Applying the Heterogeneity Level Metric in a Distributed Platform

PAULO S. L. SOUZA
FABIO HISTOSHI
MARCOS J. SANTANA
REGINA H. C. SANTANA
SARITA M. BRUSCHI
KALINKA R. L. J. C. BRANCO

USP - University of São Paulo
ICMC - Institute of Mathematics and Computer Sciences
SSC - Computer Systems Department
P.O. Box: 668 - 13560-970 - São Carlos (SP) - Brazil
{pssouza, hitoshi, mjs, rcs, sarita, kalinka}@icmc.usp.br

Abstract. Heterogeneity Level (HL) metric has been developed by our research-group to help scheduling algorithms to adapt themselves to the existent heterogeneity in the platforms. This paper presents our results considering the HL's behaviour in a real adaptive scheduling. HL metric quantifies qualitative aspects from heterogeneity in order to provide efficient performances and lower cost to the execution in both heterogeneous and homogeneous platforms. HL use is investigated under different perspectives: CPU, memory, network and considering benchmarks results. A simple but effective adaptive scheduling using HL is proposed and its results point out to performance-gains around 53% when a non-adaptive scheduling algorithm is used. Our case studies show that the HL was efficient, flexible and easily used for scheduling policies.

Keywords: heterogeneity, load balancing, cluster.

(Received February 22nd, 2011 / Accepted May 2nd, 2011)

1 Introduction

Heterogeneous distributed platforms allow exploring different and specific resources according to different demands. They extend the platform performance through both gradual improvements and reuse of the resources already available in the organization. However, associating different resources with diversified demands implies to compute a more complex strategy for this distribution. Heterogeneity must be used carefully in order to improve the computing cost vs. benefit relation.

Processes scheduling is directly affected by heterogeneity. It is necessary to consider relevant aspects from both platform and applications demand when the resources present different features, architecture or performance. On the other hand, when the platform is homogeneous, the scheduling may encapsulate details from devices and basic software (such as operating systems and compilers), because they present a uniform behavior, performance and architecture. This allows simpler and cheaper scheduling with efficiency. Other important point is that heterogeneity can be temporal as well, due to workload dynamical variation and node-changes in the platform [3].

Branco et al. [3] proposed the HL (Heterogeneity Level) metric to quantify the platform heterogeneity, considering the performance's dispersion from each node, in relation to an average performance [3]. Their preliminary results show the HL performance under simulation. This paper presents our main results from considering the HL's metric in a real adaptive schedul-

ing for distributed platforms. Our aim in this paper is compare the HL metric behavior when applied in an adaptive scheduling policy on a platform with different heterogeneity levels.

The HL behavior is investigated using two different perspectives: changes in the hardware-resources and with distinct benchmarks. These hardware resources and benchmarks allow analyzing the HL behavior according to different and real perspectives, such as: floating-point, integer operations, memory use and network consumption [6, 7, 10, 11, 12, 14, 15].

A novel adaptive scheduling algorithm has been proposed to investigate HL impact in this context. This algorithm is used by AMIGO (DynAMical FlexIble SchedulinG EnvirOnment) [13] and PVM (Parallel Virtual Machine) [5]. The choice by PVM is due to both its source code structure and its tightly coupling to AMIGO. However, the PVM choice does not imply in generality loss, because the studies presented here are focused on the HL impact mainly, independently if this scheduling is done by MPI, PVM or by a distributed operating system. Indeed, the novel scheduling policy and the investigations about the use of the HL are orthogonal to the message passing interface used.

The best results in our case studies show that the adaptive scheduling using HL metric allow a real performance gain around 53% for the application runtime. The HL is simple to be used in scheduling algorithms and presents a stable behaviour.

This paper is organized as follows. Section 2 presents some basic definitions about heterogeneity. Section 3 presents HL metric and its behavior considering benchmarks demand. Section 4 presents the adaptive scheduling algorithm proposed, describing how to use the HL metric. Section 5 describes the main results obtained using the HL metric and Section 6 presents the concluding remarks.

