Future Generation Computer Systems 25 (2009) 346–350

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Grid-based deployment and performance measurement of the Weather Research & Forecasting model

Jan Ploski [a,*], Guido Scherp [a], Thomas I. Petroliagis [c], Otto Büchner [d], Wilhelm Hasselbring [a,b]

[a] OFFIS, Escherweg 2, 26121 Oldenburg, Germany

[b] University of Oldenburg, Software Engineering Group, 26111 Oldenburg, Germany

[c] ForWind – Center for Wind Energy Research, Marie-Curie-Str. 1, 26129 Oldenburg, Germany

[d] Jülich Supercomputing Centre, Forschungszentrum Jülich, Wilhelm-Johnen-Str. 1, 52425 Jülich, Germany

ARTICLE INFO

Article history:
Received 30 November 2007
Received in revised form
14 April 2008
Accepted 14 May 2008
Available online 18 May 2008

Keywords:
Grid service deployment
Performance measurement
Quality of service

ABSTRACT

A system developed for benchmarking multi-processor Numerical Weather Prediction applications deployed on D-Grid resources is presented. The system currently serves to perform functional and non-functional software tests of the Weather Research and Forecasting (WRF) model used in the project WISENT. The resulting performance data and the input configurations are automatically published through a web portal to enable third-party comparisons with other, existing WRF deployments. The developed system relies on a self-contained, domain-independent software module for running MPI and other multi-processor Grid (Globus Toolkit 4) jobs that require a user-defined synchronized initialization and cleanup phase.

1. Introduction

In the national research project WISENT [1] computer scientists, physicists and meteorologists work together in an effort to optimize the cooperation of scientific organizations in the field of energy meteorology. The main objective of energy meteorology is obtaining the information needed to characterize the fluctuating generation of solar and wind energy. One of the main research tools in energy meteorology are Numerical Weather Prediction (NWP) models. To cope with the heavy computational and storage requirements of numerical weather prediction, WISENT relies on the distributed computing resources provided by the German Grid [2].

This paper begins with an introduction to the Weather Research & Forecasting (WRF) model and its run-time characteristics (Sections 2 and 3). Section 4 addresses the challenges involved in executing multi-processor Grid jobs. The proposed approach for submitting multi-processor Globus jobs with an initialization and cleanup phase, although motivated by WRF, could be transferred readily to other Grid projects, even with non-MPI applications. Section 5 covers the WISENT Performance Benchmarking System, which successfully employs the proposed job submission approach. Section 6 discusses performance measurements obtained within the system. Section 7 concludes and outlines future work.

2. Weather Research & Forecasting model

The Weather Research and Forecasting (WRF) model [3] is a mesoscale NWP model, suitable for research and operational forecasting. The currently installed version makes use of the ARW (Advanced Research WRF) solver, which is composed of several initialization programs for idealized and real-data simulations, and a numerical integration program.

The WRF is fully compressible, Euler non-hydrostatic with a run-time hydrostatic option available and it is conservative for scalar variables. Its prognostic variables are: the velocity components u and v in Cartesian coordinates, the vertical velocity w, the perturbation potential temperature, the perturbation geopotential, and the perturbation surface pressure of dry air. Optionally, the turbulent kinetic energy and any number of scalars such as water vapor mixing ratio, rain/snow mixing ratio, and cloud water/ice mixing ratio can be also predicted.

From a more technical viewpoint, WRF is a complex software system which has been developed by multiple research institutions and designed for running on a wide range of hardware from individual PCs (for educational use) through Linux clusters to dedicated supercomputers (for research and operational forecasting).

* Corresponding author.
E-mail address: jan.ploski@offis.de (J. Ploski).

