Scheduling Based on Process Behavior Analysis

Paulo H. R. Gabriel[1] **and Rodrigo F. de Mello**[1]

[1]Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Av. Trabalhador são-carlense, 400 – São Carlos – SP – Brazil

{phrg,mello}@icmc.usp.br

Abstract. *Process scheduling researches attempt to understand the dynamics of applications in order to improve resource allocation policies. Recent studies in this area have analyzed the behavior of single processes without considering how they interact with each other. This drawback motivated this paper, which proposes a new process scheduling approach based on how applications interact when competing for resources. This approach is based on concepts of dynamical systems theory which state that stabler organizations can be reached by means of perturbations in system components. First, we analyze process occupation variables to obtain behavioral states which represent a system component here. Next, we combine the execution of processes by prioritizing the one with higher estimated CPU load. The execution of processes is then interleaved according to their predicted workloads and the system tends to be stabler. Here, the term system refers to the combination of behavioral states of all processes. To validate this approach, we consider simulated scenarios representing different workloads of a computational environment, obtaining shorter execution times and, therefore, higher performance as result.*

Resumo. *Pesquisas na área de escalonamento têm procurado entender a dinamicidade de aplicações, a fim de melhorar políticas de alocação de recursos. Estudos recentes analisam o comportamento de processos individuais sem considerar como eles interagem entre si. Essa limitação motivou este trabalho, que propõe uma nova abordagem de escalonamento de processos com base na maneira como as aplicações interagem quando competindo por recursos. Essa abordagem é baseada em conceitos de sistemas dinâmicos segundo os quais organizações estáveis podem ser alcançadas por meio de pertubações nos componentes do sistema. Foram analisados processos com diferentes ocupações a fim de obter estados comportamentais que representam um componente do sistema. Em seguida, combinou-se a execução de processos, priorizando os de maior carga estimada de CPU. Intercala-se, portanto, a execução de processos de acordo com suas cargas de trabalhos preditas, o que tende a um sistema mais estável. No contexto deste artigo, o termo sistema se refere à combinação de estados comportamentais de todos os processos. Para validar a abordagem, considerou-se cenários simulados representando diferentes cargas de um ambiente computacional obtendo, como resultado, tempos de execução menores, ou seja, maior desempenho.*

1. Introduction

The interaction of multiple systems is a major research topic for different scientific areas [Kennedy et al. 2001, Alligood et al. 1996, González-Miranda 2004]. In Physics, for example, the synchronization of multiple out-of-phase harmonic oscillators has been studied and formalized. In Astronomy, Kepler equations are part of several studies to analyze the influences of planets' orbits on satellites, comets, etc. In Biology, the relationship of predators and preys can be described by Lotka-Volterra equations. Still in Biology, the behavior of swarms, such as birds migration and organization of ant and bee colonies has been studied in an attempt to understand emergent characteristics.

More recently, different branches of Computer Sciences have looked for inspiration in such studies. For example, bio-inspired meta-heuristics, like ant colony and particle swarm, have been successfully applied to solve several optimization and computational intelligence problems [Kennedy et al. 2001]. Still in this area, process scheduling researches have been attempting to understand the dynamics of applications to improve resource allocation policies [Mello and Yang 2009, Dodonov and Mello 2010]. The main studies in this subject consider the monitoring and analysis of application occupation variables (*e.g.,* CPU load, memory accesses, and hard disk utilization over time) in order to characterize behavioral states and, based on those, improve resource allocation. However, these studies analyze single processes and, consequently, there is no particular study to understand how they interact with each other[1].

The possibility of understanding the behavior of individual processes and also their interactions and influences has motivated this paper, which proposes a new process scheduling approach based on how applications interact when competing for resources. This approach is based on dynamical systems theory [Alligood et al. 1996, González-Miranda 2004] and considers how the behavior of different processes can be rearranged or combined over time in order to obtain stabler dynamical systems[2]. By finding a stabler representation for the combination of multiple processes, we attempt to optimize resource allocation over time. For example, by having two or more processes competing for the same CPU, we analyze occupation variables for every one and, thus, obtain behavioral states associated with each process. Such states represent, for instance, amounts of CPU, memory and hard disk consumed. Afterwards, we attempt to interleave the execution of both processes by prioritizing the one with higher estimated CPU load. This strategy tends to a stabler system, resulting in shorter execution times, meaning higher performance. All these steps are performed by our scheduling policy.

The remainder of this paper is organized as follows: Section 2, presents some related work, focusing on how predictions improve scheduling decisions; Section 3 address concepts of dynamical systems; Section 4 details our approach, presents experimental results and also the analysis; Conclusions and future directions are reported in Section 5.

[1]This interaction is related to how processes compete to obtain resources and modify one each others behavior.

[2]It is important to make clear that 'stabler' is a term related to dynamical systems which means there is less behavioral variation. Such variations could jeopardize system forecasting.

