed. In presenting early versions of the software to groups like DataONE and the OCO-2 Science Team, we did not identify any significantly different analytical workflows for these types of data (Level III gridded products).

Changes in manuscript: We revised this sentence to read: "The Carbon Data Explorer solves this problem by introducing a new API for text-based representations of data cubes, thereby enabling easy integration with and high performance in browser-based web applications while also providing capabilities for dynamic querying, aggregation, differencing, and anomaly calculations."

"Is CMIP6 the first time 'distributed analyses' will be used in a CMIP? I don't think the Meehl et al. 2014 reference support[s] this, but rather the organization of CMIP6 will be distributed (in a non-computing sense."

Authors' response: Thanks for this correction. We will consult this paper again and revise this section for clarity.

Changes in manuscript: We revised this sentence in the Introduction to read: "The next phase of the Climate Model Intercomparison Project, CMIP6, will for the first time allow 'anyone at any time [to] download model data for analysis' (Meehl et al. 2014)."

> *"Secondly, sharing 'web-compatible scientific datasets' was accomplished with the ESGF used in CMIP5."*

Authors' response: We did not give due credit to the prior work done by ESGF here and will revise this section accordingly.

Changes in manuscript: In the Introduction, just after the last-mentioned revision, we added: "As part of CMIP5, the Earth System Grid Federation (ESGF) provides a unified gateway to scientific datasets hosted anywhere in the world. Thus, the ability to share and compare model results should motivate the further development of web-compatible scientific analyses." The ESGF is also mentioned and cited at the bottom of the Introduction.

> *"How does the CDE 'lower or eliminate barriers to bringing scientific results online'? Files and metadata need to be organized prior to loading into CDE. How is the process of making data available via CDE easier than other software such as THREDDS?"*

Authors' response: CDE provides a fairly high-level command-line interface for managing the database, including for inserting data. Files and metadata only need to be organized insofar as they either need to be formatted to match a predefined data model or a data model needs to be written to accommodate them. The latter option is not a low barrier to entry but it does require only a few lines of Python code. In addition, if a data provider consistently generates data in a given format, the data model for that format can be reused for all future datasets, so subsequent database uploads are literally "one-line" commands. Once data are in the database, they are automatically available through the API and a connected CDE front-end web application. We will revise the manuscript to include a comparison to data entry with THREDDS.

Changes in manuscript: The ambiguous statement about "lower[ing] or eliminat[ing] barriers" was replaced; the sentence now reads: "The Carbon Data Explorer was designed specifically to enable climate modeling outputs to be brought online, visualized, and compared." A new paragraph immediately following this was inserted, which reads: "In its capacity as a data management and data access web server, the Carbon Data Explorer is similar to the THREDDS Data Server (TDS); its analytical capabilities make it similar to Ferret-THREDDS. The Carbon Data Explorer expands on both by providing an integrated front-end for visualization and analysis. While the THREDDS Client Catalog requires data to be registered with XML descriptors, the Carbon Data Explorer Python API has a user-friendly command-line interface that allows for faster, repeatable, one-time registration

of data without the need to open a text editor and write XML. In data interchange, it also substitutes bulky XML for light-weight and more human-readable JavaScript Object Notation (JSON). In addition, it eschews the vulnerability-prone Java environment and Tomcat web server for a light-weight, non-blocking web server in Node.js that can be hidden behind a proxy server such as Apache. This design choice trades off the protocol interoperability of TDS, which was not identified as a requirement by our user community, for the ease of development and deployment of web services with newer, JavaScript-based technologies on the server."

> *"The application seems to handle one polygon/ROI, can data 'quickly' be aggregated over many polygons such as counties, states, or countries?"*

Authors' response: The CDE only supports aggregation to one polygon/ROI at a time. We have experimented with offline aggregation over multiple geometries at once, for instance, in preparing a time-series animation of state-level carbon flux (http://spatial.mtri.org/flux/us-states-breathing.html), but this functionality was not identified as an important feature for the CDE.

Changes in manuscript: In the last paragraph of section 3.1, we added the sentence: "Currently, only a single polygon can be used at a time."

> *"Many atmospheric models and ocean models have irregularly spaced grids. Can CDE read / map those grids appropriately?"*

Authors' response: CDE supports irregular spacing (i.e., holes) but not irregular cell sizes or non-rectangular cell shapes. To be precise, CDE supports only "structured" grids and not "unstructured" grids. Support for unstructured grids is a compelling use case as we're aware that there are geostatistical advantages to non-rectangular grid cells. It would be possible to modify CDE to support unstructured grids (e.g., hex-bins) in the future.

Changes in manuscript: We added the following sentence to the end of section 2.3: "At the present time, the Carbon Data Explorer supports only structured grids; that is, the gridded data in a scenario must share the same uniform, rectangular grid."

2 Re: Anonymous Reviewer #2

> *"The tool is called 'Carbon Data Explorer', though the authors indicate that it is not limited to carbon data sources alone. Why not name [it] more generically at this point?"*

Authors' response: Other reviewers have also remarked on the name. However, as the software we developed with the aim of visualizing carbon science datasets, and as its features are likely best suited to that purpose, we feel the name is appropriate.

Changes in manuscript: We do not feel that a change to the name of the software is required.

"In the introduction, one of the key innovations in CDE is that it has a new API for text based representations of data cubes. Does this mean, the data representations are converted to text-based representations internally (as most earth science datasets have large volumes and are not in text formats)? How does this scale for large data? What are the limitations of this approach? Presumably the initial cost of registering the data to the database must be high."

Authors' response: The data are indeed converted to a text-based representation. This has proven to be adequate global gridded datasets at 1-degree resolution The CDE was designed for regional and global carbon science datasets at this similar scales. Performance will suffer at significantly higher resolutions. The initial cost of registering the data to the database is indeed high but needs be done only once by the data manager. Also, the current high cost of registration is largely a reflection of a MongoDB API implementation on our end that is syntactically simpler but not as efficient as other currently available methods. This part of the overall CDE architecture, the "Models "and "Mediators "of the Python API, is amenable to revision by end-users with experience in Python and does not currently represent the potential performance of data registration. MongoDB was also rapidly evolving during our development of the CDE and has since incorporated important performance improvements.

Changes in manuscript: We revised section 2.3 to expand the discussion about the text-based format and to include the sentence: "Thus, a document-oriented database like MongoDB allows the latency associated with preparing scientific data for the web to be pushed offline, during initial registration and insertion of the data to the database."

"It is also mentioned that the text based representations has the added benefits of compressing data and enabling rapid filtering and aggregation. Generally text data don't lend themselves to compression formats and methods. How does the JSON document style compare to other typically compression methods (HDF5, grib)? (The article mentions that the data is only 'slightly' compressed)."

Authors' response: The use of the word "compress" in the manuscript was an error and has regrettably caused confusion. We will clarify this in revising the manuscript. The text-based representation is not compressed relative to the original data, rather, we have eliminated redundancies that would come from a naive implementation of the dataset as text. Specifically, the spatial structure of the data have been separated from the measurement values so they can be transmitted to the client separately and without redundancy.

Changes in manuscript: In the revised manuscript, "This text-based representation..." was revised to read: "This text-based representation is not only compatible with web browsers, it allows for the data to be manipulated directly in the website, providing asynchronous rapid filtering and aggregation."

"The background makes no mention of similar systems that have been developed and are being widely used. NASA itself has a whole host of similar tools (Giovanni, Mirador, etc.). It will be good to describe CDE in the context of such tools and by describing how different CDE is from these tools."

Authors' response: We will place CDE in the context of the tools you mentioned in our revisions. Thanks for your suggestions!

Changes in manuscript: We revised the Introduction, adding the paragraph: "The Carbon Data Explorer shares similar aims with technologies such as NASA's World Wind virtual globe, Giovanni, and Mirador. Compared with World Wind, the Carbon Data Explorer provides access to analytical capabilities that would be awkward or impossible to reproduce in a virtual globe. Also, unlike World Wind, it requires neither a stand-alone installation nor a dependency library such as Java and runs in any web browser. While Mirador allows users to download spatially explicit scientific datasets from NASA missions, it has no analytical or visualization capabilities. The Carbon Data Explorer most closely resembles Giovanni in that both are web-based, map-centered viewers. While Giovanni provides more sophisticated analytical capabilities, the Carbon Data Explorer is designed to deliver results faster and allows for greater customization of the visualization and the querying of measurement values within the web client. In sum, the Carbon Data Explorer is intended for more rapid examination and comparison of climate model outputs by the modelers themselves." In this same section, we added: "The ability to quickly load spatially explicit scientific data in a web browser allows for the online querying and comparison of measurement values at specific locations across datasets and the rapid, online filtering and aggregation of measurement data. These features are not currently available in Giovanni and alternative data management frameworks and web servers, such as TDS—particularly those that provide only rasterized data representations, such as WMS—are not capable of delivering this level of analysis or speed of interactivity."