2 Heterogeneity and Homogeneity

There are different kinds of heterogeneity [3]. Initially, it can occur considering the configuration and architecture. There is configuration heterogeneity when differences of performance are observed on devices with the same platform (hardware and basic software). Architectural heterogeneity implies different devices when considering hardware and/or basic systems.

Heterogeneity can be considered as positive or negative, depending on heterogeneity contribution for the system performance. We have a positive heterogeneity when devices with better performance in relation to previous ones are added in the platform. A negative heterogeneity (or a performance lack) can occur when devices with worse performance are added in a platform [3].

Time is other important feature related to heterogeneity. There is temporal or dynamical heterogeneity if the platform presents a homogeneous behavior in determined situations and heterogeneous in other. Factors that contribute for a temporal heterogeneity are: workload, multi-users and the resources being considered to report the heterogeneity level. Considering the last one (resources), for example, when two nodes have both distinct processors and the same memory quantity/type, they can be heterogeneous under CPU point-of-view and homogeneous when considering memory. Depending on application demand, the platform can be considered heterogeneous or homogeneous [3].

Different research works quantify the platforms heterogeneity degree [1, 3] [6, 14, 16]. Some models and metrics for heterogeneous systems were proposed by Zhang and Yang [16], in which heterogeneous computing systems can be represented by a graph (M, C), where $M = M_1, M_2, M_3, M_4, M_5, ..., M_n$ is considered a set of heterogeneous workstations and C is the communication network linking the workstations (with a homogeneous bandwidth). Aiming to quantify the heterogeneity of a system machines without using complex measurements, Zhang and Yang [16] proposed two metrics to evaluate the relative computing power of a set of workstations (the capacity of each workstation is evaluated in comparison to the fastest one):

$$W_i(A) = \frac{S_i(A)}{max_{i=1}^{n}\{S_i(A)\}} \quad (1)$$

Where $i = 1, ..., n$ and $S_i(A)$ represents the speed of M_i to execute application A dedicatedly. Speed can be defined by the number of basic operations per time unit, for instance, and the computing power of each workstation is represented by a relative speed. A second metrics proposed is:

$$W_i(A) = \frac{min_{i=1}^{n}\{T(A, M_i)\}}{T(A, M_i)} \quad (2)$$

Where $i = 1, ..., n$ and $T(A, M_i)$ is the time required to execute application A at workstation M_i. Grosu [6] extends these metrics so that the computing power is given by the relative speed of the workstation in relation to the slowest one:

$$W_i(A) = \frac{min_{i=1}^{n}\{S_i(A)\}}{S_i(A)} \quad (3)$$

Where $i = 1, ..., n$ and $S_i(A)$ is the speed of workstation M_i to execute application A dedicatedly, and the

computing power is given by relative speeds. Furthermore, Grosu [6] defines:

$$W_i(A) = \frac{T(A, M_i)}{max_{i=1}^{n} \{T(A, M_i)\}} \quad (4)$$

Where $i = 1, ..., n$ and $T(A, M_i)$ is the time it takes to execute application A at workstation M_i.

Thus, equations 1 and 2 now act as the basis to define the computing power, considering the fastest machine as a reference point, which is renamed $W_i^f(f - fast)$. On the other hand, equations 3 and 4 identify the computing power based on the slowest machine, which is represented by $W_i^s(s - slow)$. Four ways to quantify the heterogeneity level in a system based on the value of W are proposed in [16] and [6]. The first and second use the standard deviation H_1, which can be calculated based on the computing powers compared to either the fastest or the slowest workstation:

$$H_1 = \sqrt{\frac{\sum_{i=1}^{n}(W_{med} - W_i)^2}{n}} \quad (5)$$

The mean absolute deviation, called H_2, also calculated based on the fastest or the slowest workstation:

$$H_2 = \frac{\sum_{i=1}^{n} |W_{med} - W_i|}{n} \quad (6)$$

where:

$$W_{med} = \frac{\sum_{i=1}^{n} W_i}{n} \quad (7)$$

The values in both H_1 and H_2 are observed and analyzed uniformly, using the average to find the standard deviation and the mean absolute deviation. However, this uniformity invalidates the analysis when there are reasonable differences among the workstations computing powers, since the standard deviation cannot reflect computer systems.