2. Related Work

Several studies have confirmed that application knowledge can improve scheduling decisions. Many of these researches confront features provided by a set of computational resources in order to optimize allocation and minimize the response time of processes. In this venue, [Ferrari and Zhou 1988] performed an experimental study considering some load balancing algorithms proposed in the literature. It starts looking for clues to confirm the costs related to resource allocation and decision making. Using analytical models and simulation, [Sevcik 1989] demonstrated that it is possible to substantially improve scheduling decisions using knowledge about applications.

Following this same research venue, [Devarakonda and Iyer 1989] developed one of the first studies employing a pattern recognition technique to predict the utilization of resources, such as CPU processing, input/output operations and memory accesses. They employed a k-means algorithm to identify states of resource occupation in a monoprocessor UNIX system. The authors, however, did not consider communication and synchronization costs. On the other hand, their study motivated further works focused on the use of historical information.

[Feitelson and Nitzberg 1995], for example, showed that the execution time of parallel applications can be estimated from repeated executions, indicating that historical behaviors of applications can be used to improve future scheduling decisions. The authors also analyzed production data and confirmed that approximately two thirds of applications are executed multiple times. In another study, [Feitelson et al. 1997] noticed that repeated executions of the same application tend to present similar patterns of resource utilization.

Other researchers, like [Gibbons 1997], [Smith et al. 2004] and [Downey 1997] focused their studies on the prediction of response times of parallel applications based on previous executions. [Gibbons 1997] made predictions examining categories of applications while [Smith et al. 2004] used search techniques, such as greedy search and genetic algorithms, to find historical information based on application similarity. They used such information to characterize and predict the behavior of new applications. The same methodology was followed by [Krishnaswamy et al. 2004], who proposed algorithms to estimate the process execution times.

On the other hand, [Downey 1997] estimated process execution times by modeling cumulative distribution functions for every application category. The author employed them to approximate the behavior of future applications. More recently, [Schopf and Berman 1999] proposed the use of stochastic processes to parametrize performance characteristics, such as bandwidth, available CPU, message size and operation accounting. Later, [Lee and Schopf 2003] adopted regression models to establish relationships between performance characteristics and execution times of past applications.

Still in terms of historical information, [Dinda 2001] studied the scheduling problem considering a different point of view: the resource behavior instead of the process one. The author assumed an auto-regressive model with sixteen coefficients to estimate the CPU load of computers. According to his policy, processes are allocated on computers at lower predicted loads. He concluded that this policy improves load balancing, which was also later confirmed by [Chunlin and Layuan 2009].

Looking for a more precise prediction, [Mello and Yang 2009] and [Dodonov and Mello 2010] considered dynamical systems concepts and nonlinear prediction techniques to model and predict process behaviors. [Mello and Yang 2009] monitored processes occupation variables over time and evaluated their similarity and recurrence of behavioral patterns. They calculated the required number of past variable observations to efficiently forecast their future occurrences (*e.g.*, CPU load, memory occupation, etc.). The authors presented a theoretical proof of efficiency for the estimation of the embedded and separation dimensions using real-world application traces. Such estimations are applied to Takens' immersion theorem [Takens 1980] in order to predict future process occupations. The study was extended by [Dodonov and Mello 2010], who proposed a framework to provide an on-line and adaptive behavior prediction mechanism for an efficient application scheduling based on the anticipation of communication events. Results obtained in heterogeneous environments confirm the efficiency of the prediction mechanism, which outperforms conventional scheduling policies.

All these studies have highlighted the relevance of using application knowledge in process scheduling. However, even works that attempt to obtain more precise predictions [Mello and Yang 2009, Dodonov and Mello 2010] are limited to the analysis of individual processes and do not consider the interaction of multiple processes and how it can affect the scheduling. In order to overcome this drawback, we here study and consider adaptations in the interaction of multiple processes to reach a stabler resultant system. Such stability, derived from dynamical systems [Alligood et al. 1996], improves the predictability of behavioral states. The next section describes the dynamical systems concepts considered in our approach.

3. Dynamical Systems

A dynamical system is composed of a set of possible states and a rule that determines its current state in terms of past ones. Mathematically, a dynamical system is described by $x_{n+1} = f(x_n)$, where $n \in B$ ($B \subset \mathbb{R}$) denotes time, $x : B \rightarrow \mathbb{R}$ represents the state of the system and $f : \mathbb{R} \rightarrow \mathbb{R}$ is the rule or evolution law [Alligood et al. 1996]. In this case, the rule defines the next state in function of past ones, therefore this system is characterized as deterministic. Besides deterministic systems, there is also another class, named stochastic, in which rules also involve random terms [Alligood et al. 1996]. In addition, such dynamical systems also rely on initial conditions. These conditions define the input values for the rule which, consequently, affect system outputs.

In order to illustrate a dynamical system, let us consider the Logistic map presented in Equation 1, which is traditionally used to model population growth over time. Let this map start with the following initial conditions $b = 3.8$ and $x_0 = 0.5$, for $t \in [0, 500]$ iterations.

$$x_{t+1} = b \cdot x_t \cdot (1.0 - x_t) \tag{1}$$

Figure 2(a) shows the Logistic map outputs given the presented conditions. By conducting a detailed analysis, one can conclude this function presents low recurrence, chaoticity and behavior instability [Alligood et al. 1996]. Therefore, it is difficult to predict such a system by using statistical methods. However, one can reconstruct this system in order to observe internal regularities and simplify its understanding. This reconstruction supports the estimation of the rule, which indicates how the system evolves over time. Consequently, by reconstructing a given set of outputs, one may obtain that particular system rule, understand and predict it.

