

Metrics for Effective Resource Management in Global Computing Environments

Michela Taufer¹, Patricia J. Teller¹, David P. Anderson², and Charles L. Brooks, III³

¹ College of Engineering
University of Texas -El Paso
El Paso, TX 79968, U.S.A.

² Space Sciences Laboratory
University of California -Berkeley
Berkeley, CA 94720, U.S.A.

³ Department of Molecular Biology
The Scripps Research Institute
La Jolla, CA 92037, U.S.A.

{mtaufer, pteller}@utep.edu, davea@ssl.berkeley.edu, brooks@scripps.edu

Abstract

Global computing uses Internet-connected PCs volunteered by their owners. These PCs are diverse, volatile, and error-prone. Sophisticated scheduling methods commonly applied in Grid computing may not be sufficiently scalable and flexible for global computing environments.

This paper shows that it is possible to classify global computing hosts based on simple metrics such as availability and reliability, and that it is efficient to assign tasks to such hosts accordingly. The proposed classification of workers is applied to P@H, a global computing project for protein structure prediction.

Keywords: Large-scale simulations, resource classification, Monte Carlo simulations, molecular dynamics.

1. Introduction

Global Computing (GC) is a scientific computing paradigm that uses computing resources (e.g., desktops and notebooks) connected to the Internet and volunteered by their owners. Many GC projects are simulations of natural phenomena. Examples include: Climateprediction.net [1], which studies climate change; FightingAIDS@Home [2], which is used to discover new drugs to treat AIDS; the United Devices Cancer Research Project [3], which is used to discover drugs to treat cancer; Predictor@Home or P@H [4], which predicts protein structures; and Folding@Home [5], which explores the physical processes of protein folding.

GC environments provide higher throughput than traditional computing systems, such as clusters and

supercomputers, at a lower cost in terms of installation, maintenance, power, and infrastructure. GC projects are based on the master-worker paradigm, where a GC master is a dedicated server machine and GC workers are volunteered PCs.

Strategies for scheduling distributed computation are traditionally based on system behavior and performance predictions [6-8]. These methods do not work well for GC because they are too demanding in terms of resources.

This paper studies the problem of characterizing GC resources with respect to scheduling. We present:

1. an in-depth analysis of the most common strategies used for scheduling computation on large distributed systems, and why these strategies are not suitable for GC (Section 2);
2. a definition of availability and reliability for GC environments, and the classification of GC workers based on these two metrics (Section 3); and
3. the measurement of availability and reliability of workers for a real GC project, the P@H project (Section 4).

2. Common Strategies for Choosing Resources

Adaptive scheduling in large-scale computing environments is discussed in [9-11]. These studies primarily target Grid environments, where resources are managed by organizations such as universities and research centers. In contrast, in GC environments resources are managed by volunteers, and shared among projects based on volunteer preferences. Accordingly, concepts in Grid environments, such as availability of resources (“on” or “off”), do not match

with the characteristics of GC environments, where a resource that is “on” may not be available. Additionally, aspects such as reliability of the resources and their application results, which are important when scientific phenomena are simulated on GC systems, are not considered. In the work reported above, the selection of computing resources does not take into account computing resources that are not frequently available or are not reliable; in GC environments resource availability and reliability must be taken into account. To address this issue, this paper defines the concept of availability and reliability of GC resources. We present a dynamic strategy to hierarchically classify GC based on both these two metrics and to dynamically adapt the distribution of computation so that all resources (GC workers) can participate in GC projects.

Dongarra and others [12-14] have done significant work on adaptive Grid computing and, in particular, have introduced adaptation on GrADS systems. GrADS systems are Grid environments that, as discussed above, have different features than GC environments. The solutions, if applied to GC environments, may compromise scalability and portability because of their high complexity and overhead. The method presented in this paper pays particular attention to both scalability and portability by introducing a reduced number of metrics, i.e., availability and reliability, which can be computed and updated efficiently and are representative of real GC environments.

Wolski and others [6-8, 15, 16] studied the effectiveness of statistical models for predicting machine failure/availability distributions. Statistical models such as exponential, hyper exponential, Pareto, and Weibull distributions were investigated using historical availability, where availability means time during which the machine is powered on. The models were used for simulating scheduling on enterprise- and wide-area distributed computing systems. Three major aspects differentiate the work in [6-8, 15, and 16] from the proposed research. First, it is based on simulations and has not been validated on GC systems. Second, these sophisticated statistical models are not suitable for GC environments on which they may cause high load: the time required by the GC master for accurate predictions of worker availability may be high. Moreover, in GC environments such as BOINC [17], the selection of some workers rather than others may create a condition of starvation for the workers considered less available or less reliable. These workers may overwhelm the master with requests for computation. Besides, volunteers to whom these workers belong may be upset because of their exclusion. Third, the

applications considered in the work discussed above are short-lived applications; the paper focuses on long computations.