"The authors claim that the tool supports scalable analysis which is very important when working with large datasets. Can you include some computational estimates that demonstrate this fact?"

Authors' response: We will include performance metrics in our revisions.

Changes in manuscript: We included a table of performance metrics.

"The data example shown in the paper is very coarse (1 deg x 1 deg) and is not representative of modern day satellite products that get down to resolutions of meters (SRTM, MODIS, etc.). To really claim that this technology is viable for such large data, examples should be presented using such datasets (and with associated computational estimates)."

Authors' response: We did not mean to imply that SRTM or MODIS data would be viewed in the CDE. Rather, "geophysical variables...derived from Earth observation satellites or models" are

the expected data. OCO-2, rather, is an example of the satellite platform we had in mind. Regional and global gridded products derived from satellite measurements, such as bias-corrected carbon concentrations, are the kinds of datasets for which the CDE was developed. We will strike the phrase "Earth observation" from the quoted sentence to help mitigate any confusion.

Changes in manuscript: The phrase "Earth observation" was replaced or contextualized throughout the document to emphasize that only measurements from certain satellites, and more generally the predictions from models that operate on those measurements, are intended for use in the Carbon Data Explorer.

3 Re: Anonymous Reviewer #3

> *"Authors claim CDE is offering distributed visualization, however, this is not substantiated in detail; from what I can infer data are always loaded from (server) local files. Visualization is addressing 3D x/y/t cubes plus multi-variables, but it remains essentially 2D plus "movie", no 3D portrayal is mentioned."*

Authors' response: The data cube metaphor is not meant to imply 3D visualization; rather, three to four variables–planar coordinates, measurement value, and time–are represented. Altitude is clearly an important dimension in climate datasets but the CDE does not support a joint visualization of measurement value and altitude.

Changes in manuscript: In the Introduction, we added the sentences: "The three-dimensional data cube metaphor should not be taken to mean that visualizations of the data cube are necessarily 3D. Rather, the minimum three dimensions of the data cube include the two axes of a Cartesian coordinate system and an additional axis for time."

> *"[W]hat is the exact storage scheme for the datacube in MongoDB?"*

Authors' response: Data cube(s) are stored as one or more "scenarios" which may each correspond to a single model run (with particular parameters) or multiple model runs under some unified conditions. Scenarios might also separately encapsulate measurement values and uncertainty, allowing them to be viewed side-by-side in the Coordinated View subsystem. Page 5748, Lines 2-4 state: "Each scenario has one timeline associated with it and gridded data belonging to that scenario are uniquely keyed by their date and time." Slices in time ("X-Y" slices) of the data are stored. Page 5747, Lines 18-19 state: "Specifically, the data are stored and transmitted as JavaScript Object Notation (JSON) documents." We will revise the quoted lines to make this more clear.

Changes in manuscript: Section 2.3, "Data management and storage," was thoroughly revised. New passages include: "MongoDB is one of several 'document-oriented' databases capable of storing semi-structured data as key-value pairs. As the goal was to get the data on the web, we chose MongoDB for its transparent, text-based storage. Alternatives such as Apache Hadoop and Cassandra, while offering performance advantages, do not provide a clear pathway for rendering binary files

as text. These alternatives may faithfully and rapidly operate on chunks of the data but would require that input binary files be split and transformed into some kind of operational format for handling in a map-reduce framework. As no obvious intermediate format was known at the time of development, we opted for a format that most closely resembled the output representation—the native, text-based representation required for the web browser—as the intermediate format to be stored and operated on in the database. This design choice trades off performance for flexibility and the operational demands of bringing data in binary files onto the web.

Array databases such as PostGIS, a spatial extension to the relational database management system (RDBMS) PostgreSQL, are another alternative to MongoDB that we considered. The use of an array database would have satisfied the need for an intermediate format—in this case, array stores—but for the purposes of enabling in-client manipulation of the data (e.g., querying measurement values, changing the stretch) this approach would have required the transformation of requested data to another format, likely text. Based on the authors' experience with PostGIS, there were also no clear performance advantages to array databases. Thus, a document-oriented database like MongoDB allows the latency associated with preparing scientific data for the web to be pushed offline, during initial registration and insertion of the data to the database...

Transforming heterogeneous data to a uniform structure is a typically onerous task. In developing the interface for storing data in MongoDB, we aimed for a flexible system predicated on sensible defaults. The Model class of the Python API defines how measurement values can be read from any file interface available in Python. This flexibility was also driven by the historical development of related software systems. For instance, we discovered that Matlab has changed the format of its saved binary output files over the years from a proprietary data structure to one that is compatible with HDF5. We selected Python and its essential SciPy library as together they provide support for Matlab, HDF4, and HDF5 formats. Thus, our experience is a reminder of the importance of backwards compatibility, which is likely well-recognized in the scientific programming community."

> *"[W]hat part of the analysis is pushed into MongoDB, and what is solved in the middleware?"*

Authors' response: Page 5751, Lines 13 through 18 read: "The temporal aggregation is handled by the MongoDB aggregation pipeline, which facilitates very fast aggregation of multiple X–Y slices (maps spanning time). Spatial aggregation of one or more pixels (an aggregate value spanning a spatially filtered subset) is achieved using a combination of the JSTS Topology Suite JavaScript library and MongoDB's geospatial query operators." We will revise and extend this to elaborate that temporal aggregation *and differencing* are handled by the MongoDB aggregation pipeline; that calculation and display of anomalies is done client-side in JavaScript; that population summary statistics are calculated in the Python API and stored in MongoDB; that all other visualization tweaks and statistical stretching is done "on-the-fly" in JavaScript.

Changes in manuscript: To section 2.3, in addition to the aforementioned expansion in this section, we added: "The metadata also encode population summary statistics, which are calculated by the Python API during insertion to MongoDB, to aid in visualization (e.g., calculating a stretch)." To section 3.1, we added: "Both temporal aggregation and differencing are handled by the MongoDB aggregation pipeline. The calculation and display of anomalies is done client-side in JavaScript. All other visualization tweaks and statistical stretching are done 'on-the-fly' in JavaScript."

> *"In how far do the authors consider this architecture scalable? To this end, at least a few performance figures would have been helpful, even better so a comprehensive evaluation: what are response times in general? where in the architecture is time spent, eg: how much of the query response time goes into MongoDB, and how much into the JavaScript middleware? How does this compare to, eg, C/C++ implementations?"*

Authors' response: We will provide performance metrics in the revised manuscript.

Changes in manuscript: We provided a table of performance metrics.

> *"[W]hile the paper mentions some tools and one standard (WMS) in the field it lacks a solid comparison against immediately "competitors". Hadoop, Array Databases, as well as virtual globes like NASA WorldWind come to my mind."*

Authors' response: We will provide more context for the CDE in relation to Hadoop, Array Databases, and NASA WorldWind within the revised manuscript.

Changes in manuscript: We provided a discussion (mentioned above) in the Introduction on NASA World Wind. We also added a discussion of Hadoop and Array Databases to section 2.3: "MongoDB is one of several 'document-oriented' databases capable of storing semi-structured data as key-value pairs. As the goal was to get the data on the web, we chose MongoDB for its transparent, text-based storage. Alternatives such as Apache Hadoop and Cassandra, while offering performance advantages, do not provide a clear pathway for rendering binary files as text. These alternatives may faithfully and rapidly operate on chunks of the data but would require that input binary files be split and transformed into some kind of operational format for handling in a map-reduce framework. As no obvious intermediate format was known at the time of development, we opted for a format that most closely resembled the output representation—the native, text-based representation required for the web browser—as the intermediate format to be stored and operated on in the database. This design choice trades off performance for flexibility and the operational demands of bringing data in binary files onto the web.

