Finding Tropical Cyclones on a Cloud Computing Cluster: Using Parallel Virtualization for Large-Scale Climate Simulation Analysis

D. Hasenkamp, A. Sim, M. Wehner and K. Wu
Lawrence Berkeley National Laboratory, USA
{dhasenkamp, asim, mfwehner, kwu}@lbl.gov

ABSTRACT

Extensive computing power has been used to tackle issues such as climate changes, fusion energy, and other pressing scientific challenges. These computations produce a tremendous amount of data; however, many of the data analysis programs currently only run a single processor. In this work, we explore the possibility of using the emerging cloud computing platform to parallelize such sequential data analysis tasks. As a proof of concept, we wrap a program for analyzing trends of tropical cyclones in a set of virtual machines (VMs). This approach allows the user to keep their familiar data analysis environment in the VMs, while we provide the coordination and data transfer services to ensure the necessary input and output are directed to the desired locations. This work extensively exercises the networking capability of the cloud computing systems and has revealed a number of weaknesses in the current cloud system software. In our tests, we are able to scale the parallel data analysis job to a modest number of VMs and achieve a speedup that is comparable to running the same analysis task using MPI. However, compared to MPI based parallelization, the cloud-based approach has a number of advantages. The cloud-based approach is more flexible because the VMs can capture arbitrary software dependencies without requiring the user to rewrite their programs. The cloud-based approach is also more resilient to failure; as long as a single VM is running, it can make progress while as soon as one MPI node fails the whole analysis job fails. In short, this initial work demonstrates that a cloud computing system is a viable platform for distributed scientific data analyses traditionally conducted on dedicated supercomputing systems.

Keywords: tropical cyclone, cloud computing, virtualization, virtual machine, parallelization, climate data

1. INTRODUCTION

Climate change is one of the most pressing issues we as a species currently face. In recent years, evidence for global warming has become increasingly difficult to refute, with numerous surveys showing clear increasing trends in average sea-surface and air temperatures. The National Oceanographic Data Center, for example, compiled data from a survey buoy in the Gulf of Alaska showing a 1 degree Celsius increase in average sea-surface temperature from 1975 to 2005 [1]. Both scientists and governments seek to understand the potential threat to human society posed by this change. Many damaging consequences must be considered: a warming climate could cause problems such as heat waves, biological extinctions, violent weather conditions, and pronounced rises in sea level. By understanding how such threats propagate and worsen, governments can plan ahead to mitigate many potential damages. Moreover, by understanding the role of human activity in creating these threats, we can modify our own actions to lessen the stress we place on our planet.

A promising method of predicting the future effects of climate change is to run large-scale simulations of the global climate many years into the future [2, 3]. Climatologists discretize our globe, setting up differential equations to model Earth's climate. These differential equations are run forward using numerical methods, such as Runge-Kutta [4], over many time-steps encapsulating decades or centuries; data for the entire globe at certain time steps are stored into large repositories. Because the Earth's climate is extremely sensitive to initial conditions (the famous "butterfly effect") [5], these simulations are commonly run many times with slightly varying initial conditions. A large number of analysis techniques have been developed to make use of these ensembles to generate more accurate understanding of climate models [19][20].

The data generated by these simulations is massive. For example, a climate model currently in use, called fvCAM [3], runs over 15 simulated years with output every 6 simulated hour, and a mesh point resolution of 0.5 degrees latitude by 0.625 degrees longitude generates roughly 500GB of data. Climatologists plan to run this model many times for 100 simulated years with with different initial conditions, thus generating petabytes, of raw data.

This raw data requires significant compute power to yield any useful information to climatologists, who are interested in finding a variety of climatological phenomena including heat waves, hurricane counting/tracking, alterations in wind patterns, etc. To conduct these analysis tasks, we need to transfer terabytes even petabytes of raw data from storage systems to the computational systems. Managing such volume of data is frequently tedious and time consuming. Many of the climate analysis programs avoid such issues by working with one data file that is already on disk. To better utilize the computer resources, we seek to automate the data management for such analysis programs.

Many of these climate data analysis programs are parallelizable in a data-driven manner, since each time step in the data can usually be analyzed independently of others, for example, to search for hurricane conditions or high-temperature regions. Hence, it is often possible to greatly speed up these analysis programs by running them many times in parallel, giving each process a subset of the data.

Additionally, these analysis programs typically contain numerous software dependencies. For example, climate simulation output is often stored in the NetCDF [6] file format; NetCDF files are only readable using software libraries distributed by Unidata [7].