3. Availability and Reliability of GC Workers

This paper defines the availability and reliability of workers in GC environments and shows how to apply these metrics to a class of applications commonly used in GC projects.

3.1 GC Simulations

A common class of GC applications search for solutions close to results in nature. In this class of applications, the search is commonly partitioned into work-units (WUs) and implemented via large-scale simulations. Examples include biophysical applications that use Monte Carlo (MC) and Molecular Dynamics (MD) simulations to search conformational spaces of protein structures: protein structure prediction, protein folding, and protein-ligand docking. Several GC projects belong to this class: Predictor@Home, Folding@Home, and FightingAIDS@Home.

A simulation in these GC projects is a set of N searches or WUs:

$$\text{simulation} = \{ WU_i \mid i=0, \dots, N \}$$

where N is on the order of thousands or millions.

Each WU_i is characterized by input parameters (e.g., for biophysical applications, number of MD or MC steps, temperature ranges, atom velocities, and coordinates) that represent initial conditions, p_i for searching in the conformation. In current GC projects, several input parameters such as number of steps or temperature ranges are common to sets of WUs and are defined manually by users based on guesses and intuition. Other parameters, such as the atom velocities in MD simulations, are different for each WU and are defined randomly. Assuming that the models used for representing the simulated system have a high level of accuracy, users can quantify the quality of WU results and sort them based on **quantitative quality measurements**, such as folding and unfolding rates, free energies, and Root-Mean-Square Deviations (RMSDs), which are computed at runtime or at the end of WUs, and then use these measurements to determine whether or not, and how, to adapt initial conditions in an on-going simulation.

In GC environments the following holds:

$$\begin{aligned} WU_{generated}(time_int) &\geq WU_{distributed}(time_int) \geq \\ WU_{completed}(time_int) &\geq WU_{validated}(time_int) \end{aligned}$$

where $WU_{generated}$ is the number of WUs generated by the GC master, $WU_{distributed}$ is the number of WUs distributed to GC workers so far (in progress, completed, or failed WUs), $WU_{completed}$ is the number of WUs completed (and, therefore, returned to the GC master), and $WU_{validated}$ is the number of WUs not affected by errors (with reliable results) over an interval of time ($time_int$). It is unlikely that all the distributed WUs return, but applications based on conformational space searching can afford loss of searches. Also, note that $WU_{completed} \geq WU_{validated}$ since applications require that completed results are reliable and a sanity check is executed to identify these valid results.

3.2 Availability in GC

An available GC worker is not “on” all the time as is the case in cluster or Grid computing. Instead, an available GC worker is one that is productive, i.e., it returns a number of results over a certain interval of time. Indeed the worker can be “on” but because of volunteer-imposed restrictions, such as exclusive CPU use, it may not be dedicated to GC applications. Consequently, the availability of a GC worker, $worker_i$, is defined as:

$$availability_i(time_int) = \frac{WU_{completed\ i}(time_int)}{WU_{distributed\ i}(time_int)}$$

In GC environments supported by middleware packages such as BOINC, measurement of worker availability is not a time- and resource-demanding process. A database already provides the GC master with the number of WUs distributed and completed per worker; an extension of this GC database with fields containing availability and time interval is a minor task. A similar metric is used in [9] for selecting machines that are likely to deliver good aggregate performance in master/slave Grid environments. However, this work does not take into account rates of incomplete WU results due, for example, to volatility of GC resources. Instead it assumes: $WU_{completed} = WU_{distributed}$.

3.3 Reliability in GC

An available worker is not necessarily a reliable worker: returned results may be affected by hardware malfunctions, incorrect software modifications, or malicious attacks. Many participants in public computing projects increase the CPU clock rate. This “overclocking” can cause hardware bit errors (**hardware malfunctions**). These errors can affect floating-point calculations but do not crash the computer. Moreover, these errors occur sporadically, for example because of fluctuations in ambient temperature.