[Whitney 1936] observed the possibility of reconstructing data in multidimensional spaces, applying the concept of differential manifolds. Mathematically, $M \subset \mathbb{R}^k$ is a manifold of dimension m if, for each point $p \in M$, there is a neighborhood $U \subset M$ of p and a homeomorphism $x : U \to U_0$. U_0 is an open set of $\mathbb{R}^m$, such that the inverse homeomorphism $x^{-1} : U_0 \to U \subset \mathbb{R}^k$ is an immersion of class C^∞. In summary, for each $u \in U_0$, the derivative $\frac{d(u)}{dx} : \mathbb{R}^m \to \mathbb{R}^k$ is biunivocal.

Figure 1 shows an example of a plane parametrization $\mathbb{R}^{m-1} \times \mathbb{R}_+$ to $\mathbb{R}^k$. Given a point q', we can, by means of $\phi_q : H_0 \to H \cap M$, find a corresponding point $q \in M$. This example illustrates the mapping of a point and its neighborhood in a space with a higher number of dimensions.

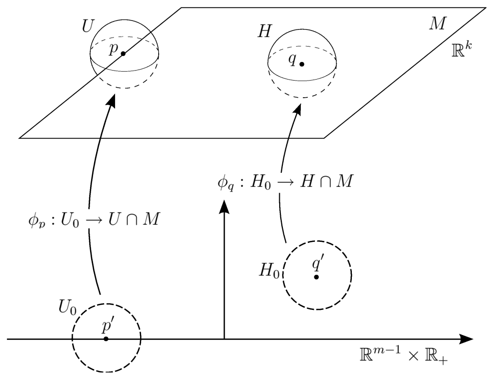
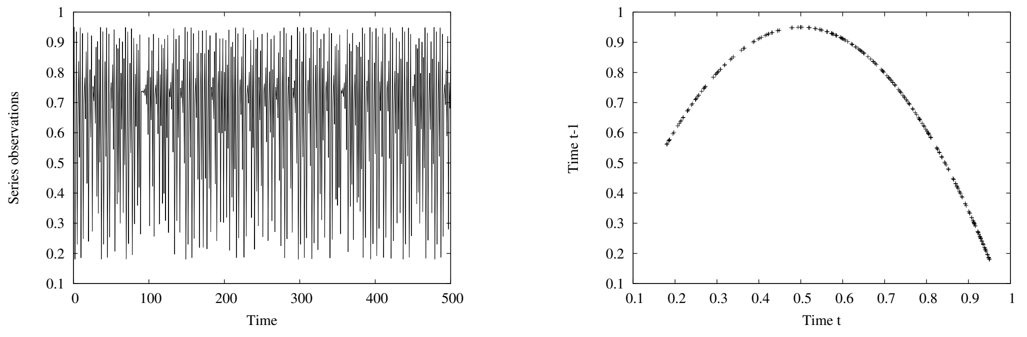

U
p
H
q
M
$\mathbb{R}^k$
$\phi_q : H_0 \to H \cap M$
$\phi_p : U_0 \to U \cap M$
H_0
q'
U_0
p'
$\mathbb{R}^{m-1} \times \mathbb{R}_+$

Figure 1. Example of a manifold.

This mapping allow understanding unobservable or underrepresented behaviors which are better described in a higher number of dimensions. From this study, [Whitney 1936] proposed his immersion theorem, whereby every trajectory in m dimensions can be mapped into a space with $2n + 1$ dimensions.

Based on such a theorem, [Takens 1980] proved that, instead of mapping states into a $2n + 1$-dimensional space, one can improve reconstruction considering time offsets. Thus, the outputs of a dynamical system, here seen as an one-dimensional time series $x_0, x_1, \ldots, x_{n-1}$, can be unfolded in a multidimensional (or phase) space in the form $x_n(m, \tau) = (x_n, x_{n+\tau}, \ldots, x_{n+(m-1)\tau})$, where m is the embedded dimension and τ represents the time delay. This theorem has been successfully employed to estimate dynamical system rules, thus, simplifying the behavioral studies and predictions [Alligood et al. 1996]

To illustrate the concepts of embedded dimension and time delay, we consider the output of the Logistic map (Equation 1) unfolded in a 2-dimensional space (*i.e.*, $m = 2$) and with $\tau = 1$, which results in pairs of points (x_t, x_{t+1}) (see Figure 2(b)). After such unfolding, we can observe the rule which generated the outputs and, consequently, study, understand and model other real-world problems by using Takens' immersion theorem. By making a regression of the resulting point pairs, we obtain the dynamical system rule and can determine future states.



(a) Logistic map outputs.

(b) Logistic map reconstructed with embedded dimension 2 and time delay 1.

Figure 2. Study on Logistic map.