Opportunities for parallelization combined with a dependence on library routines create an interesting conundrum for climatologists. Clearly, climatologists could benefit from taking advantage of massively parallel supercomputing infrastructure to run their data analysis programs, but the program execution environment provided by any given supercomputing facility is typically unique to the given facility or machine architecture. The execution environment is likely to lack or provide the wrong version of an essential library, rendering execution of analysis code difficult. Thus, to move their analysis code to new supercomputing environments, climatologists face an unfortunate amount of overhead. If the supercomputing environment does not provide the proper libraries, users will have to install them by hand, which is tedious and often involves learning more about the underlying supercomputing system than the user would like to; for example, where on the file system the libraries should be installed, or how libraries are linked in to the analysis code.

Hence, the motivation for a self-contained, parallelizable method of packaging climate analysis code for execution on supercomputer infrastructure is clear. Utilization of virtualization-based cloud computing infrastructure [8, 9] is a natural solution to this problem: Virtualization technology provides a desirable configure-once, run-anywhere environment and the cloud computing paradigm can provide massive parallelism. What we need to provide a set of tools to coordinate the virtual machines and to transfer the necessary data files between the virtual machines and the storage systems.

The most popular virtualization-based cloud computing infrastructure is currently Amazon's EC2 [10]; in this paper, we used an open-source implementation of the EC2 API called Eucalyptus [11, 12]. The abstraction provided by such systems, while requiring a fair amount of overhead, is simple and attractive to programmers. Users create virtual machines, customizing them with whatever operating systems and software they like, and the cloud infrastructure provides a processor and RAM on which to run these virtual machines. Hence, users can replicate in their virtual machines the environment required by their code, allowing them to run their code on any system that can run their virtual machine. This would allow climatologists to package their analysis code into a format that can be run on a myriad of different systems and architectures without any need for reconfiguration when moving to new computational platforms.

We explored methods of using virtualization-based cloud computing resources to perform massively parallelized climate analysis. We built virtual machines to parallelize a program called TSTORMS [2], which finds tropical storms in climate simulation output data. Section 2 gives an overview of tropical storms and TSTORMS. We ran our virtual machine using two systems: the Eucalyptus cluster on the Magellan Scientific Cloud [13] at Argonne National Laboratory, and the Grid Laboratory of Wisconsin (GLOW) at the University of Wisconsin, Madison [14]. We used two different techniques to network and to coordinate virtual machine instances to perform data-driven parallel analysis of climate simulation output data; these techniques are outlined in Section 3. We compared the results from VM based runs with similar results from MPI based runs at NERSC [17] in Section 4, followed by conclusion in Section 5.

2. Tropical Storms and TSTORMS

Tropical storms are one of the most damaging climate events in terms of both monetary destruction and loss of human life [3]. In recent years, researchers have collected data demonstrating clear increases in tropical storm frequency and intensity [1, 2]. Groups of climatologists are using simulations to understand why this is happening and whether it will worsen. They devise models and run them first on recent past to establish their validity. After validating a model against recent observations, they run it far into the future to see how tropical storm characteristics will change. Often, the prognosis is grim: Using the fvCAM2.2 climate model, Wehner et al [3] demonstrated the likelihood of an increase in tropical storm frequency between the 1990s and the 2090s.

To find tropical storms in climate simulation output, Wehner et al used a program called TSTORMS, written by Tom Knutson's group at the Geophysical Fluid Dynamics Library [2]. The TSTORMS code is representative of many such climate analysis programs: It is very computationally intensive, requiring numerical min/max searches over splines generated from several large multidimensional arrays. It contains a number of library dependencies, and it offers opportunities for data-driven parallelism.

The TSTORMS code takes as an argument a NetCDF file containing climate simulation data and finds points in space and time satisfying the following conditions [2]:

1. A local relative vorticity maximum at 850 hPa exceeds $1.6*10^{-4}$ s^{-1}. Vorticity is the curl of wind velocity, and s is time in seconds.