0167-739X/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2008.05.003

3. Compile and run-time requirements of WRF

Like most of today's scientific applications, WRF does not provide any specific support for execution in a heterogenous Grid environment [4]. In particular, the compiled binary code is not portable across clusters. Thus, the effort of deploying WRF "on the Grid" is equivalent to deploying it multiple times, in multiple independent clusters. Unfortunately, the "compile once, run anywhere" or "interpreted script" approaches suitable for simple Grid applications do not apply to WRF due to its compile-time and run-time dependencies on libraries that implement the MPI standard. Similarly, if the OpenMP parallelization option is used, it is necessary to compile WRF with one of the supported optimizing compilers and use a run-time support library provided by the (commercial) compiler vendor. Additionally, the supporting NetCDF library and, in some cases, parallel libraries such as MPICH, must be built with the same compiler as WRF itself. In research contexts, user-defined modules also exist that must be compiled together with the rest of the WRF code. All these requirements call for careful, coordinated configuration management to support convenient access of multiple users to multiple model variants deployed in multiple clusters. Uploading and compiling WRF as part of each job is not viable because of interactive configuration steps, the amount of required master data (over 10 GB if preprocessing support is included), and long build time (20 min on current hardware).

Before running a compiled version of WRF, a working directory with the following items must be set up:

- a namelist.input file with various configuration parameters,
- a number of specifically named master data files (or appropriate symbolic links to a central installation directory),
- input file set referenced from namelist.input.

The input file set for the working directory is pre-defined and specific to a benchmark case. For real (non-benchmark) model runs, these items would have to be supplied along with the user's job. They originate from an earlier (preprocessing) stage of the WRF workflow, which we do not discuss in this paper. The master data files, on the other hand, are a part of the installed variant of WRF and do not usually need user-specific modifications.

If compiled to use MPI, the WRF executable is started using a system-specific MPI startup program (such as mpirun, mpiexec, etc.), which spawns one or more processes per allocated execution node. If compiled to use OpenMP, each WRF process spawns multiple threads. The parallelism options can be combined as well.

A single model run produces a variable number of output files in the NetCDF format, depending on the number of domains defined in namelist.input and other parameters. When running benchmarks, we are less interested in the actual output files (whose expected contents are known) and more in the measured execution times.

4. Running multiprocessor jobs using Globus Toolkit 4

The specific run-time requirements of the WRF can be rephrased into a general use case: submit jobs comprised of the following steps (Fig. 1):

(1) Prepare a working directory.
(2) Run an MPI-based computation on multiple processors.
(3) Before stage-out, post-process files produced by the MPI executable.

While this use case is not exotic, it is impossible for end users without sophisticated programming skills to implement it using Globus Toolkit 4. Specifically, the WS GRAM service supports jobs of types *single*, *multiple*, and *mpi*, none of which satisfy the above requirements:

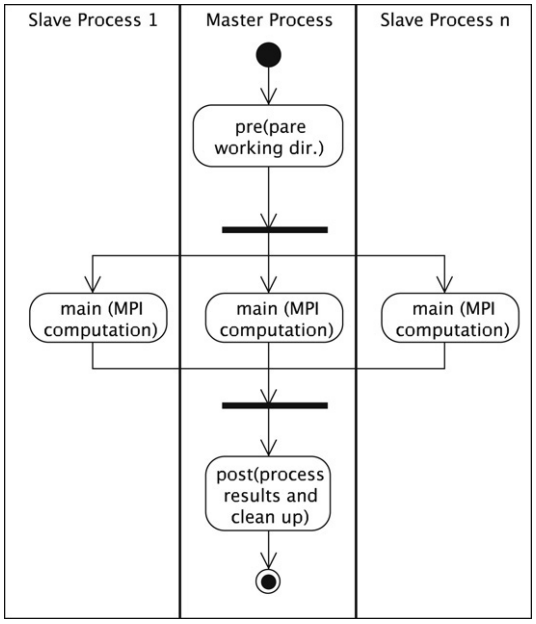

Fig. 1. A multi-processor job with a synchronized initialization and cleanup phase.

(1) Jobs of the type *single* are handled by allocating one or more processors through the resource manager and starting a user-specified executable on the first allocated processor. This job type is suitable for the initialization and cleanup phases, but not for the main MPI-based computation. In theory, it is possible to allocate several nodes using the *hostCount* parameter and invoke the cluster-specific MPI startup program from a user-provided script. In practice, this solution falls short due to both design and implementation constraints. First, it breaks abstraction by requiring the user to deal with the implementation details of the MPI startup mechanism — and accordingly to keep track of the different MPI implementations across Grid sites. Second, it fails because the job execution environment does not provide sufficient information to control the MPI startup program. Specifically, it is difficult to determine from a job which particular nodes were allocated to it.
(2) Jobs of the type *multiple* are handled by allocating multiple processors through the resource manager and starting a user-specified executable on each of the allocated processors. This job type is unsuitable for MPI for the same reasons provided above.
(3) Jobs of the type *mpi* are handled similarly to those of type *multiple*. However, the user-specified executable is started using the MPI startup program. This is perfect for running an executable compiled with MPI support, but it does not offer an option for the required intialization/cleanup steps—unless one hardcodes them into the executable itself.