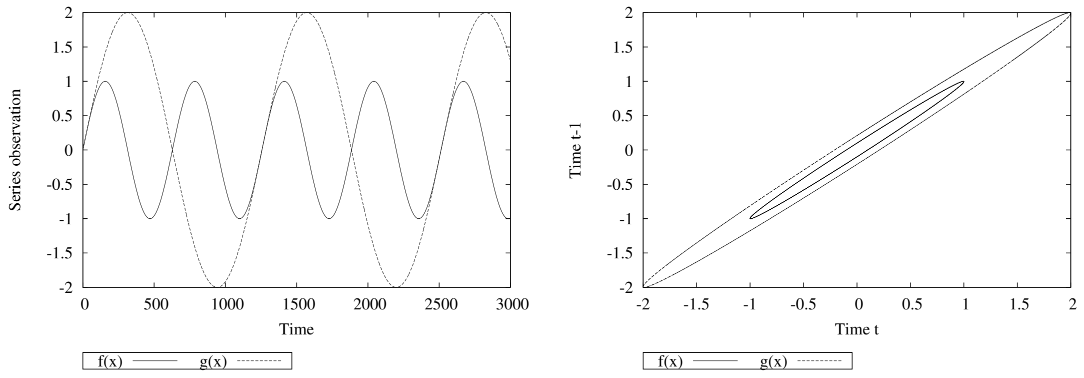
The embedded dimension defines the number of axis of the phase space, which are required to unfold the inherent behavior of systems. In this example, the series demands two axis, but other cases may require more (*e.g.*, the Lorenz map requires 3 axis [Alligood et al. 1996]). Several methods have been proposed to determine the embedded dimension as well as the time delay. The most accepted ones are the False Nearest Neighbors [Kennel et al. 1992] for the embedded dimension and the Auto-Mutual Information [Fraser and Swinney 1986] for the time delay.

In this paper we employ such methods to estimate embedded dimensions and time delays in order to improve the reconstruction of system outputs in multidimensional spaces. By estimating dynamical system rules, we are able to understand and make predictions for a given system. Systems of interest here are processes and their outputs are composed of occupation variables which change over time. As we are particularly interested in how CPU load varies over time, we monitor that variable for every process and, then, reconstruct each process in a different multidimensional space, obtaining the rule for every process which determines its CPU occupation states. The states of different processes may influence one each other, reducing and/or improving performance. This paper focuses essentially on how to modify such influences in order to make processes run faster.

4. The Proposed Approach

This paper was motivated by influences and interactions of multiple dynamical systems [González-Miranda 2004]. For example, a system A may modify the behavior of another system B as they compete for resources. Understanding such influences, we claim that it is possible to modify the points of interaction among them and, thus, improve performance.

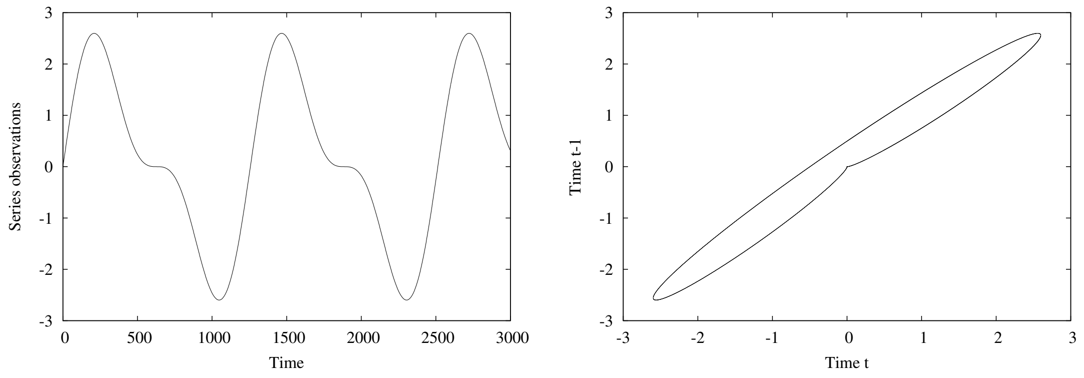
In order to illustrate the main ideas that motivated this study, consider two dynamical systems represented by the following rules: $f(x) = \sin(x)$ and $g(x) = \sin(x/2.0 + y) \cdot 2.0$. The outputs of such systems are shown in Figure 3(a). Also consider that the person who obtained those outputs does not know the rules, thus he/she needs to obtain them by using the procedure described in Section 3. One will reconstruct the outputs by computing the False Nearest Neighbors and the Auto-Mutual Information methods and apply the resultant embedded dimension and time delay to Takens' immersion theorem [Takens 1980], obtaining the reconstructions presented in Figure 3(b).



(a) Functions $f(x)$ and $g(x)$. (b) Attractors of $f(x)$ and $g(x)$ functions.

Figure 3. Functions $f(x)$ and $g(x)$ and their attractors.