Based on this weakness of the H_1 and H_2 metrics, Zhang and Yang [16] proposed a third metric, H_3, evaluated from the fastest workstation in the computing system:

$$H_3 = \frac{\sum_{i=1}^{n}(1 - W_i^f(A))}{n} \quad (8)$$

Similarly, Grosu [6] defines H_4 based on the computing power of the slowest workstation in the computing system:

$$H_4 = \frac{\sum_{i=1}^{n}(1 - W_i^s(A))}{n} \quad (9)$$

In H_3, the computing power of the fastest workstation is equal to 1 while, in H_4, the slowest machine has a computing power value of 1. Thus, H_4 represents the difference of computing power between each machine and the fastest machine and H_3 calculates the same difference between each machine and the slowest one.

Based on his experiments, Grosu [6] states that the metric H_4 is more suitable than H_3. However, the case studies presented in [3] demonstrate the fallacy of that statement in some situations, because the metrics present contradictory behaviour when evaluating the same platform, in different cases.

Branco et al. [3] propose the *HL* (Heterogeneity Level) metric to eliminate these discrepancies. *HL* considers a hypothetic (not real) standard node, representing the nodes average performance [3]. This metric will be detailed in next section due to its importance for this work.

3 The HL Metric

Platform heterogeneity can be quantified considering different perspectives, such as: architectural aspects, operating systems or resources performance [15]. In this sense, Branco et al. proposed the *HL* metric [3] to quantify the heterogeneity, using a virtual node (called standard node), which represents the average performance in the platform. The dispersion around this standard node allows quantifying the platform heterogeneity level, in a similar way to the works presented in [6] and [16] that use respectively the worst and the best node as standard node. The main difference between *HL* metric and those two is that *HL* has a uniform behaviour when quantifying distinct heterogeneity levels and also both the positive and the negative heterogeneities. *HL* quantifies the platform heterogeneity level through equation 10, where n represents the amount of nodes in the platform, X_i is the $node_i$ performance and $\overline{X}$ is the virtual standard node performance.

$$HL = \frac{\sum_{i=1}^{n} \left|X_i - \overline{X}\right|}{n * \overline{X}} \quad (10)$$

The preliminaries results using the *HL* [3] considered a general parameter called "speed", in order to qualify application demands. Indeed, Branco et al. considered "speed" as a generic value, which should be easily instantiated later at real parameters, such as: MIPS, MFLOPS, runtime, RAM amount or other.

Different experiments were conducted by Branco et al. [3] to simulate the *HL* behaviour when including new nodes in a heterogeneous platform with just three nodes with "speeds" 10, 100 and 1000.

Figure 1 show the *HL* behaviour when nodes identical to the fastest one are added. The heterogeneity degree behavior is coherent, since as similar high-speed

nodes are being added, the system heterogeneity level drops and stabilizes close to zero. Few nodes with high-speeds change the platform status quickly for homogeneous. This situation could be represented by the metrics proposed by Zhang (Zhang & Yang) as well; however, it is not properly represented by Grosu's metrics [6], due to standard node, based on the slowest workstation.

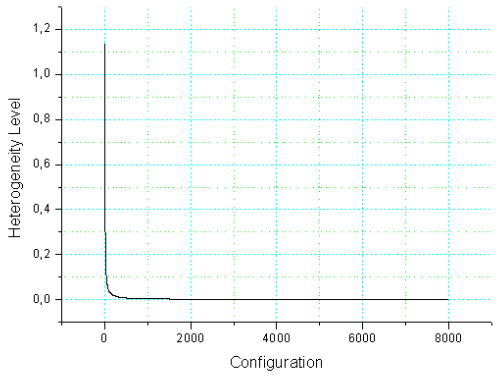

Figure 1: Behavior of the heterogeneity degree when nodes identical to the system's fastest node are added (initial speeds of 10, 100 and 1000)

Figure 2 shows the *HL* behaviour when nodes identical to the slowest one are added. The heterogeneity level rises at a first moment, because more nodes with "speed" 10 are required to reach the speed of the fastest node. When they reach this threshold, the heterogeneity level falls near to zero, where it must stabilize. The *HL* metric evaluations done by Branco et al. [3] were not concerned about static or dynamic behaviour of the environment being used for the experiments, since the "speed" parameter encapsulated this question.