Array databases such as PostGIS, a spatial extension to the relational database management system (RDBMS) PostgreSQL, are another alternative to MongoDB that we considered. The use of an array database would have satisfied the need for an intermediate format—in this case, array stores—but for the purposes of enabling in-client manipulation of the data (e.g., querying measurement values, changing the stretch) this approach would have required the transformation of requested data to

another format, likely text. Based on the authors' experience with PostGIS, there were also no clear performance advantages to array databases. Thus, a document-oriented database like MongoDB allows the latency associated with preparing scientific data for the web to be pushed offline, during initial registration and insertion of the data to the database."

> *"[D]ata ingestion, ie: massaging heterogeneous incoming data to a suitable service structure, typically is an involved task. Section 2.2 does not detail on this, which would be interesting to know: what challenges had to be met? Any innovative approach taken?"*

Authors' response: Some challenges may be merely mundane details for some readers... We discovered that Matlab has changed the format of its saved binary output files over the years from an HDF4-like structure to one requiring an HDF5-compatible reader. We selected Python and its essential SciPy library as together they provide support for both HDF4 and HDF5 formats. Thus, our experience is a reminder of the importance of backwards compatibility, which is likely well-recognized in the scientific programming community. We will add a brief remark on this and other considerations, such as the need for calculating population summary statistics offline, at the end of Section 2.3.

Changes in manuscript: We added a paragraph to section 2.3: "Transforming heterogeneous data to a uniform structure is a typically onerous task. In developing the interface for storing data in MongoDB, we aimed for a flexible system predicated on sensible defaults. The Model class of the Python API defines how measurement values can be read from any file interface available in Python. This flexibility was also driven by the historical development of related software systems. For instance, we discovered that Matlab has changed the format of its saved binary output files over the years from a proprietary data structure to one that is compatible with HDF5. We selected Python and its essential SciPy library as together they provide support for Matlab, HDF4, and HDF5 formats. Thus, our experience is a reminder of the importance of backwards compatibility, which is likely well-recognized in the scientific programming community."

> *"[M]assive binary data encoded in text form seems like a big impediment for transfer and processing. MongoDB querying certainly does not offer competitive performance on datacubes, and only limited functionality. Unfortunately, the paper remains superficial here and does not explain the detailed storage schema."*

Authors' response: Our experience with the CDE is that, for regional and global gridded climate datasets, there are no significant impediments to display and analysis on the web. Comparisons of the performance of NoSQL databases lacked consensus at the time we began development (in 2012) but have since begun to show that, indeed, Cassandra and HBase provide better performance in most applications than MongoDB (e.g., Dede et al. 2013 in *Proceedings of the 4th ACM Workshop on Scientific Cloud Computing*). However, for single nodes, MongoDB can be still provide equal or

better performance than alternatives such as Hadoop (as cited by Nyati et al. 2013, at *International Conference on Advances in Computing, Communications and Informatics, ICACCI*). As for its "limited functionality," in working with our domain experts, the key analytical capabilities they needed were all implemented in the CDE using either MongoDB or more practical front-end capabilities. In presenting early versions of the software to groups like DataONE and the OCO-2 Science Team, we did not identify any analytical workflows for these types of data (Level III gridded products) that we could not support.

Changes in manuscript: The demo video, provided as supplemental material, demonstrates that the Carbon Data Explorer performs its intended functions in real time. We thoroughly revised section 2.3 to provide more detail about the data management and storage system. We make it clear in multiple places within the manuscript that rather than "a big impediment for transfer and processing," the text-based representation is essential to the "rapid, online filtering and aggregation of measurement data;" at the end of the Introduction, at the end of section 2.3, and in the Concluding Remarks.

> *"Example in 2.4: the result to me, following the query logic, should be a 3D cube extending allong the full spatial footprint and temporally reduced to the start and end point indicated in the query. However, authors call the result a 'timeseries' which earlier has been introduced as being 1-D. This might be clarified."*

Authors' response: Only 1D time series or 2D slices of the data cube are made available through the web API. The "t.json" endpoint is not constrained in its design to deliver 1D time series, however, as 2D time series were not required for any of the features identified by the user community and would be expensive to generate, the "t.json" endpoint requires "aggregate" and "interval" keywords so that it can deliver a time series. We will expound on this in the revised manuscript.

Changes in manuscript: In the Introduction, we added the sentences: "The three-dimensional data cube metaphor should not be taken to mean that visualizations of the data cube are necessarily 3D. Rather, the minimum three dimensions of the data cube include the two axes of a Cartesian coordinate system and an additional axis for time." We also added, to the end of section 2.4 where the quoted example is found: "While the 't.json' endpoint could be conceived of as delivering multiple 2D maps ('X-Y' slices), it actually delivers a 1D time series. This is because fully-2D time series were not required for any of the features identified by the user community and would be expensive to generate. The 'aggregate' and 'interval' keywords, which designate the the statistic and bin size of the aggregation, respectively, are required parameters that describe how multiple 2D maps are collapsed into a 1D time series."

> *"[A]uthors characterize retrieval from MongoDB as 'very fast' but without indicating measurements, and no comparison to tools offering the same functionality."*

Authors' response: Performance metrics for the CDE will be included in the revised manuscript.

Changes in manuscript: We provided a table of performance metrics in the revised manuscript.

Manuscript prepared for Geosci. Model Dev. Discuss.
with version 2015/04/24 7.83 Copernicus papers of the LaTeX class copernicus.cls.
Date: 12 November 2015

Distributed visualization of gridded geophysical data:~~a web API for carbon flux~~The Carbon Data Explorer, version 0.2.3

K. A. Endsley[1,2] and M. G. Billmire[1]

[1]Michigan Tech Research Institute (MTRI), Michigan Technological University, Ann Arbor, MI, USA
[2]School of Natural Resources and Environment (SNRE), University of Michigan, Ann Arbor, MI, USA

Correspondence to: K. A. Endsley (endsley@umich.edu)

Abstract

Due to the proliferation of geophysical models, particularly climate models, the increasing resolution of their spatiotemporal estimates of Earth system processes, and the desire to easily share results with collaborators, there is a genuine need for tools to manage, aggregate, visualize, and share datasets. We present a new, web-based software tool – the Carbon Data Explorer – that provides these capabilities for gridded geophysical datasets. While originally developed for visualizing carbon flux, this tool can accommodate any time-varying, spatially explicit scientific dataset, particularly NASA Earth system science Level III products. In addition, the tool's open-source licensing and web presence facilitate distributed scientific visualization, comparison with other datasets and uncertainty estimates, and data publishing and distribution.

1 Introduction

Today's scientific enterprise must consider the challenges and opportunities associated with the growing scale of scientific observations, the need for scalable analyses, and the benefits and obligations of sharing scientific outputs. In ~~geophysical modelsand earth observation science~~climate models, in particular, a wealth of observations can be generated or collected but rich, collaborative insight requires additional frameworks and software tools. Hence, there is a renewed emphasis in the Earth system sciences on tools and best practices for the documentation and sharing of analyses, metadata generation (e.g., Earth System Documentation, ES-DOC), and scientific provenance (e.g., The Kepler Project; Altintas et al., 2004).

In this paper, we describe a new, web-based framework for managing, analyzing, and collaboratively visualizing Earth system science datasets: the Carbon Data Explorer (http://spatial.mtri.org/flux-client/), version 0.2.3. Although the tool's intended use is for carbon science datasets (e.g., regional carbon flux, global carbon concentration), the Carbon Data Explorer is compatible with any time-varying, spatially explicit Earth system dataset

or model output (e.g., land surface temperature, evapotranspiration, aerosol optical thickness). We present the tool as a prototype system that addresses the challenges of increasing scientific data volumes, the need for online analysis, and the desire to share results with collaborators.

Commensurate with the growth of computing power, geophysical models and earth observation systems are producing data with increasingly fine spatial and/or temporal resolution (Nativi et al., 2015). Considering the spatial and temporal dimensions within a dataset simultaneously can be demanding both on computational resources and on a scientist's ability to manage and visualize results. As a conceptual aid, a spatiotemporal dataset consisting of only one parameter of interest can be visualized as a three-dimensional "data cube" (Fig. 1), a representation commonly used in scientific computing (Alder and Hostetler, 2015). The three-dimensional data cube metaphor should not be taken to mean that visualizations of the data cube are necessarily 3D. Rather, the minimum three dimensions of the data cube include the two axes of a Cartesian coordinate system and an additional axis for time. The data cube representation has also gained traction in recent scientific visualization tools; UV-CDAT (Santos et al., 2013) and Panoply (Schmunk, 2015) are two examples.