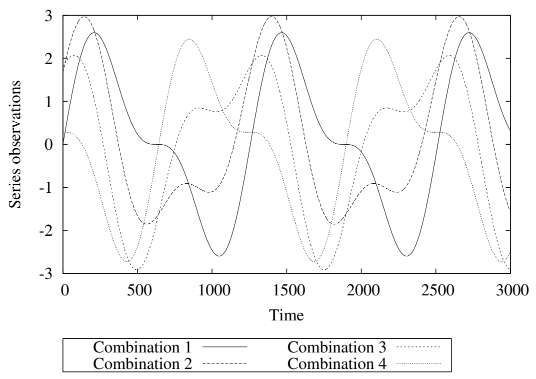
Now consider that both systems are combined through a simple sum of observations. This combination results in a third system which represents the same computer receiving both processes at the same time; the scheduler will then attempt to run both together. This third system is characterized by function $h(x) = \sin(x)+\sin(x/y+2.0)+2.0$, whose outputs are illustrated in Figure 4(a). The reconstructed behavior of this third system (Figure 4(b)) has a region with a behavior different from that previously observed when evaluating single processes. This reconstruction confirms that the combination of systems forces their interactions in terms of resource utilization, modifying the reconstructed behavior, which will also be called dynamical system attractor or simply attractor from now on.



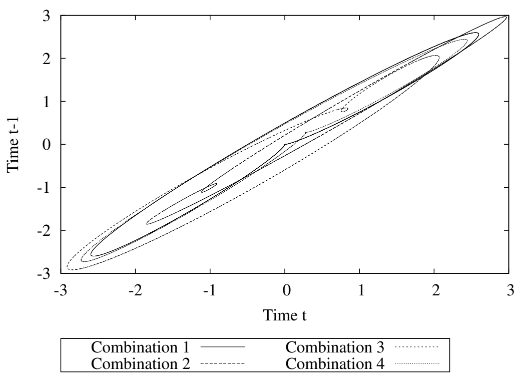
(a) Function $h(x)$. (b) Attractor of $h(x)$ function.

Figure 4. Function $h(x)$ and its attractor.

Additional combinations of functions $f(x)$ and $g(x)$ are illustrated in Figure 5(a). We observe that the attractors (Figure 5(b)) show different behavior for different combinations, motivating the search for situations in which, by combining distinct functions in different ways (*i.e.*, in terms of time displacements), there is an equilibrium between these functions, generating stabler attractors and, therefore, a more uniform behavior.



(a) Offsets of $f(x)$ and $g(x)$



(b) Attractors of the offsets.

Figure 5. Offsets of $f(x)$ and $g(x)$ and their attractors.

To apply this study in the context of process scheduling, we have proposed an analytical model to evaluate combinations (in terms of time displacements) considering the behavior of two processes characterized by the same function: $p(x) = q(x) = 50 \cdot \sin(x) + 50$. Through offsets in x-axis (which represents time), different attractors were obtained, allowing assessing and obtaining stabler situations. Figure 6 shows the response time according to such a model.

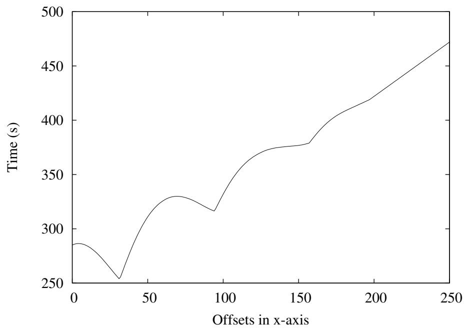
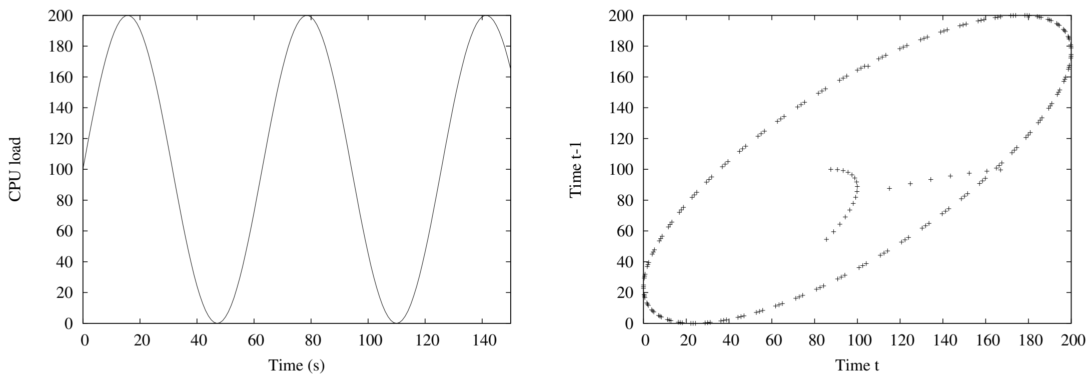


Figure 6. Response times.