However, metrics based on static data, such as hardware features, offer just a partial view of both performance and heterogeneity. They are not able to represent usual dynamic changes in the platforms that make them temporarily heterogeneous. In this sense, dynamic metrics, such as runtime, can point out the heterogeneity level on-the-fly and according to user point-of-view [12]. Runtime encapsulates basic details from hardware and software, grouping them in a common point: to offer better performance (considering time) to end-user applications. Indeed, if used properly, time allows a performance comparison while encapsulating architecture details.

We develop initially two new experiments with *HL* metric to show these perspectives. For the first one

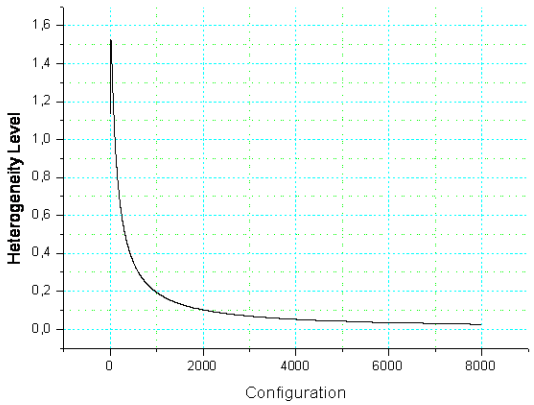

Figure 2: Behavior of the heterogeneity degree when nodes identical to the system's slowest node are added (initial speeds of 10, 100 and 1000)

we use real and static hardware features to evaluate the *HL* metric. For the second one we use dynamic benchmarks results to determine the heterogeneity. Both experiments consider the same platform. The hardware features considered for the first analysis are: CPU frequency, cache amount, RAM amount, swap-memory amount and network peak throughput (see Table 1). The second analysis considers six different open-source benchmarks: (Whetstone, Dhrystone and Linpack), memory (Stream and Cachebench) and network (Netperf). Table 2 points out the main features evaluated in each benchmark and their respective metrics [4, 7, 9, 10, 11, 14]. These benchmarks were chosen because they are: (1) meet the specific-demands planned for our experiments, (2) open-source and (3) free.

The experiments were executed on a cluster with 5 nodes, all using GNU/Linux, distribution OpenSuse 10.0, 100Mbits ethernet network and gcc compiler. Table 1 contains the *HL* resulting from each hardware feature analyzed, where it is possible to observe a stable *HL* behavior. This platform can be viewed as homogeneous if the network maximum throughput is considered and as heterogeneous one if the CPU performance is taken account. The demand generated by application should determine which metric must be used to reach effectiveness when using the heterogeneity level with these data. This implies in a previous study of the demand and usually is associated with monitoring software tools, which automates this process and helps managers to characterize the demand in a correct way [8].

Table 1: *HL* results when evaluated on a five nodes cluster and according to hardware features such as: CPU frequency, cache amount, RAM amount, swap amount and network peak throughput.

Features	*Node1*	*Node2*	*Node3*	*Node4*	*Node5*	*HL*
CPU (MHz)	400.91	451.05	1200.07	1666.73	2017.99	0.50
Cache (KB)	512	512	64	256	512	0.45
RAM (MB)	192	128	256	256	256	0.21
Swap (MB)	196	243	415	512	256	0.34
Network (Mbps)	100	100	100	100	100	0.00

It can be observed through Table 3 that *HL* metric is also efficient to represent the platform heterogeneity and it presents a stable behaviour according to benchmarks results. The values showed in the Table 3 represent the average of 30 executions.

The *HL* values in Tables 2 and 3 point out some possible discrepancies that occur when considering heterogeneity in a distributed platform. An example is the *HL* value obtained from CPU feature (0.50) and the Whetstone result (0.74), because both consider CPU performance. In these cases, the results obtained from the benchmark were considered more efficient to represent the platform heterogeneity, since they intend to indicate the real performance for the user.