The most obvious solution for the problem described would be to submit and coordinate three Globus jobs, the first one of type *single* to set up the working directory, then an *mpi* job to run the actual application, and finally another *single* job to perform cleanup and stage-out. Obviously, this solution, which requires client-side coordination and separates the three phases that logically form a single, atomic job, does not appeal to users. Likewise, treating the three steps as elements of a "Grid workflow" which requires a heavy-weight execution engine as support would not be appropriate.

For these reasons, we developed an alternate solution which is quite portable, lightweight and easily explained to end users with just basic scripting skills. The approach consists of submitting a job of type *mpi*, whose executable is not the actual MPI executable, but a script responsible for process synchronization,

including the single-processor initialization, the multi-processor MPI run, and the single-processor cleanup phase. This user-defined script, implemented in the universally available Perl [5] language, is launched by the MPI startup program on each allocated processor. It implements three procedures: *pre*, *main* and *post*, corresponding to the execution phases shown in Fig. 1. The procedures are invoked on user's behalf at appropriate times by the Perl module `MultiJob`, developed in WISENT for this purpose. Our module relies on a locking mechanism based on atomic hard linking of files across NFS [6].

Note that the execution mechanism of job described does not require modifications or extensions to the Globus middleware. The basic requirement of the described job execution mechanism is that the home directory is shared across the worker nodes in a cluster. This is the case for the D-Grid reference installation and recommended for D-Grid sites [7]. Another limitation, which did not become evident until testing, is that the MPI startup program must not proactively kill its children right after the final MPI message exchange. If this happens, as in the JuGGLE cluster [8], then the *post* subroutine will not run and errors during file stage-out will occur. Some MPI startup programs employ the kill mechanism to ensure that their distributed child processes are terminated in all cases. A possible solution (which, however, is specific to the resource manager and has to be implemented by the MPI tools vendor) is to use the PBS APIs rather than rsh/ssh to create child processes, and then to rely on the resource manager or a periodically running script to terminate run-away processes.

5. The WRF benchmarking system

Even though WRF is a widely deployed model, little data exist concerning the recommended software and hardware configurations. Users have to learn by guessing or stay with an initially compiled, suboptimal (but working) configuration. Likewise, because WRF supports both MPI and OpenMP, the IT decision makers responsible for purchasing hardware miss recommendations on whether to invest in a fast network interconnect, prefer more nodes with a slower network, or maybe fewer, more powerful multi-processor nodes. Benchmark results provided by WRF developers concentrate on high-end supercomputing hardware. The benchmarking system developed in WISENT provides better guidance for users of Linux clusters, similar to those found at many D-Grid sites.

Another motivation for developing a WRF benchmarking system is that it is capable of executing typical test cases, thus establishing the appropriateness of the available D-Grid resources for our application. Benchmark cases that obtain expected results with acceptable performance clear the way to running more complicated, real-world models in D-Grid. Users cannot be compelled to use the Grid by scalability arguments alone. They expect that Grid jobs execute at least as fast and reliably as local jobs and are reluctant to move their jobs to the Grid if tradeoffs in service quality are involved.