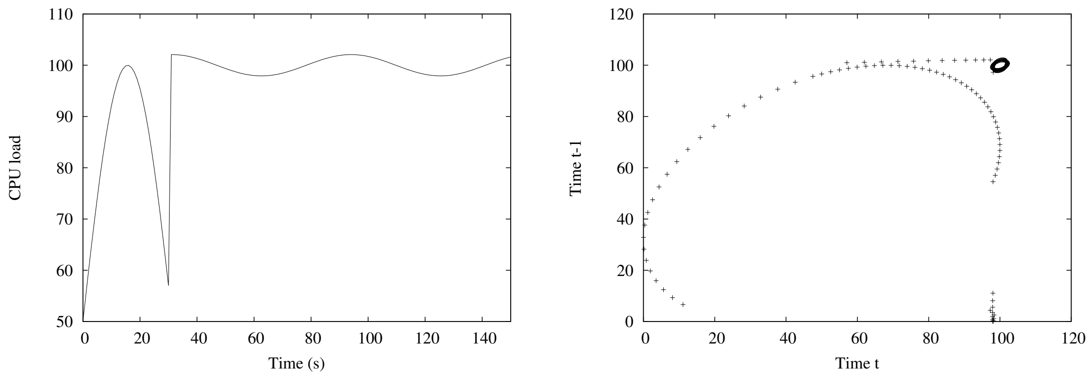
We can observe that, for certain offsets in x, the response time is significantly lower, providing evidence that some combinations of $p(x)$ and $q(x)$ result in a more appropriate scheduling in terms of makespan (makespan represents the total execution time of an application). Figures 7, 8 and 9 illustrate the behavior of combinations of $p(x)$ and $q(x)$ varying offsets in x. The respective attractors are also shown.

By considering this example, we observe that a stabler attractor has been obtained by using offset 31 on x-axis, *i.e.*, the combined behavior of processes tends to a region of higher stability. Such stability is observed, in terms of dynamical systems [Alligood et al. 1996], because there is a slight variation in the resulting attractor formed by the combination of both systems, as seen in Figure 8(b). In different words, the states of the resulting system are more concentrated in a specific region. On the other hand, when such combination results in unstabler systems, we observe novel orbits being formed such as in Figure 9(b) (two well-defined concentric orbits). The presence of more orbits indicates the system has higher probability to vary, making prediction more complex. This



(a) Offset 0. (b) Attractor to offset 0.

Figure 7. Combination of $p(x)$ and $q(x)$ with offset 0.



(a) Offset 31. (b) Attractor to offset 31.

Figure 8. Combination of $p(x)$ and $q(x)$ with offset 31.

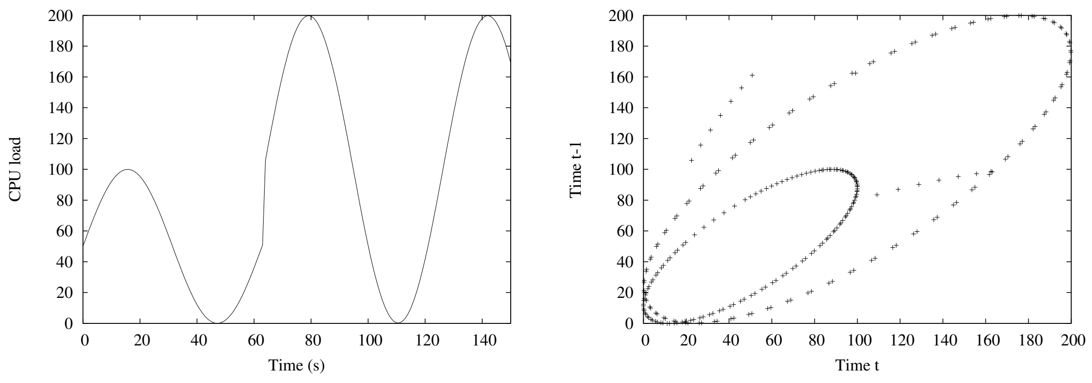
offset corresponds to the point in Figure 6 whose response time is minimal.

Based on these initial observations, we designed and implemented two heuristics in a simulator. By considering series collected during the execution of real applications, they schedule processes according to equilibrium situations.

4.1. Proposed Heuristics and Results

The first heuristic proposed, Heuristic A, considers the temporal behavior of the processes and performs the scheduling based on their predicted CPU usage. This heuristic assumes the prediction of the processes behavior in a time slice (after the current moment). For example, considering a series $S = \{x_0, x_1, \ldots, x_{n-1}\}$ with n observations, we perform the prediction of the next observation x_n based on the approach proposed in [Mello and Yang 2009]. Based on such a prediction, Heuristic A schedules the process with higher CPU utilization at the next observation x_n. Thus, it maximizes the processor utilization and reduces the response time of processes.

By prioritizing processes with higher CPU usage, Heuristic A tends to delay the execution of others with lower demands. To avoid this limitation, we have proposed a variation of the first heuristic which introduces a probabilistic term to select the next process to run. This term is proportional to the next observation x_n for every process and



(a) Offset 64.

(b) Attractor to offset 64.