This difference can also be observed with memory and network. Network is a critical case, since the platform is homogeneous (*HL*=0.00) when considering peak performance (100Mbps). However, Netperf benchmark shows that when different nodes send messages to (or receive from) node 5, the platform presented the second major *HL* result (0.67) and thus, can be considered heterogeneous. Again, in these cases, the benchmark results should be used because they represent the performance expected by final user. In these cases, a simple view considering just one hardware feature is not a better choice to estimate the heterogeneity level.

4 An Adaptive Scheduling *As/Hl*

An adaptive scheduling based on the heterogeneity level (or *As/Hl*) was developed in this work to investigate the *HL* metric impact in a real scenario, considering the end-user perspective. The *As/Hl* is adaptive because it changes dynamically the scheduling algorithm according to the *HL* metric.

AMIGO (DynAMical FlexIble SchedulinG EnvirOnment) [13] and PVM (Parallel Virtual Machine) [5] were used to insert the *HL* metric in the *As/Hl*. AMIGO allows grouping specific scheduling policies according to different demands. The choice by PVM is due to its source code structure and because it is tightly coupled to AMIGO. However, it is important to note that the choice by AMIGO/PVM does not cause generality loss, because this policy could be applied in other contexts, such as: MPI environment, operating system or directly inside parallel application code. The *As/Hl* determines which scheduling algorithm must be used considering the platform heterogeneity. The aim is minimizing scheduling costs and at the same time maximizing its benefits.

Table 2: Benchmarks used to evaluate the *HL* metric behaviour.

Category	*Benchmark*	*Demand*	*Metric*
CPU	Whetstone	Floating-Point	Execution Time
	Dhrystone	simple arithmetic, strings, logical and access to the memory	execution time and dhrystones; performance in relation to Vax 11/780 for one benchmark iter
	Linpack	linear eq systems float/double in arrays	FLOPS and execution time
Memory	Stream	memory throughput	throughput and avg execution time
	Cachebench	accesses to memory and to cache	throughput
Network	Netperf	latency, TCP/UDP throughput	throughput and avg execution time

Table 3: *HL* results when evaluated on a five nodes cluster and according to six distinct benchmarks.

Benchmark	*Node1*	*Node2*	*Node3*	*Node4*	*Node5*	*HL*
Whetstone (s)	1119.0	995.4	265.0	190.4	1265.0	0.74
Dhrystone (s)	3.2	2.8	1.0	0.7	0.8	0.49
Linpack (s)	0.020	0.018	0.006	0.004	0.005	0.52
Stream (s)	0.31	0.30	0.07	0.06	0.04	0.58
Cachebench (Mb/s)	984.2	1105.0	3509.6	5054.5	5400.3	0.53
Netperf (Mb/s)	64.9	5753.3	5559.5	15988.2	x	0.67

AMIGO is basically composed by an upper and a

lower layer. Upper layer is responsible by the configuration, while lower layer is responsible by scheduling policies, AMIGOD (AMIGO Daemon), message-passing environment (in this work instantiated by PVM) and parallel applications [13].

AMIGO has scheduling policies for memory-bound, network-bound and CPU bound applications. DPWP (Dynamical Policy Without Preemption) is one of them, which presents features such as: dynamic (decides the scheduling at runtime), specific to CPU-bound applications and does not consider preemption [2]. The scheduling done by DPWP aims to balance new workloads considering the existent nodes load. It tries to normalize this workload using a relative-performance in relation to the whole platform. DPWP normalizes the ready-processes in the ready-queue to determine the target node.

AMIGO acts just when the message-passing environment needs to schedule new processes on the platform. Figure 3(a) shows the steps followed by the original PVM in this case. An application requests the scheduling from $pvm\_spawn()$, which forwards the request to local PVMD. The local PVMD, running $tm\_spawn()$, create a list contending nodes that will receive the new processes using by default a round-robin policy. The functions $assign\_tasks$ and dm_exec are called later to create and to start these processes, respectively.