A number of public computing projects make their application source code available, allowing participants to port it to new platforms, modify it, and recompile it so that it runs faster on particular architectures. These modifications may change the output (**incorrect software modifications**). Finally, some participants replace the client software so that it returns incorrect results. These attackers are typically motivated by getting “credit” points, improving their position on the ranked list of participants (**malicious attacks**). Results affected by any of these phenomena (artificial divergences) may introduce errors in simulated results that ultimately cause researchers to come to wrong conclusions. Therefore, these results must be identifiable (i.e., through a sanity check) and not considered. Consequently, a GC worker is reliable if its results pass the sanity check with a high success rate over a certain interval of time. GC projects cannot capture artificial divergences without significantly changing the GC software platform and, consequently, negatively impacting the portability and scalability of GC systems.

To make the situation more challenging, differences in results from two identical computations are not always due to hardware malfunctions, incorrect software modifications, or malicious attacks. Differences in floating-point hardware or in libraries and compilers may also be the cause of divergences. Moreover, MD and MC simulations are subject to positive Lyapunov exponents and, consequently, their results are highly sensitive to initial conditions [18, 19]. All these are natural divergences. In general, it is important to differentiate between artificial and natural divergences, and only dismiss the former.

There is a lack of tools in GC environments to differentiate these two classes of divergences. A common technique to deal with errors or result variations is redundant computing, where the same computation is performed on different workers. If results “agree”, they are correct. Result agreement techniques based on strict equality comparison (i.e., bit-to-bit identical results), fuzzy direct comparison (i.e., results must agree within application-specific tolerances), or fuzzy indirect comparison (i.e., a derived quantity such as energy is within a given threshold) fail for MC and MD simulations because they are not able to differentiate natural divergences from artificial divergences.

The solution applied to GC projects such as P@H emanates from [20], which uses systematic validation techniques based on inspection/validation processes for performance modeling and designing of microprocessors. It extends this concept by applying divergence inspections to GC projects to perform sanity checks of results. A **divergence inspection** is

a post-mortem result comparison: the results related to one instance of a WU are sent to the GC master where they are compared with other results related to other instances of the same WU and transmitted by other GC workers. The GC master inspector can remotely identify hardware malfunctions, incorrect software modifications, and malicious attacks.

Divergence inspections are performed though sanity checks in GC environments such as BOINC. In particular, in BOINC these checks are based on strict equality comparison combined with Homogenous Redundant (HR) techniques [21]. GC masters identify numerically identical workers based on features such as vendor and operating system, and distribute instances of the same WU among them. Sanity checks are performed on the results of multiple instances of the same WUs and the number of valid WUs, $WU_{validated}$, is counted. The reliability of a worker, $worker_i$, is defined as:

$$reliability_i(time_int) = \frac{WU_{validated}_i(time_int)}{WU_{completed}_i(time_int)}$$

3.4 Classification of GC Workers

Availability and reliability are key metrics for the distribution of tasks in GC environments. This is not the case for the capability of workers (i.e., the maximal capacity of the machine in terms of CPU and network speeds) because a worker may have a high CPU speed but be poorly available (e.g., configured so that only a reduced fraction of its computing power is dedicated to the GC project) or be poorly reliable (e.g., affected by hardware malfunctions).

Availability and reliability of GC workers are identified in an *a posteriori* rather an *a priori* way. Accordingly, availability and reliability of workers is not predicted using sophisticated statistical techniques as in [6-8]. Rather the history of workers is the major indicator. Each GC worker, $worker_i$, is characterized by its availability and reliability:

$$worker_i(time_int) \Rightarrow \{availability_i(time_int), reliability_i(time_int)\}$$

At runtime, when a worker applies for new computation, the GC master updates the availability and reliability of each GC worker as well as the availability and reliability thresholds for the entire worker population, $num_workers$:

$$threshold_{availability} = \frac{1}{num_workers} \sum_{i=0}^{num_workers} availability_i(time_int)$$

$$threshold_{reliability} = \frac{1}{num_workers} \sum_{i=0}^{num_workers} reliability_i(time_int)$$

Workers participating in a GC project dynamically fall into four classes: HA/HR, LA/HR, HA/LR, and LA/LR (see Figure 1) based on whether or not their availability and reliability are greater or equal to the thresholds.