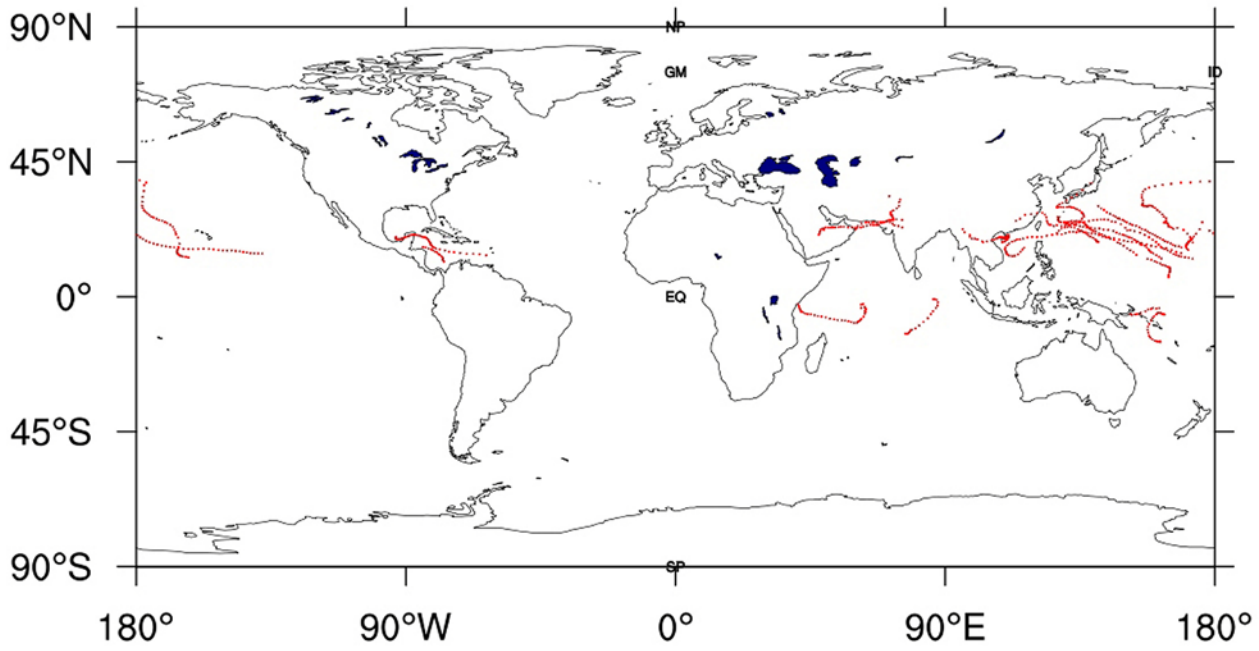

Figure 1: Simulated tropical storms, September 1993, from fvCAM2.2 simulation encompassing 1979-1993.

2. The surface pressure increases by at least 4 hPa from the storm center within a radius of 5 degrees. The closest local minimum in sea level pressure, within a distance of 2 degrees latitude or longitude from the vorticity maximum, is defined as the center of the storm.
3. The distance of the warm-core center from the storm center does not exceed 2 degrees. The temperature decreases by at least 0.8 degrees Celsius in all directions from the warm-core center within a distance of 5 degrees. The closest local maximum in temperature averaged between 300 and 500 hPa is defined as the center of the warm core.

Figure 1 was produced by TSTORMS running on our virtual machine on Magellan. This particular plot, which shows tropical cyclones in the year 1993 in a simulation of 1979-1993, would be useful to climatologists in establishing the validity of the climate model that produced it. They can compare the characteristics of a plot such as this with observation data to show that the underlying climate model is accurate.

TSTORMS is a single thread sequential program; running on a single processor, analysis of 500GB of climate simulation output can take several days. Parallelization is possible by running multiple TSTORMS processes simultaneously; however, doing this on traditional batch processing-based supercomputing systems can be difficult. This program requires a number of different support libraries. It also needs a large number of input files and produces a large number of output files. Finding the disk space for the 500GB of test data does not seem to be a challenging task, however, much larger datasets are expected in the near future. It is highly desirable to automate the data management tasks such as distributing the data files among the parallel processors and moving the data files to and from the storage systems.

We used virtualization-based cloud computing to parallelize this code, using multiple cloud nodes to produce data analysis output much more quickly than is possible on traditional workstations. Virtualization and cloud computing can provide both of the properties we desire: Support for massive parallelism as well as a self-contained environment in which climate analysis code can be run on a variety of platforms and architectures without reconfiguration.

3. Virtual Machine Coordination

Using virtual machines on cloud clusters for parallel data analysis requires coordination among virtual machine instances. In our case, virtual machines need to split up and analyze a large repository of climate data files. Our virtual machine contains a list of URLs to these data files; instances of the virtual machine must be equipped with some method of deciding which files each VM instance will process so that no redundant analysis is performed. This coordination is subject to a number of constraints introduced by the properties of cloud-based virtualization platforms. These constraints, though far from insurmountable, necessitate parallelization architectures somewhat different from the MPI-like paradigms many programmers are familiar with. We identified the following constraints:

- At launch, no virtual machine instance knows the location of or how to address other virtual machine

instances. Instances can use broadcast packets or IP/port scanning to find each other, and once they do, can open TCP connections with each other. Packets could be dropped between instances; however, virtual machine instances running on a cloud cluster are typically topologically close enough, i.e. there are very few routers between each instance, that packet transmission success rates are very good.

- In practice, it is difficult to control exactly how many virtual machines instances are successfully launched. A user requesting 40 instances might, for example, only receive 36. Not all cloud clusters share this property, but it our tests we hardly ever get all instances we have requested.
- Virtual machine instances launch at varying times: If a user makes a request for 20 VM instances, the first instance might start a half hour before the last.