To interact with AMIGO, $tm\_spawn()$ routine was modified (Figure 3(b)), where processes selection is always requested to AMIGOD through the $GetHostsFromAMIGOD()$. This request will be attended by an AMIGO's scheduling policy, independently if round-robin policy existing in PVM could present better results or not, according to used platform. The *HL* metric was inserted in this scenario trying to solve this problem by analyzing the platform heterogeneity and creating a simple but efficient adaptive scheduling algorithm in this point.

AMIGO is requested to give a nodes-relation according to DPWP policy when the platform is heterogeneous; when it is homogeneous, the request is not sent to AMIGO and the original round-robin policy is used to determine the nodes target to the scheduling. Figure 3(c) shows the algorithm basic steps, highlighting just the $tm\_spawn()$ routine. This adaptive algorithm is based on a conditional structure comparing the *HL* value returned from $eval\_HL()$ with a threshold, called $standard\_HL$. The correct choice of this value is a complex question and needs to be investigated in a more detailed sense. Unfortunately, this study does not belong to the scope of this paper. The $standard\_HL$ must be defined by the system manager considering the expected demand, used platform and objectives. The $standard\_HL$ was empirically fixed as 0.01 in this paper, as explained in the next section.

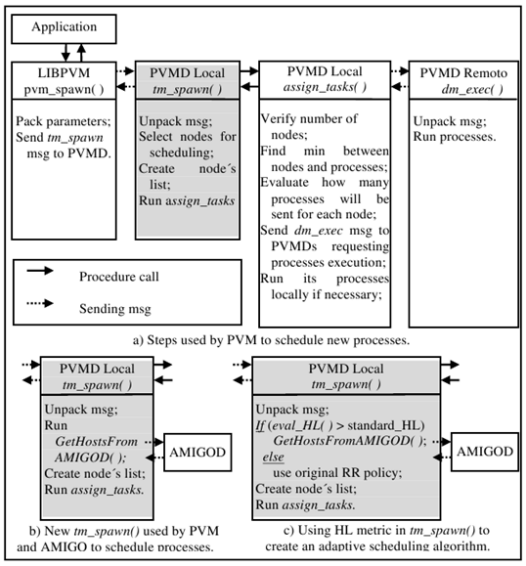

Figure 3: Steps followed by (a) original PVM, (b) PVM/AMIGO and (c) As/Hl algorithm in order to schedule new processes.

The *HL* is evaluated by the $eval\_HL()$ considering the nodes performance. The activities conducted by $eval\_HL()$ are: gets the computing performance from each node, evaluates the *HL* through equation 10 and returns the *HL* value.

The node performance can be determined from different ways. In this paper, they were instantiated through benchmarks previously executed.

5 Experiments and Results with *As/Hl*

The objective of the experiments with the *As/Hl* is to analyze the *HL* efficiency when it is applied in scheduling policies on distributed platforms. To reach this objective, it was developed an experimental study using a parallel application responsible to solve linear systems, based on Gauss-Jacobi iterative method. Gauss-Jacobi application was chosen because it is CPU-bound application, representative for a large number of HPC programs. The version executed in this work is compound by a master code responsible by dynamically generate slaves, these ones able to solve a variable sub-group from the linear system. The new slave processes generation is made on-the-fly by master processes. The master code evaluates the linear system convergence and, before starts a new iteration, it decides either to stop or

not the execution.

Experiments were done in a Beowulf cluster with ten nodes, all of them with: Intel Pentium4 Processor (64bits and 3.4GHz), RAM with 4GBytes, a Gigabit Ethernet network and GNU/Linux operating system.

This cluster is a homogeneous platform. Fixed-and-extra workloads were inserted into some nodes, turning this platform a temporally-heterogeneous one with three different levels: totally homogeneous, partially heterogeneous and totally heterogeneous. The synthetic extra workloads were generated using the Linpack, Dhrystone and Whetstone benchmarks. The homogeneous scenario is composed by ten nodes without any extra workload. In the partially heterogeneous scenario the nodes are grouped into five distinct pairs with workload ranging from 0% to 40% in relation to CPU utilization. For the totally heterogeneous scenario the nodes are also grouped in pairs, being generated for them extra workloads ranging from 0% to 80% in relation to the CPU utilization. Benchmarks used to create the two heterogeneous-scenarios affected the platform performance in a fixed and constant way, during the whole experiment.