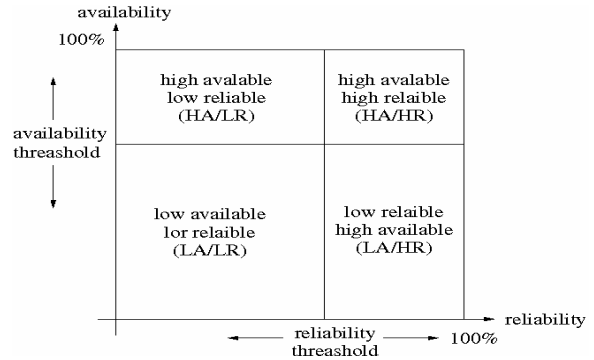


Figure 1: Availability and reliability thresholds

3.5 Discussion

For scientists that deploy GC environments for studying phenomena in nature, result accuracy and reliability are critical issues. Moreover, accurate and reliable results need to be computed with a short turn-around time. Therefore, these GC environments must prioritize WUs, either during their generation (e.g., according to the quality of WU initial conditions) or during their distribution (e.g., according to the WU replication factor). A possible strategy is to favor newly generated WUs with no instance in progress and to assign a certain number of these WU instances to workers by first selecting those workers with high reliability and then those with high availability. The large number of WUs that are handled by a GC project (on the order of thousands or even millions per day) assures the applicability of this scheduling strategy: WUs are continuously generated in queues and waiting for distribution.

The proposed methodology assures that no worker, no matter how available or reliable, starves. For example, assume divergence inspections are used for WUs with three instances dispatched to numerically identical PCs. Additionally assume strict equality comparison is done over the first two results while the third result is used as a double check. In this case, the first two instances are assigned high priority in the queue and are scheduled on HA/HR workers (if available). Because a high level of service is provided by the first two instances, the third instance may be assigned to a worker that does not necessarily supersede both thresholds. Another example is when, while searching for effective initial conditions, the GC master identifies a set of conditions that has high

quality measurements. In this case, rather than remove queued WUs that do not have initial conditions within the identified set, these WUs can be used to double check simulation results. This strategy ensures the simulation does not converge to a local minimum. Moreover, removal of queued WUs is more costly to the GC master than assigning them to less reliable or less available workers

4. A Case Study: Predictor@Home

In this section, the applicability of the two newly defined metrics, availability and reliability, is shown for a real GC project, P@H.

4.1 Predictor@Home

Predictor@Home [4] or P@H is a GC project that searches for protein structures in a large-scale space of protein conformations. Protein structure prediction starts from a sequence of amino acids and attempts to predict the folded, functioning form of the protein either *a priori*, i.e., in the absence of detailed structural knowledge, or by homology with other known, but not identical, proteins. In the case of *a priori* folding or “new fold” prediction, no homology information is available and a search based on the sequence alone is performed. In contrast, homology modeling first identifies other proteins of known structure with some level of sequence similarity to the unknown structure, and then constructs a prediction for the unknown protein by homology. In both cases the choice of a putative candidate is based on energy values; moreover, both approaches may utilize multi-scale optimization techniques to identify the most favorable structural models, e.g., force fields and solvent representations. The protein structure prediction algorithm in Predictor@Home is a multi-step pipeline that consists of (a) sequence analysis and identification of secondary structures as well as potential homology modeling templates using web-based servers such as BLAST [22], SAM-T02, and PSIPRED [23]; (b) conformational search using a MFold/MONSSTER MC simulated annealing approach [24]; and (c) protein refinement, scoring, and clustering using the CHARMM MD simulation package [25]. Predicting the structure of an unknown protein is a critical problem in enabling structure-based drug design to treat new and existing diseases.

P@H uses BOINC as its GC environment for the management of its MC and MD simulations. BOINC is recognized as a standard GC environment. It supports six worldwide GC projects [1, 4, 26-29] to which over 500,000 active volunteers donate cycles. WU instances of MC simulations may last between one and several hours based on the speed of the worker on which they are computed while WU

instances for the MD simulations may last from minutes to one hour [4].

4.2 Measurements

We monitored WU instances and the workers actively participating in P@H for the MFold/MONSSTER and CHARMM applications over an interval of time of 14 days ranging from August 17th to August 31st 2004. Figure 2 shows the two overlapping time intervals during which the two applications were run and monitored. MFold/MONSSTER was monitored for 14 consecutive days and CHARMM was monitored for four days during the Sixth Biannual Critical Assessment of Techniques for Protein Structure Prediction (CASP) [30].

During this time, each worker that returned some results to the P@H master and asked for new computation was classified in one of the four classes (i.e., LA/LR, HA/LR, LA/HR, and HA/HR) based on their availability and reliability as defined in Section 3.2 and Section 3.3, respectively. Two intervals of time, *time_int*, were considered: (1) a cumulative interval that began with the start of the simulations and grew as the time elapsed and (2) a limited interval of 24 hours. Similar worker behaviors were observed for the two intervals of time. In this paper we present the data related to the latter case. We also considered three different availability and reliability thresholds (i.e., 50%, 75%, and 90%).