Using MPI, process coordination for data-driven parallelism comes easier: At launch, each process can learn how many other processes exist as well as its "rank", a unique ID between 0 and the number of processes. Using this information, each process can compute a logical split of the input data without ever having to contact another process, as shown in Figure 2. Virtual machines on cloud clusters do not share this case. They do not know how many other virtual machine instances exist, since instances launch sporadically and usually in non-deterministically fewer numbers than requested; even if they did, there would be no easy way for an instance to determine its position in an ordering of instances, which MPI provides with process rank. Hence, we must devise a method for virtual machines to coordinate their data analysis. We explored two possibilities for virtual machine coordination: Coordination through leader election, and coordination through remote services.

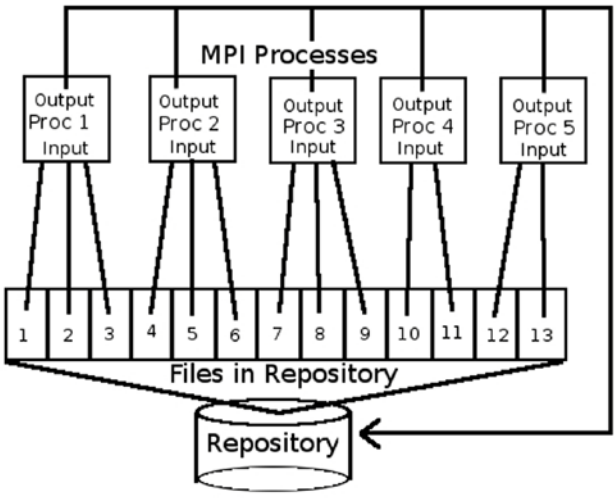

Figure 2: Sample design of MPI-based data analysis

3.1 Coordination Using Distributed Leader Election

A reasonable method of virtual machine coordination is to elect one virtual machine instance as a leader at launch time to track job status and coordinate virtual machine instances. In our case, the leader maintains a synchronized queue of URLs of input files from which all other VM instances pull one URL at a time. The main advantage of leader election is that the job is self-contained: a user can launch as many instances as she would like, and does not have to perform any further tasks, such as setting up a server for the VMs to contact.

The technical details of our leader election-based coordination scheme, as in Figure 3, are as follows: When our virtual machines launch, they run a script, written in Python 2.6, using their initialization file. This script first runs a distributed leader election algorithm (see section 3.2); after this algorithm, we are guaranteed that all running instances have decided on a single leader and know how to address it. Late-starting instances will be contacted by the leader or another work, and informed of the correct leader; this is built into the leader election scheme. The leader initializes a synchronized queue of remote URLs to climate simulation output files, as we store these files in a remote repository, and sets up an XML-RPC (xml-based remote procedure call) wrapper around this synchronized queue. The workers use Python's "ServerProxy" abstraction to contact this XML-RPC library; each call returns the next URL in the leader's synchronized queue. Each time a worker receives a URL, it uses GridFTP [15] to copy the file to its local file system, runs the TSTORMS code on it, and uses GridFTP to stage the results out to a remote directory. As a small optimization, we used multiple threads to perform data staging and analysis at the same time, since staging data in and out is primarily I/O-bound, whereas the TSTORMS code is decidedly CPU-bound. Once the leader's synchronized queue is empty, the workers receive "null" values from calls to the XML-RPC library; when this happens, they shut themselves down.

We ran our leader election-based virtual machine on the Magellan Scientific Cloud.