The Linpack benchmark was used to evaluate the node performance in these three platforms. It was chosen because it has features close to the parallel application used: operations with floating-point vectors to solve linear systems. The results obtained from the Linpack were applied to equation 10 in order to evaluate the *HL* metric. Using Linpack to establish the heterogeneity in platform allows focusing on CPU features, majorly those related to FPU. Table 4 presents the results when using Linpack and *HL* for the three scenarios discussed.

Table 4: Results for Linpack benchmark and *HL* metric.

	Scenarios Used					
	Totally Homogeneous $HL = 0.0002$		Partially Heterogeneous $HL = 0.22$		Totally Heterogeneous $HL = 0.57$	
Nodes	Extra Load	Time (ms)	Extra Load	Time (ms)	Extra Load	Time (ms)
1	0%	0.506	0%	0.505	0%	0.505
2	0%	0.504	0%	0.504	0%	0.504
3	0%	0.505	20%	0.684	20%	0.684
4	0%	0.504	20%	0.684	20%	0.684
5	0%	0.505	40%	1.074	40%	1.073
6	0%	0.504	40%	1.072	40%	1.072
7	0%	0.505	10%	0.585	60%	1.965
8	0%	0.504	10%	0.585	60%	1.964
9	0%	0.505	30%	0.907	80%	4.944
10	0%	0.504	30%	0.907	80%	4.940

The threshold used by *HL* to determine if the platform is either homogeneous or heterogeneous ($standard\_HL$) was arbitrarily fixed in 0.1, due to *HL* results obtained from the three platforms. This *HL* value allows separating the totally homogeneous platform (*HL*=0.0002) from other two possibilities: partially and totally heterogeneous with *HL* 0.22 and 0.57, respectively.

In a first execution, the Gauss-Jacobi application was scheduled using the *As/Hl* algorithm on the three platforms and according $standard\_HL$ value. This means that the policy used was the round-robin when executing the homogeneous scenario and the DPWP when executing on the partially and totally heterogeneous platforms.

In order to compare the scheduling done for each platform, we repeated the executions, changing the policies used. In these complementary executions the policy DPWP was chosen when executing the homogeneous scenario and the round-robin policy was the option when executing both the partially and totally heterogeneous platforms.

Table 5 presents the runtime average in seconds for thirty Gauss-Jacobi parallel application executions, using the round-robin/PVM and the DPWP/AMIGO/PVM scheduling policies, and considering platforms: homogeneous, partially heterogeneous and totally heterogeneous. The values between parentheses indicate the complementary execution.

Table 5: Gauss-Jacobi parallel application runtime using round-robin and DPWP policies on three different platforms. Values between parentheses indicate the complementary executions to compare the correct scheduling done in each platform by *As/Hl*.

Schedule Policy	*Totally Homogeneous*	*Partially Heterogeneous*	*Totally Heterogeneous*
Round-Robin	5,1s	(17,6s)	(28,6s)
DPWP	(7,8s)	11,6s	23,7s

The round-robin policy in the homogeneous scenario presents a better performance when compared to the results from the DPWP. These results point-out a 53% performance loss, in this case. The DPWP spent more time to find target nodes to receive new processes, while the original round-robin policy distribute these same processes equally among the nodes, doing scheduling in a simple and efficient way. Since the platform used is homogeneous and there were not external interferences from other applications, concurring to the available resources, the DPWP is unnecessary and inefficient. In this case the *HL* metric is capable to prevent

the use of a higher computing cost scheduling policy (such as the DPWP).

The DPWP policy is more efficient than the round-robin policy when considering the partially heterogeneous platform. In this case the performance gain was around 51.7%. This is due to the heterogeneity presented in the platform, fact considered only by the DPWP. Again, the *HL* metric can choose properly the schedule policy.