The Carbon Data Explorer also adopts the data cube as a functional interface for high-volume spatiotemporal data. A map view of a single point in time can be visualized as "slicing" the data cube perpendicular to the time ("T") axis and parallel to the geographic ("X–Y") plane. Conversely, a time series display at one point in space (one pair of geographic coordinates) can be visualized as a narrow threading along the time axis and perpendicular to the X–Y plane. For multivariate data we must begin to construct and think in terms of higher-dimensional data "hypercubes". The Carbon Data Explorer is agnostic as to the type of data contained in data cubes and can simultaneously accommodate any number of variables.

While data cubes work well for storing scientific data offline, web browsers and web applications are designed to work largely with plain text documents (interpreted variously as HTML, XML, JavaScript, or other documents). Non-text formats can be downloaded directly

from an online directory or through File Transfer Protocol (FTP). Indeed, many scientists, unable to procure or unaware of a more sophisticated solution, provide large collections of outputs directly through FTP – essentially a networked folder available to the public. Indexing, searching, or manipulating data must then be done offline.

As an alternative, open API standards, such as the Open Geospatial Consoritum (OGC) Web Map Service (WMS), allow two computers – a web browser and a remote web server – to communicate about data through an agreed-upon protocol (Blower et al., 2013). WMS, as an example, allows web applications to find tiled map images such as those that form the background of modern, interactive web maps like Google Maps. The "Open-source Project for a Network Data Access Protocol" (OPeNDAP) is another ~~open standard and~~ protocol that describes how Hierarchical Data Files (HDFs) and Network Common Data Form (NetCDF) files, among other file types, are stored and accessed (Cornillon et al., 2003).

Thus, dissemination of scientific data on the web typically requires a metadata-driven application programming interface (API) or resource description framework (RDF); these are implemented as a kind of text-based communication protocol that describes (to a computer) where binary data can be found and how they can be accessed. This enables web applications to ultimately retrieve and display data in formats that are not native to the web. However, these APIs incur considerable performance costs when online analysis of datasets is required or when representations are generated dynamically from incoming, real-time data streams (e.g., Sun et al., 2012; Alder and Hostetler, 2015).

The Carbon Data Explorer solves this problem by introducing a new API for text-based representations of data cubes, thereby enabling easy integration with and high performance in browser-based web applications while also providing ~~rich analytical capabilities on dynamic datasets~~capabilities for dynamic querying, aggregation, differencing, and anomaly calculations. This text-based representation ~~has the added benefits of compressing the data and enabling~~ is not only compatible with web browsers, it allows for the data to be manipulated directly in the website, providing asynchronous rapid filtering and aggregation. Only the OGC Web Coverage Service (WCS), a protocol also based in a data cube metaphor, allows for this level of interaction and online analysis of data. However,

in our experience, stand-alone WCS implementations are usually undocumented or the documentation is relegated to the WCS standard.

The adoption of web APIs for sharing data is further evidence of the scientific community's desire to share results with a ~~–~~wider audience. In addition, the ubiquity of social media is bringing online conversations about science, albeit informal, and there are even emerging social networks dedicated to scientific discourse and exchange (e.g., ResearchGate, Academia.edu). This unprecedented interconnectivity is also motivated by best practices in collaborative science. The next phase of the Climate Model Intercomparison Project, CMIP6, will for the first time ~~involve distributed analyses of climate scenarios (Meehl et al., 2014) .~~ allow "anyone at any time [to] download model data for analysis" (Meehl et al., 2014) . As part of CMIP5, the Earth System Grid Federation (ESGF) provides a unified gateway to scientific datasets hosted anywhere in the world. Thus, the ability to share and compare model results should motivate the further development of web-compatible scientific ~~datasets~~analyses.

In response to this need, the Carbon Data Explorer allows data providers to share scientific datasets, analyses, and visualizations directly on the web. A data provider might be a modeler, the principal investigator of an interdisciplinary research team, or a technician or information technology (IT) professional embedded in a research team. NASA estimates that these scientists and model developers spend more than 60 % of their time preparing model inputs and model inter-comparisons (as cited by Rood and Edwards, 2014). The Carbon Data Explorer was designed specifically to ~~lower or eliminate barriers to bringing scientific results onlineand making comparisons~~enable climate modeling outputs to be brought online, visualized, and compared.

In its capacity as a data management and data access web server, the Carbon Data Explorer is similar to the THREDDS Data Server (TDS); its analytical capabilities make it similar to Ferret-THREDDS. The Carbon Data Explorer expands on both by providing an integrated front-end for visualization and analysis. While the THREDDS Client Catalog requires data to be registered with XML descriptors, the Carbon Data Explorer Python API has a user-friendly command-line interface that allows for faster, repeatable, one-time

registration of data without the need to open a text editor and write XML. In data interchange, it also substitutes bulky XML for light-weight and more human-readable JavaScript Object Notation (JSON). In addition, it eschews the vulnerability-prone Java environment and Tomcat web server for a light-weight, non-blocking web server in Node.js that can be hidden behind a proxy server such as Apache. This design choice trades off the protocol interoperability of TDS, which was not identified as a requirement by our user community, for the ease of development and deployment of web services with newer, JavaScript-based technologies on the server.

The scientific datasets supported by the Carbon Data Explorer include any gridded or non-gridded time-varying, spatially explicit data that can be decomposed into one variable at a time. The canonical example of a supported dataset is any NASA Level III scientific data product, defined as "variables mapped on uniform space–time grid scales" (NASA, 2010). These geophysical variables are usually derived from ~~Earth observation satellites~~ satellites (e.g., OCO-2) or models, reanalysis datasets and global or regional Earth system models. Many scientific datasets, particularly Level III products, are already stored as binary ("flat") files or in complex, hierarchical data structures (e.g. NetCDF or HDF) that were designed to accommodate data cubes (Blower et al., 2013).

The Carbon Data Explorer shares similar aims with technologies such as NASA's World Wind virtual globe, Giovanni, and Mirador. Compared with World Wind, the Carbon Data Explorer provides access to analytical capabilities that would be awkward or impossible to reproduce in a virtual globe. Also, unlike World Wind, it requires neither a stand-alone installation nor a dependency library such as Java and runs in any web browser. While Mirador allows users to download spatially explicit scientific datasets from NASA missions, it has no analytical or visualization capabilities. The Carbon Data Explorer most closely resembles Giovanni in that both are web-based, map-centered viewers. While Giovanni provides more sophisticated analytical capabilities, the Carbon Data Explorer is designed to deliver results faster and allows for greater customization of the visualization and the querying of measurement values within the web client. In sum, the Carbon Data Explorer

is intended for more rapid examination and comparison of climate model outputs by the modelers themselves.

In common with the Earth System Grid Federation (Williams et al., 2009), the Carbon Data Explorer aims to provide a common environment for the access to and analysis and visualization of Earth system science datasets. The ability to quickly load spatially explicit scientific data in a web browser allows for the online querying and comparison of measurement values at specific locations across datasets and the rapid, online filtering and aggregation of measurement data. These features are not currently available in Giovanni and alternative data management frameworks and web servers, such as TDS—particularly those that provide only rasterized data representations, such as WMS—are not capable of delivering this level of analysis or speed of interactivity. We expect that these and other features of the Carbon Data Explorer make it a useful contribution to the emerging frameworks for data analysis and intercomparison. The remainder of the paper discusses these and other technical details and describes the full suite of features available.

2 Technical description

2.1 Data sources

In the development and evaluation of the tool, we relied heavily on some reference datasets exemplary of those we intend to support. These included a 1°-by-1° carbon flux estimate at 3-hour time steps from the NASA Carnegie Ames Stanford Approach (CASA) model run with Global Fire Emissions Dataset (GFED) input data and 1°-by-1° carbon concentration (XCO_2) data at 6-day time steps modeled by the Carnegie Institution for Science's Department of Global Ecology at Stanford University (http://dge.stanford.edu/labs/michalaklab/CO2DAAD/). The CASA-GFED model outputs included monthly uncertainty estimates; the XCO_2 data were gridded by kriging from bias-corrected XCO_2 retrievals.