Figure 9. Combination of $p(x)$ and $q(x)$ with offset 64.

is computed as follows. Let P_i be the probability of a process i to be selected, then

$$P_i = \frac{x_n^i}{\sum_{j=1}^{N} x_n^j},$$

where x_n^i is the next observation of process i, that is, the predicted CPU usage. For instance, given a set of three processes running, assuming that the workload of every process in the next time slice is $\{x_n^1 = 13.96, x_n^2 = 6.53, x_n^3 = 2.27\}$, the probability of selecting each one to execute in the next time slice is given by $\{P_1 = 0.61, P_2 = 0.28, P_3 = 0.11\}$. Thus, even processes that demand less resources may also be selected. This new heuristic has been called Heuristic B.

In order to validate both heuristics, we considered five scenarios representing different environment workloads. Such scenarios were designed based on execution traces[3] of the tool `grep` and also on a `C`-language version of the `Fibonacci` algorithm.The tool `grep` was executed in two different ways:

```
grep -r "?" /,
```

which prints the result in standard output, alternating computation with I/O operations, and

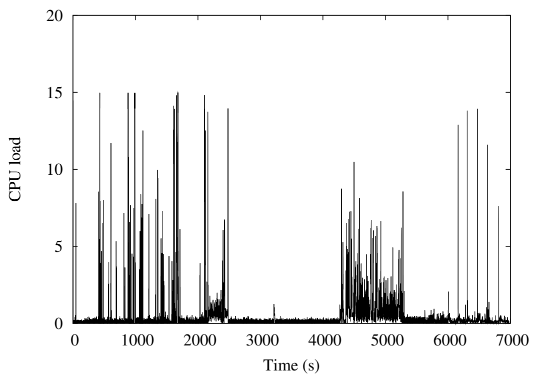
```
grep -r "?" / > /dev/null,
```

which avoids output operations, increasing CPU usage. When using `Fibonacci`, no output was generated.

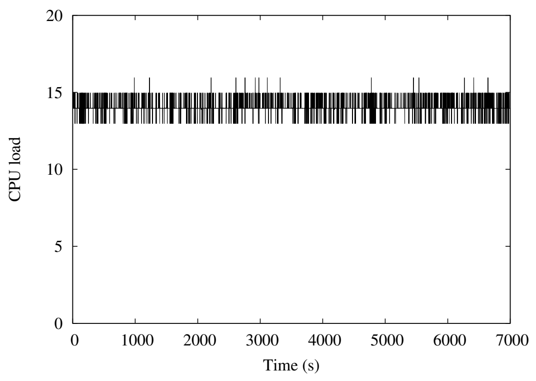
Figures 10(a) and 10(b) show the CPU load for `grep` and `Fibonacci`, respectively. For `grep`, we observe there is an alternated pattern of CPU utilization and output operations, while `Fibonacci` demands a higher and constant CPU usage.

Based on the obtained traces (shown in Figure 10), the heuristics were simulated and compared against the *Round-Robin* (RR) policy. Only for clarification purposes, RR assigns identical CPU times for each process, which are organized in a circular queue.

[3]We developed a script to run on GNU/Linux to obtain CPU usage at every 150 ms, that reads file `/proc/id/stat` – `id` corresponds to the process identifier.



(a) CPU usage for tool `grep`.



(b) CPU usage for `Fibonacci` algorithm.

Figure 10. CPU usage.

Table 1 presents the results for five `grep` processes simultaneously started in the computational environment, thus competing for resources. In this case, we assume the execution of `grep` generating outputs. We here compare the total execution time (makespan), the process average running time, the time necessary to switch contexts (CS), that is, the total time the scheduling policy requires to select the next process to run, and, finally, the average CPU utilization.

Table 1. Five identical executions of `grep`.

Policy	Total Time	AVG Time	CS Time	AVG Utilization
RR	33,440.0	6,688.0	13.34	1.90
Heuristic A	34,680.0	6,936.0	1,82	1.83
	Performance Gain			-3.57%
Heuristic B	33,769.0	6,753.0	14.00	1.88
	Performance Gain			-0.96%

In this case, RR presented better results, since all processes impose the same workload and, thus, a simply execution interleaving tends to better results, in opposition to strategies proposed by both heuristics evaluated. We can also observe that Heuristic B reduced response time and improved resource usage when compared to Heuristic A.

In a second scenario, we considered three identical executions of `Fibonacci` algorithm. In such a circumstance, we obtained a similar behavior for all policies (Table 2), due to the high CPU utilization imposed by every `Fibonacci` process.

Table 2. Three identical executions of `Fibonacci` algorithm.

Policy	Total Time	AVG Time	CS Time	AVG Utilization
RR	21,347.0	7,115.0	4.27	46.73
Heuristic A	21,347.0	7,115.0	4.27	46.73
	Performance Gain			0.00%
Heuristic B	21,347.0	7,115.0	4.27	46.73
	Performance Gain			0.00%

These two scenarios are composed of processes with identical behaviors; in such a situation our heuristics do not improve performance. However, when a different set of processes is executed, the heuristics indeed affect the final performance. For example, Table 3 presents four `grep` processes, writing in the standard output, and one execution of `grep` redirecting the output to the null device (as previously shown). We then observe that Heuristic A reduced the response time to approximately 52% when compared to RR and also improved the resource utilization. The performance of Heuristic B is slightly below A, but still better than RR.

Table 3. Five executions of `grep` (four identical and one different).