Fig. 2 illustrates the overall architecture and the main workflow steps involved in using the WRF Benchmarking System. The operator starts a backend script to request the execution of multiple benchmark run jobs (1). Each job is addressed explicitly to a target Grid site, references a desired benchmark case, the model variant to use (e.g., a version of WRF compiled with MVAPICH) and the number of processors. The information about the availability of model variants and benchmark cases at Grid sites is retrieved from the configuration database (2). The job is submitted to a central Condor-G [9] queue, from which it is forwarded using WS GRAM to the target Globus Toolkit 4 site (3). The job scripts, including the supporting module described in Section 4, are staged in and the benchmark case is run at the target site. After the execution

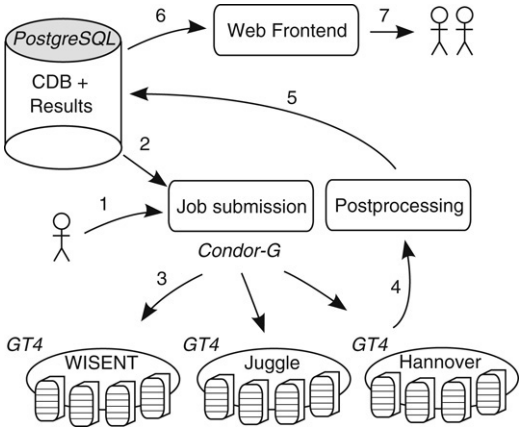

Fig. 2. The WRF benchmarking system developed in WISENT.

completes, an XML file describing the individual run is staged out (4). A periodic post-processing job inserts data from the XML file into the results database (5). The data are retrieved upon an external user's request by the web frontend (6) and presented either as a raw CSV file or in form of graphical plots. The user may also view and download the configuration details of the employed WRF variants and benchmarks. In this way, our optimal configurations can spread to benefit researchers and forecasters working with WRF outside of the WISENT project.

6. Performance measurements

This section presents sample results obtained by running the WRF benchmarks within our system. Additional data (updated as new benchmarks and model variants are added and executed) as well as past results for the existing benchmark cases are available online through the WISENT web site [10].

The key characteristics of an individual benchmark case that obviously affect WRF execution times are

- Horizontal grid sizes — derived from the extents of the geographical region over which the simulation is performed and the requested spatial resolution. For each of the grid points, the value of every WRF variable (such as temperature, wind velocity) is simulated over time. The bigger the region and resolution, the more points are needed in the geographical grid, and the higher the computational requirements.
- Time step — the interval between the subsequent steps of the simulation; the smaller the time step, the higher the computational requirements.
- Number of (nested) domains — WRF supports nested runs in which a geographical region is subdivided into subregions with a higher spatial resolution and a smaller time step.

What is less evident is the influence of some additional parameters such as the different possible combinations of physics options, the choice of the parallelization mechanism (MPI, OpenMP or hybrid), the particular implementation of that mechanism, and hardware. Our benchmarks collect empirical data to improve the understanding of these relationships.

As an example, the results for the `conus12km_2001` case are shown in Fig. 3. This benchmark case was originally constructed by the WRF developers at NCAR. However, since the introduction of WRFv2.2 no performance results for this case have been published. This single-domain case covers the continental USA area with 12 km resolution (distance between adjacent grid points), forecasting 3 h from an original 48 h model run. The geographical grid size is $425 \times 300 \times 35$ points with a corresponding time step of 72 s.

The `conus12km_2001` case has been executed in Oldenburg and Jülich D-Grid clusters. WRF has been built with the same

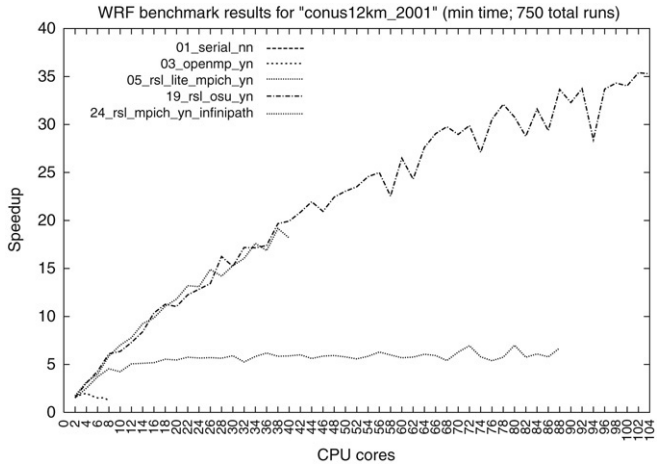

Fig. 3. Benchmark results for the conus12km_2001 case: `Serial` — a non-parallel run for baseline performance. `OpenMP` — shared-memory OpenMP parallelization with up to 8 2.2 GHz AMD Opteron processors. `MPICH` — distributed memory parallelization using MPICH over Gigabit Ethernet. `InfiniBand` — distributed memory parallelization using MVAPICH over InfiniBand [11] in Oldenburg. `InfiniPath` — distributed memory parallelization using MVAPICH over InfiniPath [12] in Jülich.