2.2 Implementation

The Carbon Data Explorer has three main components: a Python application programming interface (API) for data management, a web server API, and a client-side JavaScript web application (Fig. 2). From a data provider's perspective, data enter a pipeline from creation to visualization on the web beginning with the Python API, which transforms and stores the data in a database. The data are then automatically available on the web (or a local area network) through the server API and can be viewed and shared through the web application. This suite of software ~~is designed to~~ could be run on a single computer or separately on multiple computers, each running any UNIX-like operating system (Mac OS X or a GNU/Linux system). The suite is designed to be installed on the data provider's (modeler's) network, with the server API optionally facing the public web.

The Python programming language (version 2.7) was chosen as the framework for data management, manipulation, and storage due to its high-level language design, wide adoption in the scientific community, and available open-source libraries. In particular, as many scientific products are stored as hierarchical data files (HDF) or early Matlab files, Python provides fast and robust support for reading scientific data products through the NumPy (Van Der Walt et al., 2011) and SciPy (Jones et al., 2015) libraries. We also expect that Python provides an environment that many data providers are already familiar with or can learn easily should they need to extend the data management API to support new or customized datasets.

The web server and web client are both implemented in JavaScript. This was a strategic but also practical decision. JavaScript is fast and expressive. It is also the de facto language of the web; the only language that is natively supported by every modern web browser (Crockford, 2008). While JavaScript is not widely used for scientific computing, no experience with the language is needed to use the Carbon Data Explorer. We selected Node.js (http://nodejs.org/) as the framework for running a JavaScript server because it provides event-driven request handling, which, like multithreading, can significantly speed up server response time for most web applications (Tilkov and Vinoski, 2010).

Performance testing of the Carbon Data Explorer was conducted using Apache JMeter. For each of the requests listed in Table 1, 10 identical, repeated queries were sent to the server over 30 seconds. Tests were done sequentially and were performed three times with several hours to several days between the runs to ensure general results.

2.3 Data management and storage

Open data APIs for science capitalize on storing and sharing text-based metadata associated with scientific data that are stored in a binary or hierarchical format. We took this a step further and designed a data model that is text-only; that is, the format of the data both on-disk and when transmitted over the web is plain text. Specifically, the data are stored and transmitted as JavaScript Object Notation (JSON) documents. ~~This approach not only ensures compatibility with web browsers but also slightly compresses the data.~~ These JSON documents are stored in a MongoDB database instance which handles indexing and retrieval of ~~such~~ plain-text representations~~reasonably well~~.

MongoDB is one of several "document-oriented" databases capable of storing semi-structured data as key-value pairs. As the goal was to get the data on the web, we chose MongoDB for its transparent, text-based storage. Alternatives such as Apache Hadoop and Cassandra, while offering performance advantages, do not provide a clear pathway for rendering binary files as text. These alternatives may faithfully and rapidly operate on chunks of the data but would require that input binary files be split and transformed into some kind of operational format for handling in a map-reduce framework. As no obvious intermediate format was known at the time of development, we opted for a format that most closely resembled the output representation—the native, text-based representation required for the web browser—as the intermediate format to be stored and operated on in the database. This design choice trades off performance for flexibility and the operational demands of bringing data in binary files onto the web.

Array databases such as PostGIS, a spatial extension to the relational database management system (RDBMS) PostgreSQL, are another alternative to MongoDB that we considered. The use of an array database would have satisfied the need for an intermediate

format—in this case, array stores—but for the purposes of enabling in-client manipulation of the data (e.g., querying measurement values, changing the stretch) this approach would have required the transformation of requested data to another format, likely text. Based on the authors' experience with PostGIS, there were also no clear performance advantages to array databases. Thus, a document-oriented database like MongoDB allows the latency associated with preparing scientific data for the web to be pushed offline, during initial registration and insertion of the data to the database. In addition, MongoDB features an aggregation pipeline, which allows us to make sophisticated queries such as "net carbon flux over the last 16 days." The web server API, which facilitates connections to the MongoDB instance, contains libraries that enable further sophistication with queries, applying fast arithmetic operations for queries such as "the difference between carbon concentration (in ppm) today and this day last year."

~~Scientific data in the Carbon Data Explorer are conceived of as belonging to a particular run of a "scenario", i.e., a specific geophysical modeling objective. Each scenario has one timeline associated with it and gridded data belonging to that scenario are uniquely keyed by their date and time. Non-gridded data are assigned arbitrary unique identifiers, making it possible to have two pieces of non-gridded data that represent the same instance in time (or span of time) associated with the same scenario. The gridded data in a scenario must also share the same uniform, rectangular grid. This allows data values to be stored and transmitted independent of the spatial reference information, compressing the data storage and stream to levels that allow for rapid retrieval and display on the web. The "X–Y" values associated with gridded data – the spatial coordinates of each data point – are stored separately and transmitted only once to the web application, eliminating redundancy associated with viewing multiple points in the time series. In contrast, non-gridded data are stored with their X–Y values and transmitted as GeoJSON, a spatially explicit form of JSON, as their spatial structure may vary.~~

Users can shuttle scientific data into and out of the MongoDB instance by directly interacting with the Carbon Data Explorer Python API classes or by using a set of accompanying command line tools designed to ease workflow. Command line tools are available for query-

ing database contents as well as for loading, renaming, and removing datasets from the database. When loading a dataset, its metadata must be specified either via command line argument or via an accompanying JSON file. Examples of required metadata parameters include column identifiers, grid resolution, units, starting timestamp, and time step length. These metadata parameters inform the correct methods for transforming and querying the data for use within the web server API. The metadata also encode population summary statistics, which are calculated by the Python API during insertion to MongoDB, to aid in visualization (e.g., calculating a stretch).

The transformation of data from binary or hierarchical flat files to a database representation is facilitated by two Python classes, Models and Mediators, which are loosely based on the Transformation Interface described by Bulka (2001). The Model class is a data model that describes what a scientific dataset looks like; whether it is a time series of gridded maps or a covariance matrix, for instance. The Mediator class describes how a given Model should be read from and the data it contains translated to a database representation. Some basic Mediator and Model classes are provided in the Python API. It is expected that data providers with a particular output format can easily create new Model and Mediator subclasses to seamless read and write data to and from the MongoDB database and the files in which their data are currently stored.

Transforming heterogeneous data to a uniform structure is a typically onerous task. In developing the interface for storing data in MongoDB, we aimed for a flexible system predicated on sensible defaults. The Model class of the Python API defines how measurement values can be read from any file interface available in Python. This flexibility was also driven by the historical development of related software systems. For instance, we discovered that Matlab has changed the format of its saved binary output files over the years from a proprietary data structure to one that is compatible with HDF5 (MathWorks, 2015) . We selected Python and its essential SciPy library as together they provide support for Matlab, HDF4, and HDF5 formats. Thus, our experience is a reminder of the importance of backwards compatibility, which is likely well-recognized in the scientific programming community.

Scientific data in the Carbon Data Explorer are conceived of as belonging to a particular run of a "scenario," i.e., a specific geophysical modeling objective. Data cube(s) are stored as one or more scenarios. During data insertion to MongoDB, the "X-Y" slices at all time points are stored as separate documents. Each scenario has one timeline associated with it and gridded data belonging to that scenario are uniquely keyed by their date and time.

Non-gridded data are assigned arbitrary unique identifiers, making it possible to have two pieces of non-gridded data that represent the same instance in time (or span of time) associated with the same scenario. At the present time, the Carbon Data Explorer supports only structured grids; that is, the gridded data in a scenario must share the same uniform, rectangular grid. Measurement values are stored and transmitted independent of the spatial reference information, eliminating redundancies and allowing for rapid retrieval and display on the web. The "X-Y" values associated with gridded data—the spatial coordinates of each data point—are stored separately and transmitted only once to the web application. In contrast, non-gridded data are stored with their X-Y values and transmitted as GeoJSON, a spatially explicit form of JSON, as their spatial structure may vary.

2.4 Provision of scientific data on the web

The Carbon Data Explorer web server API is designed to work out-of-the-box so that data can be served and visualized with the web application on any web browser connected to the same local area network. That is, any user on the same network as the computer running the server can access the Carbon Data Explorer through its internet protocol (IP) address in their web browser. Data providers might choose to host the Carbon Data Explorer locally so as to keep their data private and collaborate internally. Deploying the server and web application on the public web is also easy, though it may require some familiarity with networking technology.