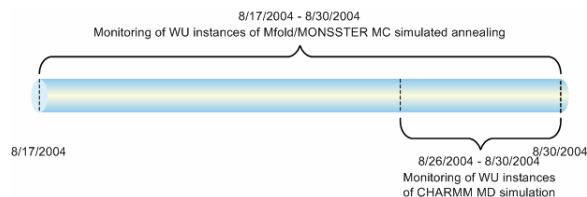


Figure 2: Monitoring time for the two applications: MFold/MONSSTER and CHARMM

Figure 3 presents the total number of workers that completed WU instances over an interval of time, *time_int*, of 24 hours with availability and reliability thresholds of 50% (Figure 3.a). In particular, Figure 3.b shows the total number of workers that contacted the P@H masters to return results and ask for new computation for MFold/MONSSTER over the 24 hours, the number of these workers with low availability and low reliability (LA/LR), the number with high availability and low reliability (HA/LR), the number with low availability and high reliability (LA/HR), and the number with high availability and

high reliability (HA/HR). Figure 3.c shows the same information for CHARMM.

In Figure 4, a threshold of 75% is considered for both availability and reliability thresholds (Figure 4.a). Figure 4.b presents the number of workers (i.e., total, LA/LR, HA/LR, LA/HA, and HA/HR) for MFold/MONSSTER and Figure 4.c presents the same information for CHARMM. Finally, Figure 5 presents the same information but for higher availability and reliability thresholds (i.e., 90%).

Our classification indicates that reliability is the most critical aspect in P@H: while the workers were highly available over the simulation period, they realized a loss in reliability at the end of the simulation. In Figures 4 and 5, for high reliability thresholds, we can clearly identify a region in which a significant number of workers have low reliability. As depicted in Figures 6 and 7, such a region is located at the end of both simulations. Figure 6.a presents the number of WUs generated, distributed, and completed over the 14 days in which the MFold/MONSSTER application was monitored. Figure 6.b reports, once again, the number of completed WUs but sorts them based on the results according to erroneous, valid, invalid, and pending (i.e., still waiting for validation). Figures 7.a and 7.b presents the same data for the CHARMM application over the shorter 4-day interval of time.

Figures 6 and 7 indicate that the loss in reliability takes place at the end of simulations, when users no longer generate and distribute large numbers of WU instances (Figures 6.a and 7.a). Figures 6.b and 7.b show that the reason for the loss in reliability has to be investigated in the P@H master and, in particular, in how BOINC manages WU instances. If a WU is returned (completed) with an error or its results do not pass the sanity check, then BOINC generates a new instance of the WU (the master can generate WU instances in addition to the instances that users submit to the GC environments) as long as enough correct instances are returned that pass the sanity check or a certain error threshold (fixed by the user) is reached. This causes continuous re-generation of WU instances that are destined to fail because of their ill-defined initial conditions. For applications that are more error-prone, such as CHARMM [21], this phenomenon is more tangible. Moreover, in the last phase of both simulations, the number of WU instances with results waiting for validation increased (pending results). These pending results were not included in the number of valid results used to measure the reliability of workers. Accordingly, if a worker has a large number of pending results, then its reliability drops even though the worker, itself, is not the cause of this. Instead, the reason for the drop in

reliability is the incapability of the P@H master to immediately validate the completed result.

These observations suggest the need for adaptive availability and reliability thresholds that change as the simulation evolves: high thresholds at the beginning of a simulation need to decrease as the simulation evolves and the number of WU instances distributed for a specific application become smaller.

5. Conclusions

This paper defines availability and reliability of computing resources (workers) in GC environments. It shows how, by using these metrics, to provide a classification and management of workers that is more effective than traditional scheduling methods available in Grid computing. Such metrics and the related classification of workers were easily applied to a real GC project, P@H. Observations of two different large-scale simulations (i.e., Monte Carlo and Molecular Dynamics simulations) on P@H demonstrated how adaptive setting of availability and reliability thresholds for the classification of workers better supports the varied computational needs of different phases of a simulation and the dynamic nature of the number of workers participating to a GC project.