compiler, but with different parallelization mechanisms. The reported metric is the best speedup observed during a total of 750 runs. This metric, measured in multiple runs with the same number of processors performed at different times of day, provides a reliable impression of WRF's scalability, reducing the possibility of obtaining results skewed by other temporary activity in the cluster (such as other users' applications executing on the same nodes or accessing the common NFS-mounted file system).

The results for `conus12km_2001` reveal the following information, which may aid decisions about the number of processors required for real cases:

- OpenMP variants of WRF scale very poorly. This confirms anecdotal evidence from the WRF user forum. However, the speedups are so bad that a further investigation of the causes is needed.
- WRF's scalability clearly benefits from the reduced latency and increased bandwidth of the InfiniBand interconnect, compared to Gigabit Ethernet, which scales very poorly beyond 8 processors. Therefore, WRF users should seek to deploy their models in InfiniBand equipped clusters.
- There is no noticeable difference between the scalability of the InfiniBand and InfiniPath based solutions. InfiniPath's expected superior performance due to tighter hardware integration did not become apparent in the benchmark.
- The curves contain irregularities which cannot be explained by the change in the number of processors or by environmental factors (e.g., between 94 and 96 processors for model variant 19). Awareness of this characteristic is helpful to avoid a mistake of assuming that adding further processors would not improve performance upon observing a first "performance slip".

7. Conclusions and future work

We have discussed the inherent difficulties of running multi-processor jobs that require an initialization and cleanup phase using Globus Toolkit 4. While many Grid workflow description languages exist that can solve the described synchronization problem in principle, the scenario is common enough to be addressed more directly without specialized middleware. The presented approach can be also applied to reduce the overhead when submitting a large number of small jobs by packaging them into a few multi-processor jobs and scheduling at the target site.

Our goal was to collect application-specific performance measurements by taking advantage of the heterogeneity inherent in a Grid environment. We took the shortest route toward this target by developing our own benchmarking solution bound to the WRF. The collected data can serve to develop a run-time performance model for this particular application. Such a model would allow predictions of the execution time based on the parameters of a specific WRF job, which would serve both users and schedulers.

While it is appropriate for benchmarks to require the exact specification of a target execution site, site selection should ideally be automatic in case of real-world models run by end-users. To achieve this automation, we intend to employ a meta-scheduler that utilizes properties such as site configuration, the availability of input data and the number of free CPUs for target site selection.

Currently, we are evaluating two meta-schedulers: GridWay [13] and Condor-G [9]. GridWay is an open source Grid middleware which builds upon Globus Toolkit 4 and offers a workload manager for meta-scheduling. It is used in Grid infrastructures based on Globus Toolkit in which jobs can be submitted via the interfaces pre-WS-GRAM (Globus Toolkit 2) or WS-GRAM (Globus Toolkit 4). The Grid adapter Condor-G in its most basic configuration submits a job to an explicitly named Grid site. However, Condor's native ClassAds match-making mechanism [14] can be used to select a site dynamically according to user-defined criteria.

Acknowledgements

The first three authors' work is supported by the German Federal Ministry of Education and Research (BMBF) under grant No. 01C5968 as part of the D-Grid initiative.