The DPWP obtains a better performance when compared to the round-robin in the heterogeneous scenario, as expected. However, the difference between both executions was lower, with a DPWP performance gain just around 20.7%. This smaller difference, when comparing to partially heterogeneous platform, is due to the overload of 60% and 80% in four nodes available for the experiments. These four nodes are near to saturation and this causes a higher impact in the DPWP performance, due to its costs. In this scenario DPWP spent more time to find the correct nodes to use and how many processes each node should receive, when comparing to the round-robin policy.

6 Concluding Remarks

This work investigates the *HL* metric behaviour in real scenarios. The *HL* metric [3] was investigated considering static and dynamic perspectives and it was also used in the *As/Hl* algorithm. The *HL* metric presented excellent behaviors in our case studies, pointing out the platform heterogeneity for both static features (e.g.: CPU frequency, memory quantity or network) and dynamical features, these obtained from benchmarks.

The results obtained from the experiments conducted with *As/Hl* algorithm, show performance loss to 53% when using the wrong scheduling policy in relation to the platform heterogeneity level. They show also performance gains to 51.7% when using the correct scheduling policy. The investigations performed in this work confirm that the correct use of the heterogeneity level is essential to improve the platform performance, therefore, producing better benefits with lower costs for the end-user. They also show that the *HL* is efficient to represent the heterogeneity degree, flexible when considering different heterogeneity perspectives and easy to be used in the processes scheduling context.

Future works include studying the *HL* metric thresholds to indicate if a platform must be handled either as homogeneous or heterogeneous, also under different perspectives such as: CPU, memory, network and a mixing of them.

7 Acknowledgments

The authors would like to thank CAPES, CNPq and FAPESP, Brazilian funding agencies, for the financial support.

References

[1] Al-Jaroodi, J., Mohamed, N., Hong, J., and Swanson, D. Modeling parallel applications performance on heterogeneous systems. In *Int. Parallel and Distributed Processing Symposium*, pages 160.2–, Washington, DC, USA, 2003. IEEE Computer Society.

[2] Araujo, A. P. F., Santana, M., Santana, R. H. C., and Souza, P. S. L. Dpwp - a new load balancing algorithm. In *5th Int. Conference on Information Systems Analysis and Synthesis - ISAS'99*, Orlando, U.S.A., 1999.

[3] Branco, K., Santana, M., and Santana, R. H. C. A novel metric for checking levels of heterogeneity in distributed computer systems. In *Advances in Intelligent System and Robotic*. IOS Press, 2003.

[4] Curnow, H. J. and Wichmann, B. A. A synthetic benchmark. *Computer Journal*, 19(1):43–49, 1976.

[5] Geist, G. A., Beguelin, A., Dongarra, J. J., Jiang, W., Manchek, R., and Sunderam, R. Pvm 3 users guide and reference manual. Oak National Lab., 1994.

[6] Grosu, D. Some performance metrics for heterogeneous distributed systems. In *Proceedings of PDPTA'96*. Las Vegas, 1996.

[7] Linpack. www.math.utah.edu/software/linpack.html#documentation, Last access: 02/02/2011.

[8] Massie, Matthew, L., Chun, Brent, N., and Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.

[9] McCalpin, J. D. Memory bandwidth and machine balance in current high performance computers. IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, 1995.

[10] Mucci, P. J. and London, K. The cachebench report, 1998.

[11] Netperf. www.netperf.org/netperf/NetperfPage.html Last access: 02/02/2011.

[12] Petterson, D. A. *Computer organization and design: the hardware/software interface*. Elsevier/Morgan Kaufmann, third edition, 2005.

[13] Souza, P. S. L., Santana, M., and Santana, R. H. C. Amigo - a dynamical flexible scheduling environment. In *5th International Conference on Information Systems Analysis and Synthesis - ISAS'99*, 1999.

[14] Weicker, R. P. Dhrystone: a synthetic systems programming benchmark. *ACM Computing Surveys*, 27:1013 – 1030, 1984.

[15] Zhang, Z. and Seidel, S. Benchmark measurements of current upc platforms. In *19th IEEE International on Parallel and Distributed Processing Symposium*, 2005.

[16] Zhang, Z. and Yan, Y. Benchmark measurements of current upc platforms. In *7th IEEE Symposium on Parallel and Distributed Proceeding*, pages 25–34, 1995.