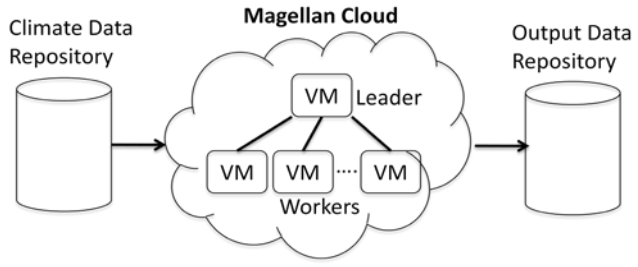

Figure 3: Design of leader election-based virtual Machine

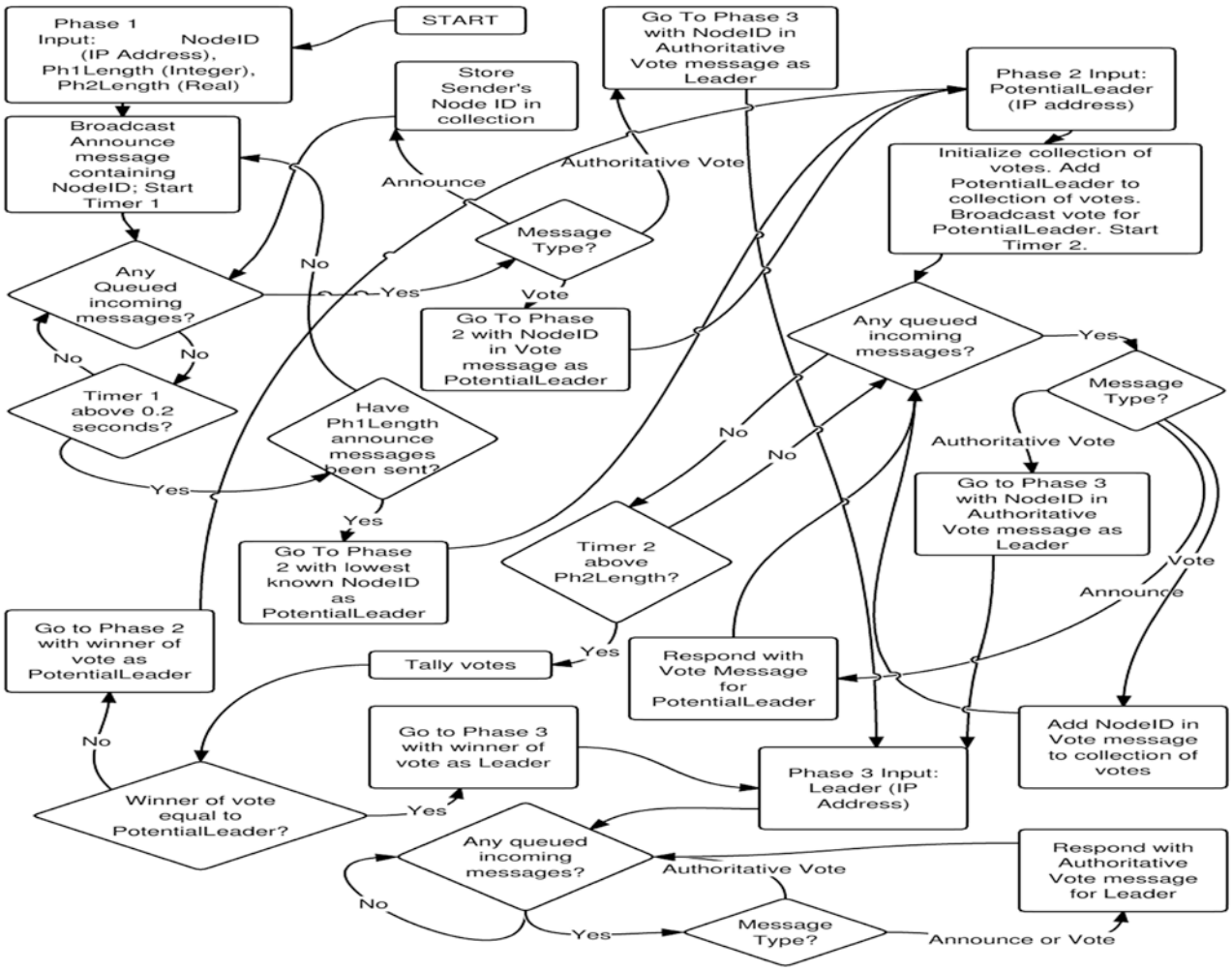

Figure 4: Leader election algorithm

3.2 Leader Election Algorithm

We implemented a custom leader election algorithm designed to work under the constraints of a virtualized cloud environment. Since no instance initially knows the address of any other instances, a "discovery" phase must be built into the election. Instances launch in unpredictable, non-deterministic patterns; hence, the algorithm must ensure late-starting instances decide upon the correct leader. Instances can easily learn their own internal IP, so we used this as a "node ID" for the election algorithm.

Our algorithm is based on the classic Bully algorithm [16], in which nodes attempt to choose the highest or lowest node ID as the leader. Our algorithm offers probabilistically correct performance with parameters that can be configured based on the packet transmission success rates of the underlying network.

Our algorithm uses 3 phases. The first phase is for discovering other nodes; in this phase, nodes send and receive broadcast messages to find the addresses of other nodes. In the second phase, nodes vote for and elect a leader. In phase 3, nodes notify late-starting nodes of who the elected leader is.

Nodes send 3 types of messages to each other:

- Announce message: For announcing your presence to other nodes.
- Vote message: For voting for a particular node to be leader. Contains the ID of the leader voted for. In our case, the ID is the node's IP address.
- Authoritative Vote message: Sent during phase 3, when a node is certain of who the leader will be. It contains the ID of the elected leader. A node receiving an Authoritative Vote message during phase 1 or 2 will skip immediately to phase 3, using the leader indicated by the Authoritative Vote.