References

[1] W. Hasselbring, et al., WISENT: e-science for energy meteorology, in: E-SCIENCE '06: Proc. 2nd IEEE Intl. Conf. on e-Science and Grid Computing, IEEE Comp. Soc., Washington, DC, USA, 2006, p. 93. URL http://wisent.d-grid.de.
[2] D-Grid, The German Grid Initiative. URL http://www.d-grid.de.
[3] W.C. Skamarock, et al., A description of the advanced research WRF version 2, NCAR Tech Note 468+STR, NCAR, 2005. URL http://box.mmm.ucar.edu/wrf/users/docs/arw_v2.pdf.
[4] J. Ploski, T. Petroliagis, D. Heinemann, T. Scheidsteger, W. Hasselbring, Grid-based modeling in "Wissensnetz Energiemeteorologie", in: Proc. DACH 2007 Meteorologentagung, CD, Deutsche Meteorologische Gesellschaft, 2007.
[5] L. Wall, T. Christiansen, J. Orwant, Programming Perl, 3rd edition, O'Reilly Media, Inc., 2000.
[6] R. Brown, File::NFSLock — Perl module to do NFS (or not) locking. URL http://search.cpan.org/ bbb/File-NFSLock-1.20.
[7] W. Gürich, et al., Betriebskonzept für die D-Grid-Infrastruktur, Tech. Rep., October 2007. URL http://www.d-grid.de/index.php?id=395.
[8] JuGGLE — Jülich German grid linux environment, last access: 2007-11-13. URL http://www.fz-juelich.de/jsc/service/juggle/.
[9] J. Frey, T. Tannenbaum, I. Foster, M. Livny, S. Tuecke, Condor-G: A computation management agent for multi-institutional Grids, in: Proc. 10th IEEE Symposium on High Performance Distributed Computing, 2001.
[10] WISENT, Online WRF benchmark results, last access: 2007-11-13. URL https://bi.offis.de/wisent/tiki-index.php?page=WRF-Benchmarks.
[11] O.I. Pentakalos, An introduction to the InfiniBand architecture, in: Proc. Intl. CMG Conference, Computer Measurement Group, 2002, pp. 163–165.
[12] L. Dickman, G. Lindahl, D. Olson, J. Rubin, J. Broughton, PathScale InfiniPath: A first look, in: Hot Interconnects, IEEE Comp. Soc., 2005, pp. 163–165.
[13] E. Hudo, R.S. Montero, I.M. LLorente, The GridWay framework for adaptive scheduling and execution on grids, Scientific International Journal for Parallel and Distributed Computing 6 (3) (2005) 1–8.
[14] M. Solomon, The ClassAd language reference manual, Tech. Rep., May 2004. URL http://www.cs.wisc.edu/condor/classad/.

Jan Ploski is a research staff member in the R&D Division Energy of the OFFIS Institute for Information Technology in Oldenburg, Germany and a member of the TrustSoft Graduate School on Trustworthy Software Systems. He received his degree in Information Systems from the Cologne University of Applied Sciences in 2003. His research interests include Grid computing, software fault diagnosis, and software design.

Guido Scherp is a research staff member in the R&D Division Energy of the OFFIS Institute for Information Technology in Oldenburg, Germany. He received his diploma in Computer Science from the University of Oldenburg in 2005. His research interests include security, resource brokering/scheduling and workflows in Grid infrastructures.

Thomas I. Petroliagis is a senior forecaster who worked several years in the development and operational application of various NWP systems. He received his Ph.D. in 2002 (Aristotelian University of Thessaloniki, Greece). Thomas has worked as the Chief Technical Advisor for an UNDP Project for the installation, operation & validation of an integrated NWP system for the Military Weather Center of the UAE (2003–2006). He had previously participated in the core development group of the Ensemble Prediction System at the ECMWF. He joined For-Wind at the beginning of 2007 to work in the field of wind energy.

Otto Büchner received his diploma in Mathematics from the Technical University of Braunschweig, Germany, in 1986. He is working as researcher and system administrator at the Jülich Supercomputing Centre, Research Centre Jülich. His current research interests include storage systems, cluster systems and Grid computing.

Wilhelm Hasselbring is a professor of software engineering, chair of the TrustSoft Graduate School on Trustworthy Software Systems, and a scientific director in the OFFIS Institute for Information Technology. He received his Diploma degree in Computer Science from the Technical University of Braunschweig, Germany, in 1989; and the Ph.D. degree in Computer Science from the University of Dortmund, Germany, in 1994. From 1998 to 2000, he was with Tilburg University, Netherlands. His research interests include software engineering and distributed systems, particularly software architecture design and evaluation. He is a member of the ACM, the IEEE Computer Society, and the German Association for Computer Science.