The web server makes data available as resources that are each associated with a uniform resource identifier (URI). The model used for organizing these resources in a single namespace (i.e., under a single host or domain name) is the Representational State Transfer (REST) model (Fielding and Taylor, 2000), in which different representations of data are

provisioned with semantics. For example, a list of all available scenarios can be obtained at, e.g., "/scenarios.json" as a JSON document. Alternately, the metadata for a single scenario, e.g., the "casa_gfed_2004" scenario, can be obtained at "/scenarios/casa_gfed_2004.json".

As another example, a map of carbon flux on 18 January 2004 at 03:00 UTC from the "casa_gfed_2004" scenario can be obtained at "/scenarios/casa_gfed_2004/xy.json?time=2004-01-18T03:00:00" where "xy" refers to the X–Y values from our data cube (i.e., a geographic map). This distinguishes map data from a time series, which could be requested in JSON format from the "t.json" resource, e.g., "t.json?start=2003-12-22T03:00&end=2005-01-01T00:00&aggregate=mean&interval=daily". While the "t.json" endpoint could be conceived of as delivering multiple 2D maps ("X-Y" slices), it actually delivers a 1D time series. This is because fully-2D time series were not required for any of the features identified by the user community and would be expensive to generate. The "aggregate" and "interval" keywords, which designate the the statistic and bin size of the aggregation, respectively, are required parameters that describe how multiple 2D maps are collapsed into a 1D time series.

These limited examples showcase only a small part of the functionality of the web server's API (Table 2). These relatively human-readable URIs allow experienced users to download data directly if preferred. They are also used behind-the-scenes in the web application to programmatically request data as indicated by a user through its graphical user interface (GUI).

The RESTful design of the web server's API underscores an important point about having scientific data directly available in the user's web browser. We believe scale changes, changes in the palette, and similar changes are in the purview of the client application; as they are merely changes in the application's state, they should be performed asynchronously in the client application without requiring interaction with the remote server. Keeping data on the server requires that new representations are generated even for relatively minor changes in application state. One example is the seamless rescaling of the visualization, e.g., changing the stretch "on-the-fly." We have seen performance issues in comparable approaches to this problem, e.g., with WMS, which must request the data again

from the server whenever scaling changes are desired. While similar tools such as Giovanni have also enabled seamless changes to visualization parameters, they don't allow for map-based querying of measurement values or simultaneous comparison of measurement values across datasets, as the Carbon Data Explorer does.

3 Features

In the Carbon Data Explorer client application, a rich user interface (Fig. 3) provides users many options for visualizing, exploring, comparing, and ultimately sharing geophysical data that have been previously imported with the Python API and made available to the client through the web server API. In Table 3 and in the subsequent text, we highlight some of the chief features available to users. A demonstration video (doi:10.5281/zenodo.18941) of the web browser application can also be seen through a link on the project website (http://spatial.mtri.org/flux/).

3.1 Spatial visualization and analysis

The default view in the Carbon Data Explorer client application is the "Single Map View" which displays a geographic view (an "X–Y slice") of the data at a particular time. The Map Settings define the map projection used (currently a choice between Equirectangular or Mercator) and what kind of basemap should be drawn (e.g., continents with or without political boundaries). When gridded data are drawn on the map, the Symbology options allow a user to specify a color palette from a selection of colorblind-safe, perceptually linear color scales designed by Brewer (2014). Both sequential and diverging color scales are available for linear data that are either constantly increasing or are diverging from a threshold or mean value, respectively. The number of bins in the color scale can also be specified. While the default stretch of the data to the color scale is a standard deviation about the mean, both the measure of central tendency and the number of standard deviations can be changed. As an alternative to this stretch, the scale can be stretched to the domain of the data or any

arbitrary endpoints as entered by the user. A binary map can also be shown, where a single color is used to code for grid cells or data points that fall within a user-specified range.

The Single Map View allows the user to explore the data as in a geographic information system (GIS). Users can zoom into the map display, pan the map around, and query the value of a data point by hovering over it with the cursor. Non-gridded data can be plotted on top of gridded data and automatically share the same color scale. An optional border drawn around the non-gridded data points can help to distinguish them from the gridded data. This feature allows, for example, the direct comparison of gridded carbon concentration with bias-corrected retrievals from atmospheric sounding.

Data can be quickly aggregated in time or space from within the web application. The temporal aggregation is handled by the MongoDB aggregation pipeline, which facilitates very fast aggregation of multiple X–Y slices (maps spanning time). Spatial aggregation of one or more pixels (an aggregate value spanning a spatially filtered subset) is achieved using a combination of the JSTS Topology Suite JavaScript library and MongoDB's geospatial query operators. Both temporal aggregation and differencing are handled by the MongoDB aggregation pipeline. The calculation and display of anomalies is done client-side in JavaScript. All other visualization tweaks and statistical stretching are done "on-the-fly" in JavaScript.

Spatial filters can be drawn directly on the map interface or imported as polygons defined using GeoJSON or well-known text (WKT), a human-readable representation of geometry. Currently, only a single polygon can be used at a time. Map data can also be differenced – one X–Y slice can be subtracted from another (from a different scenario and same time or vice-versa). This may help in identifying deviation from a seasonal trend or other anomalies as well as help in identifying differences between different models or different model runs of the same time step.

3.2 Time series analysis

While in the Single Map View, the map can be animated in time, updating its display (at time "T") with the next X–Y slice from our data cube. This update is seamless when the

web server API is hosted on the same local network or when viewed over a high-speed internet connection, making a refresh rate of one second practical for quickly reviewing model results at a rate of a few hours, days, or months every second (depending on the temporal resolution of the data). A slower animation speed can be selected for a more moderate pace. This high data throughput is made possible by the text-based data format ~~and compression~~ discussed earlier. Aggregates and differenced data can also be animated in time.

A line plot at the bottom of the map shows the "global" time series for the currently viewed scenario by default; it is the aggregate mean value across the X–Y domain at each point in time. This provides an overview of the overall trend in the data across the spatial domain. When a spatial filter is applied, an aggregate time series for only that region can be generated. The non-aggregate time series for a specific pixel can also be obtained by clicking on that grid point in the map. Retrieval of a time series data for the line plot is slower than other data requests but it still returns results in seconds.

3.3 Multiple-time and multiple-model comparison

The "Coordinated View" allows comparison of multiple adjacent map views; it is essentially a grid of multiple Single Map View elements. These maps synchronize their extent whenever the user pans or zooms so that the same portion of the globe is displayed in each one. The user's cursor will now display not just the value of a data point in one map but the value at that those spatial coordinates in every map facilitating pixel-to-pixel comparison across the maps. Up to nine (9) maps can be viewed at once which allows for nine different time points or nine different models to be viewed simultaneously.

3.4 Other features

A user's Map Settings, Symbology, and other global settings are stored in the web browser so that, upon closing the browser and returning to the web application later, the same color scale, map projection, and other settings are automatically applied. This allows users to

customize their view of a dataset and their workspace within the tool. All of these settings can also be encoded as a URI (or URL). This allows specific views of a dataset to be "bookmarked" or shared with others over the web. With this feature, a user can apply a specific color scale, stretch (or threshold to highlight a particular anomaly) or an aggregate or differenced model result and then share a link that ensures that their team member will see the data exactly the same way. This is similar to the "virtual variables" of Ferret-THREDDS but provides not only access to an analysis but access to a client for visualizing and interacting with that analysis. For offline storage and sharing of results, model visualizations and data slices can be exported as image files, CSVs (for non-gridded data), or as geospatial data (for gridded data) in the form of ESRI ASCII Grid files or GeoTIFFs; the latter two formats enable model results to be downloaded and opened in a desktop GIS like ArcGIS or QGIS.

4 Concluding remarks

The Carbon Data Explorer is presented as a ~~-~~prototype for a ~~-~~comprehensive data management, analysis, visualization, and sharing framework for Earth system science datasets, particularly gridded spatiotemporal datasets (e.g., NASA Level III data products). ~~With its unique~~ It contains all the tools necessary for online scientific data analysis in one package, including a non-blocking web server, an extensible, light-weight API, and a user-friendly web application. The text-based ~~data representations, gridded scientific datasets can be rapidly manipulated , analyzed, and displayed on the web~~JSON format for storage and data interchange is not only fundamentally compatible with web browsers, it allows for scientific data to be manipulated in the web browser, providing asynchronous, rapid filtering and aggregation. In response to the new protocols of CMIP6, the Carbon Data Explorer provides a ~~-~~framework for the distributed analysis of climate model outputs. Analyses can effectively be "bookmarked" with URIs serving as permanent links to a particular visualization and analysis of a data cube at a given point in time. The framework's open source licensing and web integration enable the visualization and sharing of scientific data through either a ~~-~~secure network or public portal. Also, as a prototype, it is hoped that the software's seamless,

interactive visualization and comparison features will inspire the expansion of existing data management and data access frameworks such as TDS and Ferret-THREDDS to support more rich, JavaScript-based visualization libraries. It is hoped they will also facilitate the future improvement of the Carbon Data Explorer and the inspiration of similar and better tools for Earth system science.