6. References

- [1] ClimatePrediction.net: a Forecast of the Climate in the 21st Century. <http://climateprediction.net>, 2004
- [2] A. Olson, et al.: FightAIDS@Home: Accelerate AIDS Research by Deploying Global "Grid" of Distributed Computing Power. <http://fightaidsathome.scripps.edu>
- [3] The United Devices Cancer Research Project. <http://www.grid.org>, 2002
- [4] M. Tauber, C. An, A. Kerstens, and C.L. Brooks III: Predictor@Home: A "Protein Structure Prediction Supercomputer" Based on Public-Resource Computing. In *Proc. of the 4th IEEE International Workshop on High Performance Computational Biology (HiCOMB'05)*, April 2005
- [5] V. Pande, et al.: Atomistic Protein Folding Simulations on the Submillisecond Time Scale Using Worldwide Distributed Computing. *Biopolymers*, 2003, 68:91-109
- [6] J. Brevik, D. Nurmi, and R. Wolski: Automatic Methods for Predicting Machine Availability in Desktop Grid and Peer-to-Peer Systems. In *Proc. of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid'04)*, Apr 2004
- [7] D. Nurmi, J. Brevik, and R. Wolski: Minimizing the Network Overhead of Checkpointing in Cycle-harvesting Cluster Environment. Submitted for publication, 2005
- [8] D. Nurmi, J. Brevik, and R. Wolski: Modeling Machine Availability in Enterprise and Wide-area Distributed Computing Environments. Submitted for publication, 2005

- [9] G. Shao, F. Berman, and R. Wolski: Master/Slave Computing on the Grid, In *Proc. of the 9th Heterogeneous Computing Workshop (HCW'00)*, May 2000
- [10] F. Berman, et al.: Adaptive Computing on the Grid using AppLeS. *IEEE Trans. on Parallel and Distributed Systems (TPDS)*, 2003, 14(4):369-382
- [11] E. Heymann, M.A. Senar, E. Luque, and M. Livny: Adaptive Scheduling for Master-Worker Applications on the Computational Grid. In *Proc. of the First IEEE/ACM International Conference on Grid Computing (GRID'00)*, Dec 2000
- [12] F. Berman, et al.: The GrADS Project: Software Support for High-Level Grid Application Development. *International Journal of High Performance Computing Applications*, 2001, 15(4):327-344
- [13] S.S. Vadhiyar and J.J. Dongarra: Self Adaptivity in Grid Computing. *Journal of Concurrency Computation: Pract. Exper.* 2004
- [14] F. Berman, et al.: New Grid Scheduling and Rescheduling Methods in the GrADS Project. *International Journal of Parallel Programming*, 2005, 33(2)
- [15] D. Kondo, M. Taufer, C. L. Brooks III, H. Casanova, and A.A. Chien: Characterizing and Evaluating Desktop Grids: An Empirical Study. In *Proc. of the IEEE/ACM International Parallel and Distributed Processing Symposium (IPDPS'04)*, Apr 2004
- [16] D. Kondo, A.A. Chien, and H. Casanova: Resource Management for Rapid Application Turnaround on Enterprise Desktop Grids. In *Proc. of the ACM Conference on High Performance Computing and Networking (SC'04)*, Nov 2004
- [17] D.P. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *Proc. of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, Nov 2004.
- [18] A.M Ozorio de Almeida: *Hamiltonian Systems: Chaos and Quantization*. Cambridge University Press, 1988, Cambridge, UK
- [19] M. Braxenthaler, R. Ron Unger, D. Auerbach, J.A. Given, and J. Moulton, J: Chaos in Protein Dynamics. *Proteins*, 1997, 29: 417-425
- [20] B. Black and J.P. Shen: Calibration of Microprocessor Performance Models. *Computer*, May 1998, 31(5):59-65
- [21] M. Taufer, D.P. Anderson, P. Cicotti, and C.L. Brooks III: Homogeneous Redundancy: a Technique to Ensure Integrity of Molecular Simulation Results Using Public Computing. In *Proc. of the 14th Heterogeneous Computing Workshop (HCW'05)*, Apr 2005
- [22] S.F. Altschul, T.L. Madden, A.A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman: Gapped BLAST and PSI-BLAST: a New Generation of Protein Database Search Programs. *Nucleic Acids Res.*, 1997, 25:3389-3402
- [23] L.J. McGuffin, K. Bryson, and D.T. Jones: The PSIPRED Protein Structure Prediction Server. *Bioinformatics*, 2000, 16:404-405
- [24] J. Skolnick, A. Kolinski, and A.R. Ortiz: MONSSTER: A Method for Folding Globular Proteins with a Small Number of Distance Restraints. *J Mol Biol*, 1997, 265:217-241
- [25] A.D. MacKerell Jr., B. Brooks, C.L. Brooks III, L. Nilsson, B. Roux, Y. Won, and M. Karplus: CHARMM: The Energy Function and Its Parameterization with an Overview of the Program. *The Encyclopedia of Computational Chemistry*, 1998, 1:271-277. P. v. R. Schleyer et al., editors (John Wiley and Sons: Chichester)
- [26] D.P. Anderson, et al.: SETI@Home: Search for Extraterrestrial Intelligence at Home. <http://setiathome.ssl.berkeley.edu>
- [27] Einstein@Home: Search for Gravitational Signals Coming from Pulsars. <http://einstein.phys.uwm.edu/>
- [28] LHC@Home: The CERN LHC Particle Accelerator. <http://athome.web.cern.ch/athome>
- [29] Cell Computing Biomedical Research. <http://www.cellcomputing.net/simple/index.php>
- [30] Critical Assessment of Techniques for Protein Structure Prediction. <http://predictioncenter.org>