Each node runs this algorithm upon startup. Figure 4 is a flowchart representing the steps in this algorithm. Notes on the flowchart:

- Each phase of the algorithm requires input. Phase 1 takes the NodeID of the node running the algorithm, as

well as two parameters (PH1Length and PH2Length) that determine how tolerant of packet loss the algorithm will be. Larger values of PH1Length and PH2Length cause the algorithm to take longer and be more accurate. Phase 2 takes in the NodeID of a potential leader to be voted for. Phase 3 takes in the NodeID of the elected leader. For example, one step on the flowchart says "Enter Phase 2 with NodeID in Vote message as PotentialLeader". This means to get the NodeID from the given Vote Packet, and use this as the input to Phase 2. Then when Phase 2 references PotentialLeader, it is referencing the NodeID from the Vote message.

- Some state is stored. In phases 1 and 2, timers are set; additionally, in phase 2, a collection of votes is stored. A box that says "Timer X above Y seconds?" means "if Timer X has been running for more than Y seconds, take the 'yes' branch; else, take the 'no' branch". The timers and vote collection are reset when revisiting the step that initializes them. So, for example, Timer 1 is reset every time the algorithm reaches the step that says "Start Timer 1".

We ran stress tests on this algorithm using simulated 90-95% packet transmission success rates, parameter PH1Length, as shown in Figure 4, equal to 20 announce packets, PH2Length equal to 10 seconds, varying numbers of nodes, and varying amounts of "stagger" between the launch time of each node. Over hundreds of test runs, we had a 100% algorithm success rate, where "success" is defined as "every node chooses the same leader". We were also able to achieve 100% success rates with packet loss rates as high as 50% by setting parameters X and Y to be suitably large. With packet loss rates much higher than 50%, we were unable to achieve 100% algorithm success rate. Regardless, we achieved 100% success rate over many dozens of runs on Magellan resources, since packet transfer success rates are generally very good between virtual machine instances on a cloud cluster.

Because node reliability becomes an issue as parallelism scales, most leader election schemes implement some level of fault tolerance. This was unnecessary in our particular case. If the leader election fails, which is very unlikely based on our test results, the submitter of the job can just kill all of his/her instances and restart them, wasting a relatively small amount of compute resources and very little of the submitter's time, since the leader election is run before anything else. Once a leader is elected, there is only one essential node, the leader, and the chances of a particular node failing is always very low, even when scaling causes the chance of at least one node failing to grow large. Hence, we decided to avoid implementation of complex fault-tolerance procedures and simply restart the job in the extremely rare case of leader node failure. Fault-tolerance for non-leader nodes is much simpler, of course. For example, we implemented a scheme where worker nodes send "heartbeats" to the leader, allowing the leader to infer when a node has failed and subsequently re-release its input URLs to other workers.

3.3 Coordination through a Remote Service

Another reasonable method of virtual machine coordination is to have the virtual machine instances connect to a remote service. This remote service does everything that an elected leader would do: It maintains a synchronized queue of URLs to input files and an XML-RPC server for connecting to the synchronized queue. This method is somewhat easier to implement, and it does not require leader election or node discovery. However, it requires maintenance of a remote service for virtual machine instances to connect to. Before any virtual machine can run, the remote service needs to be initialized for VM instances to connect to.

The technical details of this scheme as in Figure 5 are as follows: At launch, each virtual machine fetches, from a pre-defined location, a file containing a service IP/port to connect to. This service should be initialized before running instances; our implementation of this remote service takes a list of climate simulation output URLs as an argument. The service initializes a synchronized queue of input file URLs and an XML-RPC wrapper around the queue, just like in the leader election scheme. The virtual machines use Python's "ServerProxy" abstraction to connect to the remote server and pull file URLs. Each time a virtual machine pulls a URL, it uses GridFTP to stage in the data, analyzes the data using TSTORMS, and stages the results out to a remote directory. Once all URLs have been pulled from the queue, "null" values are returned and the virtual machines shut themselves down. We ran our remote service-based virtual machine on GLOW.

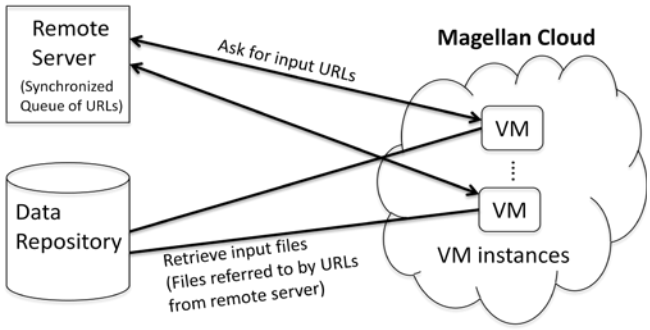

Figure 5: Design of remote server-based virtual machine

4. RESULTS

We used our virtual machine to run TSTORMS on a remote 500GB repository of climate simulation data using varying numbers of virtual machine instances on Magellan and GLOW. We also ran TSTORMS on the same dataset using MPI on the Carver cluster at NERSC [17]. Nodes on Carver and Magellan are very similar; each node on each system contains dual quad-core Intel Nehalem 2.66GHz processors and 24GB RAM. Hence, Carver is a comparable system to Magellan. The GLOW nodes we used utilized Xeon