Code availability

The source code is available from GitHub (https://github.com/MichiganTechResearchInstitute/CarbonDataExplorer) under the MIT license. A built version of the web browser application ("flux-client") is available upon request. This can significantly help integration and deployment of the visualization and analysis front-end.

The Supplement related to this article is available online at doi:10.5194/gmdd-0-1-2015-supplement.

Acknowledgements. The software described in this paper was developed by the Michigan Tech Research Institute (MTRI) in partnership with Anna Michalak at the Carnegie Institution for Science's Department of Global Ecology at Stanford University and with funding from NASA (Grant #NNX12AB90G). The authors would also like to acknowledge the contributions of Nicholas Molen to the early development of the Carbon Data Explorer and of Reid Sawtell to the front-end web browser application. Special thanks go to Mae Qiu at Stanford University and Vineet Yadav at NASA Jet Propulsion Laboratory for their contributions to design and testing. The authors would also like to thank James Arnott and Scott Kalafatis at the School of Natural Resources and Environment at the University of Michigan for their reviews of an early draft of this paper.

References

Alder, J. R. and Hostetler, S. W.: Web based visualization of large climate data sets, Environ. Modell. Softw., 68, 175–180, doi:10.1016/j.envsoft.2015.02.016, 2015.

Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., and Mock, S.: Kepler: an extensible system for design and execution of scientific workflows, in: Proceedings 16th International Conference on Scientific and Statistical Database Management, Santorini Island, Greece, 21–23 June 2004, 423–424, doi:10.1109/SSDM.2004.1311241, 2004.

Blower, J. D., Gemmell, A. L., Griffiths, G. H., Haines, K., Santokhee, A., and Yang, X.: A web map service implementation for the visualization of multidimensional gridded environmental data, Environ. Modell. Softw., 47, 218–224, doi:10.1016/j.envsoft.2013.04.002, 2013.

Brewer, C. A.: Color Brewer 2.0, available at: http://www.colorbrewer.org (last access: 24 June 2015), 2014.

Bulka, A.: Transformation Interface Design Pattern, available at: http://www.andypatterns.com/files/74091232001410AndyBulkaTransformationInterfacePattern.pdf (last access: 24 June 2015), 2001.

Cornillon, P., Gallagher, J., and Sgouros, T.: OPeNDAP: Accessing data in a distributed, heterogeneous environment, Data Science Journal, 2, 164–174, doi:10.2481/dsj.2.164, 2003.

Crockford, D.: JavaScript: The Good Parts, 1st Edn., O'Reilly Media, Inc., doi:10.1241/johokanri.44.584, 2008.

Fielding, R. T. and Taylor, R. N.: Principled design of the modern Web architecture, in: ICSE '00: Proceedings of the 22nd International Conference on Software Engineering, 407–416, doi:10.1145/337180.337228, 2000.

Jones, E., Oliphant, T. E., and Peterson, P.: SciPy: Open source scientific tools for Python, available at: http://www.scipy.org/ (last access: 24 June 2015), 2015.

MathWorks: Mat-File Versions, http://www.mathworks.com/help/matlab/import{_}export/mat-file-versions.html, 2015.

Meehl, G. A., Moss, R., Taylor, K. E., Eyring, V., Stouffer, R. J., Bony, S., and Stevens, B.: Climate model intercomparisons: preparing for the next phase, EOS T. Am. Geophys. Un., 95, 77–78, doi:10.1002/2014EO090001, 2014.

NASA: Data Processing Levels for EOSDIS Data Products, available at: http://science.nasa.gov/earth-science/earth-science-data/data-processing-levels-for-eosdis-data-products/ (last access: 24 May 2015), 2010.

Nativi, S., Mazzetti, P., Santoro, M., Papeschi, F., Craglia, M., and Ochiai, O.: Big Data challenges in building the Global Earth Observation System of Systems, Environ. Modell. Softw., 68, 1–26, doi:10.1016/j.envsoft.2015.01.017, 2015.

Rood, R. B. and Edwards, P. N.: Climate Informatics: Human Experts and the End-to-End System, Earthzine, May, available at: http://earthzine.org/2014/05/22/climate-informatics-human-experts-and-the-end-to-end-system/ (last access: 24 June 2015), 2014.

Santos, E., Poco, J., Wei, Y., Liu, S., Cook, B., Williams, D. N., and Silva, C. T.: UV-CDAT: analyzing climate datasets from a user's perspective, Comput. Sci. Eng., 15, 94–103, doi:10.1109/MCSE.2013.15, 2013.

Schmunk, R. B.: Panoply netCDF, HDF and GRIB Data Viewer, available at: http://www.giss.nasa.gov/tools/panoply/, last access: 24 June 2015.

Sun, X., Shen, S., Leptoukh, G. G., Wang, P., Di, L., and Lu, M.: Development of a web-based visualization platform for climate research using Google Earth, Comput. Geosci., 47, 160–168, doi:10.1016/j.cageo.2011.09.010, 2012.

Tilkov, S. and Vinoski, S.: Node.js: using JavaScript to build high-performance network programs, IEEE Internet Comput., 14, 80–83, doi:10.1109/MIC.2010.145, 2010.

Van Der Walt, S., Colbert, S. C., and Varoquaux, G.: The NumPy array: a structure for efficient numerical computation, Comput. Sci. Eng., 13, 22–30, doi:10.1109/MCSE.2011.37, 2011.

Williams, D. N., Ananthakrishnan, R., Bernholdt, D. E., Bharathi, S., Brown, D., Chen, M., Chervenak, A. L., Cinquini, L., Drach, R., Foster, I. T., Fox, P., Fraser, D., Garcia, J., Hankin, S., Jones, P., Middleton, D. E., Schwidder, J., Schweitzer, R., Schuler, R., Shoshani, A., Siebenlist, F., Sim, A., Strand, W. G., Su, M., and Wilhelmi, N.: The earth system grid: enabling access to multimodel climate simulation data, B. Am. Meteorol. Soc., 90, 195–205, doi:10.1175/2008BAMS2459.1, 2009.

Table 1. Results of load testing in Apache JMeter; network speeds are in seconds to request completion. The off-network tests were performed over a wireless internet connection; on-network tests were performed with a wired, direct network connection to the server.

Request	Data extent, resolution	Mean return speed, s (n=30) Off-network	On-network
Grid structure[a]	N. America, 1-by-1 deg.	0.17	0.03
Grid structure[b]	World, 1-by-1 deg.	0.44	0.08
Gridded X-Y data[c]	N. America, 1-by-1 deg.	0.14	0.02
Gridded X-Y data[d]	World, 1-by-1 deg.	0.34	0.12
Temporal aggregation, 40 X-Y grids[e]	N. America, 1-by-1 deg.	0.49	0.36
Temporal aggregation, 40 X-Y grids[f]	World, 1-by-1 deg.	2.90	2.31
Time series at X-Y point[g]	N. America, 1-by-1 deg.	0.10	0.06
Time series at X-Y point[h]	World, 1-by-1 deg.	0.75	0.62
Time series for region, 684 cells[i]	N. America, 1-by-1 deg.	0.12	0.08
Time series for region, 684 cells[j]	World, 1-by-1 deg.	0.79	0.72

Request URIs:

a: `/flux/api/scenarios/casa_gfed_2004/grid.json`;

b: `/api/scenarios/r2_xco2_kriged/grid.json`;

c: `/api/scenarios/casa_gfed_2004/xy.json?time=2004-05-01T03:00`;

d: `/api/scenarios/r2_xco2_kriged/xy.json?time=2009-06-15T00:00`;

e: `/api/scenarios/casa_gfed_2004/xy.json?start=2004-05-01T03:00&end=2004-05-06T03:00&aggregate=positive`;

f: `/api/scenarios/r2_xco2_kriged/xy.json?start=2009-06-01T00:00&end=2010-02-01T00:00&aggregate=positive`;

g: `/api/scenarios/casa_gfed_2004/t.json?coords=POINT(-50.5+69.5)&start=2004-05-01T03:00&end=2004-05-02T03:00`;

h: `/api/scenarios/r2_xco2_kriged/t.json?coords=POINT(-50.5+69.5)&start=2009-06-15T00:00&end=2009-08-05T00:00`;

i: `/api/scenarios/casa_gfed_2004/t.json?start=2004-05-01T03:00&end=2004-05-02T03:00&interval=hourly&geom=POLYGON((-97%2B46%2C-101%2B37%2C-93%2B35%2C-89%2B42%2C-97%2B46))`;

j: `/api/scenarios/r2_xco2_kriged/t.json?start=2009-06-15T00:00&end=2009-08-05T00:00&geom=POLYGON((-97%2B46%2C-101%2B37%2C-93%2B35%2C-89%2B42%2C-97%2B46))`;

Table 2. Entry points for the Carbon Data Explorer web server API.