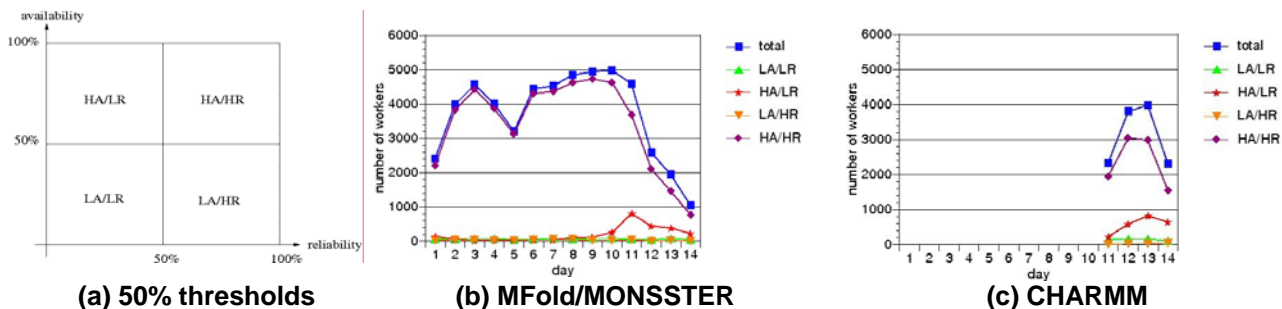


Figure 3: Availability and reliability thresholds of 90% (a), number of workers (i.e., total, LA/LR, HA/LR, LA/HA, HA/HR) for MFold/MONSSTER (b), and CHARMM (c)

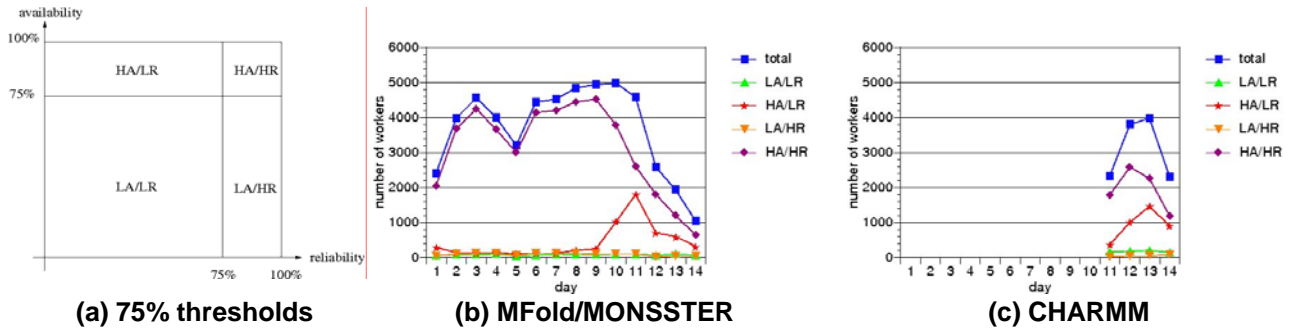


Figure 4: Availability and reliability thresholds of 50% (a), number of workers (i.e., total, LA/LR, HA/LR, LA/HA, HA/HR) for MFold/MONSSTER (b), and CHARMM (c)