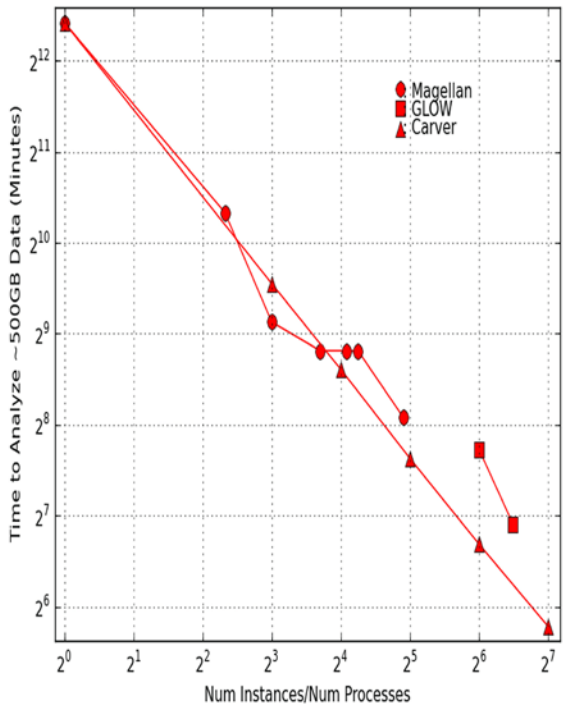

Figure 6: Analysis timing results

2.66GHz and 3.2GHz processors, and had enough RAM for TSTORMS to execute without using virtual memory, so our VM on GLOW had compute resources comparable to, though not exactly the same as, instances on Magellan and processes on Carver. We used Magellan for tests involving fewer instances, and GLOW for tests involving many instances.

Our climate simulation data was stored on GPFS file system at NERSC, and Carver had somewhat of a speed advantage over our virtual machines since data could be accessed through a local file system rather than needing to be sent across a network. Virtualization overhead put our virtual machines at a further disadvantage compared to Carver. Regardless of this, we found that both methods offer similar performance, with VM based analysis on Magellan actually performing better than MPI based analysis on Carver in one test: Analyzing our 500GB repository on Carver using 8 processes took 3 hours longer than on Magellan using 8 virtual machine instances (~12.5 vs. ~9.5 hours). Using 90 instances on GLOW, we were able to produce analysis output in ~2 hours. This is a conveniently short amount of time for a scientist to wait for analysis output, and it is comparable to analysis performance on Carver. Figure 6 is a log-log plot of all timing data we collected.

Virtual machines were substantially more unpredictable than processes on Carver with respect to total analysis time as a function of number of instances/processes. On Carver, doubling the amount of processes halves total analysis time; using virtual machines on a cloud cluster, this property holds only approximately. Since virtual machine instances can have different starting times, whereas processes in MPI start almost at the same time, this is to be expected. However, we saw a little more eccentricity than we would expect. We concluded that this is largely a consequence of staging in data over a shared network; we observed that on both Magellan and GLOW our virtual machines ran somewhat faster late at night and on weekends, when there was less competition for network resources. The anomalous 8-instance test we ran on Magellan was started on a Friday night, so competition for both network bandwidth and cloud nodes would have been relatively low. Further research will include using FUSE [18] to mount data onto our VM, which will allow us to avoid transferring data over a network.

Using 30 virtual machines, we were able to analyze the 500GB dataset in ~4.5 hours. Using a single-processor workstation, analysis of the same dataset can take several days; assuming the workstation has similar computational power to a single virtual machine instance on Magellan or MPI process on Carver, analysis would take roughly 90-95 hours. Hence, using 30 virtual machine instances, we reduced total analysis time by a factor of ~21. Ideally, the reduction factor here would be 30. We expect that by FUSE-mounting data to the VM, instead of transferring it over a network, we can reduce total analysis time even further.

5. CONCLUSIONS

In light of our results, we concluded parallel virtualization to be a viable paradigm for large-scale data analysis. Parallel virtualization offers an attractive environment in which analysis programs can be configured once and run anywhere with configurable, and potentially massive, levels of parallelism and efficiency comparable to a traditional batch-based supercomputing system. This is a distinct advantage over both methods currently in use: Traditional supercomputing systems do not offer the configure once, run anywhere property, while personal workstations cannot offer the massive levels of parallelism climatologists need to analyze the vast amounts of data their simulations produce.