Web Server API Entry Point	Description
`/scenarios.json`	Requests metadata for all or selected datasets
`/[scenario]/grid.json`	Requests a GeoJSON representation of the scenario's X–Y grid
`/[scenario]/xy.json`	Requests data values corresponding to the scenario's spatial grid
`/[scenario]/t.json`	Requests time series data

Table 3. List of features (present when marked with an "X") in the two visualization modes of the Carbon Data Explorer web browser application.

Category	Feature	Single Map View	Coordinated View
Mapping	Gridded data map display	×	×
	Non-gridded data map display	×	
	Basemap selection	×	×
	Map projection selection	×	×
	Map zoom and pan	×	×
	View multiple map frames simultaneously		×
	View global/continental/regional data	×	×
Symbology	Color palette selection	×	×
	Scaling, stretching, and thresholding	×	×
Analysis	Animation	×	
	Map pixel querying	×	×
	Temporal queries and aggregation	×	
	Spatial queries and aggregation	×	
	Time series line plot	×	
	Display difference of two maps	×	
	Side-by-side comparison		×
Sharing	Data export (as image or GIS file)	×	
	Browser remembers settings	×	
	Shareable URI/URL generation	×	

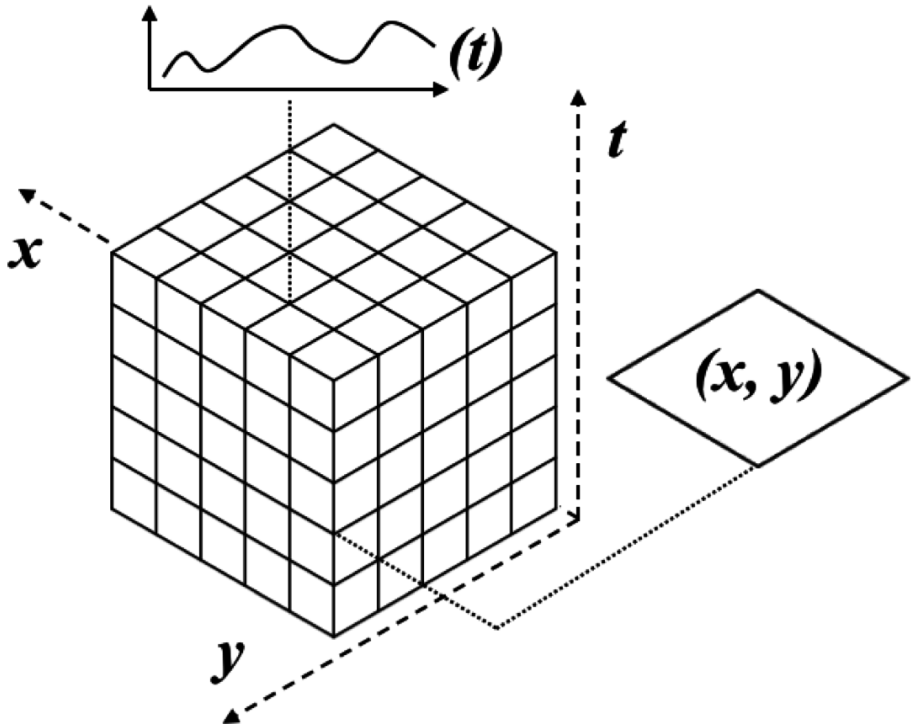

Figure 1. A three-dimensional "data cube" in which spatial data of two dimensions (e.g., latitude and longitude) are combined with a third dimension of time. In this view, a horizontal slice perpendicular to the time (t) axis corresponds to a geographic map while a line parallel to the time (t) axis represents a time series.

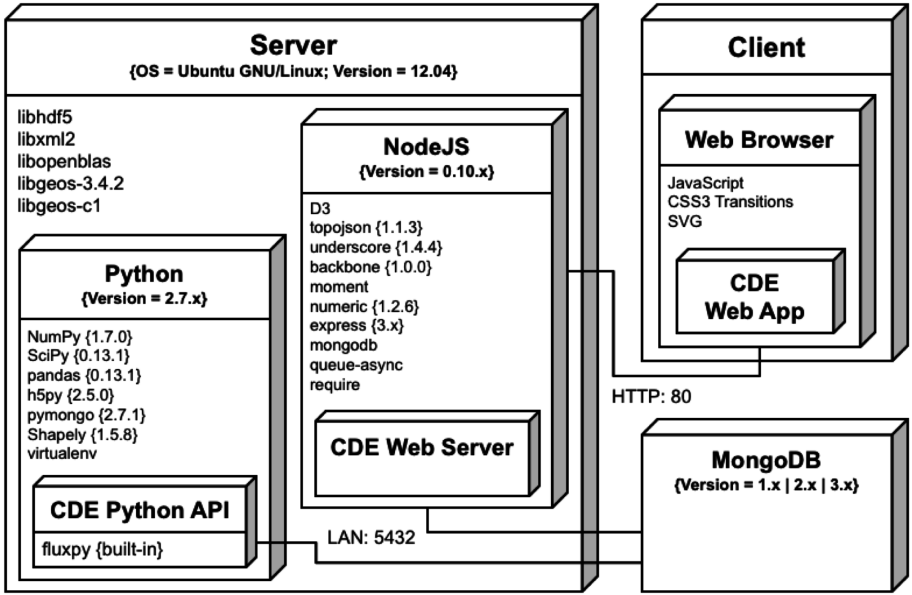

Figure 2. A unified modeling language (UML) deployment diagram for the Carbon Data Explorer (CDE), illustrating the configuration and connections between the components as currently deployed.

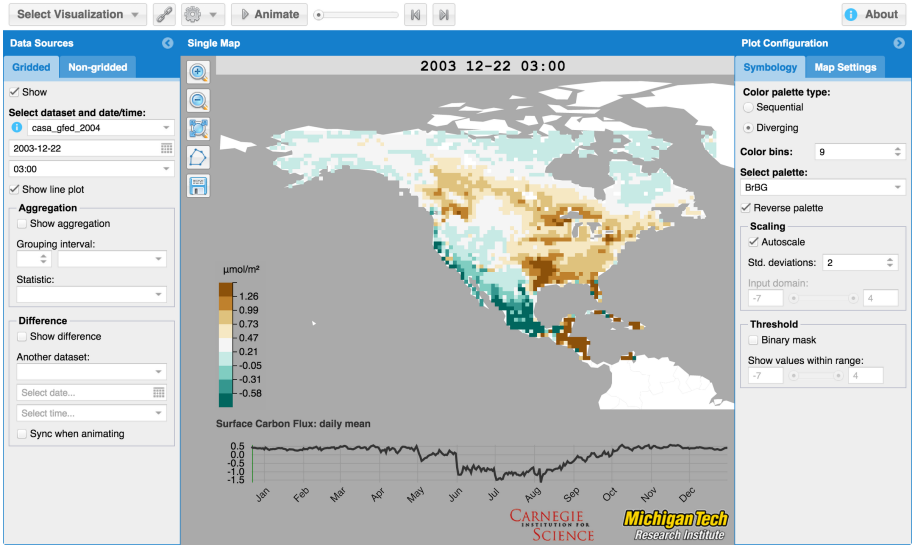

Figure 3. Screenshot of the Carbon Data Explorer web browser application in the “Single Map View” mode.