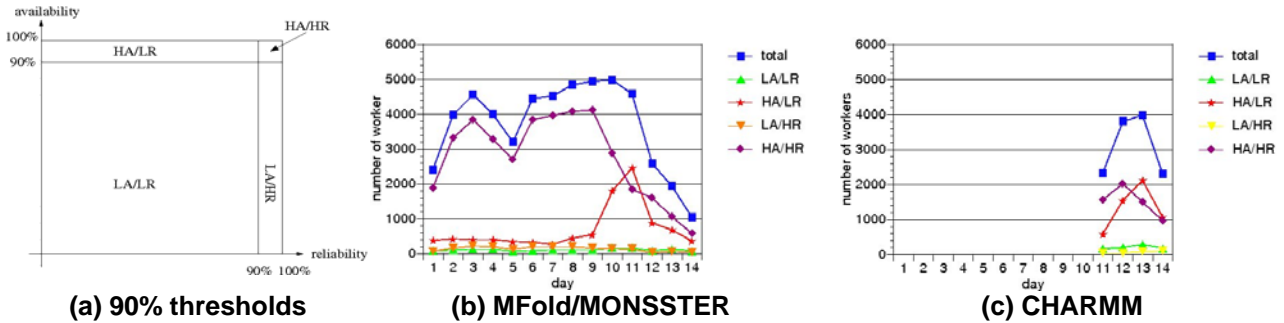


Figure 5: Availability and reliability thresholds of 90% (a), number of workers (i.e., total, LA/LR, HA/LR, LA/HA, HA/HR) for MFold/MONSSTER (b), and CHARMM (c)

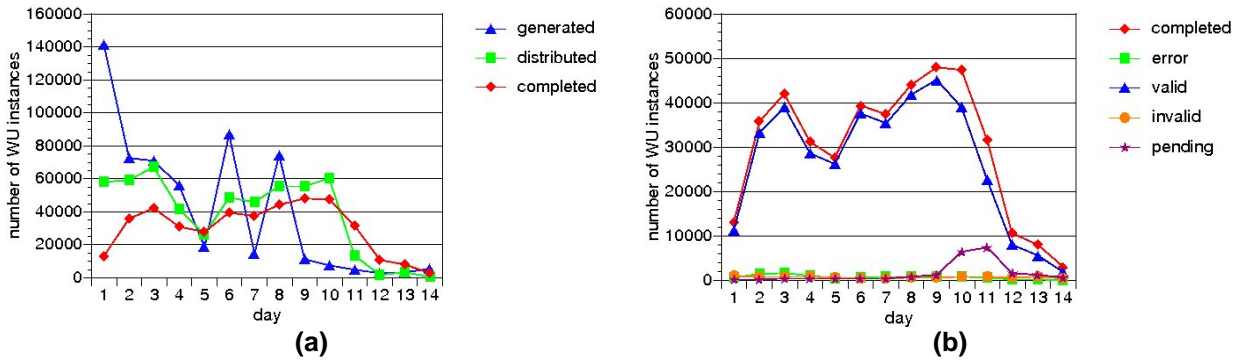


Figure 7: Number of WU instances generated, distributed, and completed per day for MFold/MONSSTER (a) and related number of completed WU instances with erroneous (error), valid, invalid and pending results (b)

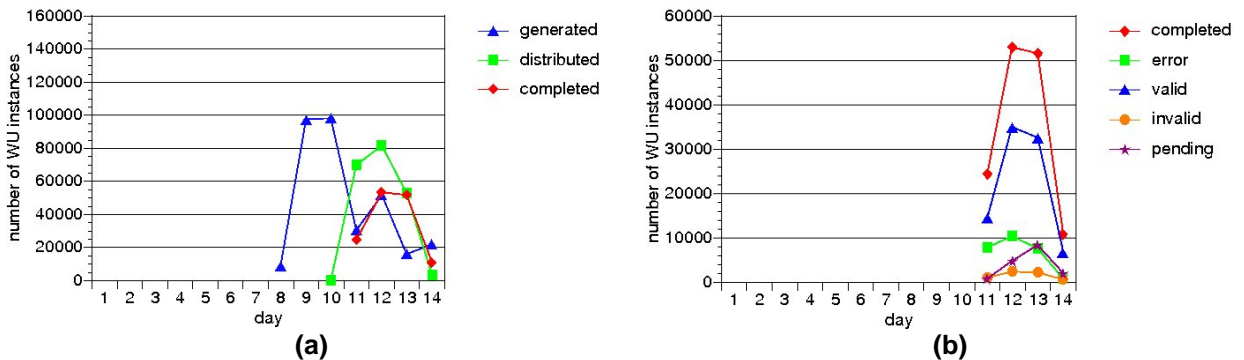


Figure 7: Number of WU instances generated, distributed, and completed per day for CHARMM (a) and related number of completed WU instances with erroneous (error), valid, invalid and pending results (b)