The usefulness of parallel virtualization is not limited to climate simulation data analysis. Any scientist with parallelizable data analysis requirements and reluctance to migrate analysis code to supercomputing facilities due to program execution environment concerns can take advantage of virtualization-based cloud computing to produce data analysis output quickly and efficiently without the usual difficulties that arise in porting analysis code to new environments.

The additional work required is to provide a set of tools for coordinating the virtual machines and facilitate data movements. In this work, we experimented with two different approaches for coordination: through leader election and through an external service. We handle the

data movements through a common Grid tool. From our experience, we believe that the job coordination and data movements can be built into a management system as a part of a cloud computing ecosystem. We plan to extend our work to produce such a system.

ACKNOWLEDGMENTS

This work was funded, and used resources of the National Energy Research Scientific Computing Center, by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under contracts DE-AC02-05CH11231. This work also used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357, funded through the American Recovery and Reinvestment Act of 2009, and resources provided by the Open Science Grid, which is supported by the National Science Foundation and the U.S. Department of Energy's Office of Science. We would like to thank Shane Canon, Jason Hick and David Skinner at National Energy Research Scientific Computing Center and the Condor Team at University of Wisconsin at Madison and the OSG Engage Team for their support on our experiments.

REFERENCES

[1] P. J. Webster, G. J. Holland, J. A. Curry, H.R. Chang. Changes in Tropical Cyclone Number, Duration, and Intensity in a Warming Environment. Science 2005 vol. 309, no. 5742, pp. 1844-1846.

[2] T. R. Knutson, J. J. Sirutis, S. T. Garner, I. M. Held, R. E. Tuleya. Simulation of the Recent Multidecadal Increase of Atlantic Hurricane Activity Using an 18-km-Grid Regional Model. Bulletin of the American Meteorological Society 2007 88:10, 1549-1565.

[3] M. F. Wehner, G. Bala, P. Duffy, A. A. Mirin, and R. Romano. Towards Direct Simulation of Future Tropical Cyclone Statistics in a High-Resolution Global Atmospheric Model. Advances in Meteorology. Volume 2010 (2010), Article ID 915303.

[4] J. Stoer and R. Bulirsch. Introduction to Numerical Analysis. New York: Springer-Verlag, 1980.

[5] Lorenz, Edward N. "Deterministic Nonperiodic Flow". Journal of the Atmospheric Sciences 20 (2): 130–141, March 1963.

[6] NetCDF. [Online] http://www.unidata.ucar.edu/software/netcdf/

[7] Unidata. [Online] http://www.unidata.ucar.edu/

[8] M. Cusumano. Cloud computing and SaaS as new computing platforms. Comm. ACM 53, 4 (Apr. 2010), 27-29.

[9] R. Buyya, C. S. Yeo, and S. Venugopal. Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. In Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications (HPCC 2008, IEEE CS Press, Los Alamitos, CA, USA), Sept. 25-27, 2008.

[10] Amazon EC2 Case Studies. [Online] http://aws.amazon.com/solutions/case-studies/

[11] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus Open-Source Cloud-Computing System. In Proceedings of the 2009 9th IEEE/ACM international Symposium on Cluster Computing and the Grid (May 18 - 21, 2009). CCGRID. IEEE Computer Society, Washington, DC, 124-131.

[12] Eucalyptus. [Online] http://www.eucalyptus.com/

[13] Magellan Scientific Cloud. [Online] http://magellan.alcf.anl.gov/

[14] UW-HEP Computing. [Online] http://www.hep.wisc.edu/computing/

[15] Allcock, W., Bresnahan, J., Kettimuthu, R., Link, M., Dumitrescu, C., Raicu, I., and Foster, I., "The Globus Striped GridFTP Framework and Server" In Proceedings of the 2005 ACM/IEEE Conference on Supercomputing (November 12 - 18, 2005).

[16] Hector Garcia-Molina, Elections in a Distributed Computing System, IEEE Transactions on Computers, Vol. C-31, No. 1, January (1982) 48-59.

[17] Carver at NERSC. [Online] http://www.nersc.gov/nusers/systems/carver/

[18] FUSE: File system in user space. [Online] http://fuse.sourceforge.net/

[19] Giorgi, F. and Mearns, L.O. “Calculation of average, uncertainty range and reliability of regional climate changes from AOGCM simulations via the ‘Reliability Ensemble Averaging’ (REA) method.” *J. Climate* 15, 1141-1158. 2002.

[20] Richard L. Smith, Claudia Tebaldi, Doug Nychka, Linda O. Mearns. Journal of the American Statistical Association. March 1, 2009, 104(485): 97-116. doi:10.1198/jasa.2009.0007.