Policy	Total Time	AVG Time	CS Time	AVG Utilization
RR	18,324.0	3,664.0	6.98	3.07
Heuristic A	8,814.0	1,762.0	3.07	3.75
	Performance Gain			+51.91%
Heuristic B	9,493.0	1,898.0	3.19	3.64
	Performance Gain			+48.19%

In another scenario, we simulated higher demands for CPU by running three `grep` processes, writing in the standard output, one `grep` execution without outputs and, finally, one `Fibonacci` process. In this case, Heuristics A and B outperformed RR in terms of response time (reducing it in about 31% and 28%, respectively) and CPU utilization (Table 4).

Table 4. Four executions of `grep` (three identical and one different) and one execution of `Fibonacci` algorithm.

Policy	Total Time	AVG Time	CS Time	AVG Utilization
RR	22,382.0	4,476.0	6.98	16.81
Heuristic A	15,448.0	3,089.0	4.15	23.61
	Performance Gain			+30.98%
Heuristic B	16,005.0	3,201.0	4.27	22.83
	Performance Gain			+28.49%

Finally, in the last scenario (Table 5), we increased CPU demands by adding another `Fibonacci` process in the previous scenario. In this case, performance was also improved by Heuristics A and B, however, such improvement was lower than the previous scenario because, by adding another `Fibonacci` process, the total CPU workload was too high to be treated by a single computer. If we add more processes, there will even be reductions in the resulting improvement, tending, in overloaded situations, to the same performance of RR.

Table 5. Four executions of `grep` (three identical and one different) and two execution of `Fibonacci` algorithm.

Policy	Total Time	AVG Time	CS Time	AVG Utilization
RR	29,364.0	4,894.0	8.38	23.92
Heuristic A	22,564.0	3,760.0	5.58	30.90
	Performance Gain			+23.17%
Heuristic B	23,064.0	3,844.0	5.65	30.26
	Performance Gain			+21.45%

We have concluded that the proposed heuristics improve both performance and resource usage when heterogeneous processes are submitted to the environment. Another important point is that when processes barely use CPU, our heuristics do not provide improvements (Tables 1 and 3). A second situation occurs when processes impose too heavy workloads; in such scenario, heuristics tend to present results closer to RR. However, when processes present a different behavior and impose neither too heavy nor too low CPU usage, both heuristics strongly improve performance results as well as resource utilization. This last scenario is typical in desktop computers. Furthermore, we can also employ such heuristics in grid computing environments by avoiding the overload of single computers and making such heuristics responsible for local scheduling.

5. Conclusions

This paper has presented studies on behavioral interactions of processes, motivating the proposal of two heuristics which allocate resources based on how processes demand CPU in future time slices. Both heuristics were implemented in a simulator and compared against the Round-Robin policy.

Results have confirmed that, when processes demand the same or similar CPU usage, both heuristics are slightly inferior to Round-Robin. Still, when processes are similar, but under higher CPU utilization, the heuristics as well as Round-Robin present similar performance. Finally, when processes have different behaviors, both heuristics outperform Round-Robin in terms of reducing the total execution time and improving CPU utilization.

Furthermore, we remember that many computational environments are mostly characterized by processes with different behavior, as desktop systems, confirming the potential usage of such heuristics in real-world scenarios. Even large-scale environments, such as grids, could take advantage of the proposed heuristics by using them as local scheduling policies.

Acknowledgments

This paper is based upon work supported by FAPESP (São Paulo Research Foundation), Brazil, under grant no. 2009/15338-1. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of FAPESP.

References

Alligood, K. T., Sauer, T. D., and Yorke, J. A. (1996). *Chaos: An Introduction to Dynamical Systems*. Springer-Verlag.

Chunlin, L. and Layuan, L. (2009). A system-centric scheduling policy for optimizing objectives of application and resource in grid computing. *Computers and Industrial Engineering*, 57(3):1052–1061.

Devarakonda, M. V. and Iyer, R. K. (1989). Predictability of process resource usage: A measurement-based study on UNIX. *IEEE Transactions on Software Engineering*, 15(12):1579–1586.

Dinda, P. A. (2001). Online prediction of the running time of tasks. In *IEEE International Symposium on High Performance Distributed Computing*, pages 383–382.

Dodonov, E. and Mello, R. F. (2010). A novel approach for distributed application scheduling based on prediction of communication events. *Future Generation Computer Systems*, 26(5):740–752.

Downey, A. B. (1997). Predicting queue times on space-sharing parallel computers. In *International Symposium on Parallel Processing*, pages 209–218.

Feitelson, D. G. and Nitzberg, B. (1995). Job characteristics of a production parallel scientific workload on the NASA ames iPSC/860. In *Job Scheduling Strategies for Parallel Processing*, volume 949 of *LNCS*, pages 337–360. Springer-Verlag.

Feitelson, D. G., Rudolph, L., Schwiegelshohn, U., Sevcik, K. C., and Wong, P. (1997). Theory and practice in parallel job scheduling. In *Job Scheduling Strategies for Parallel Processing*, volume 1291 of *LNCS*, pages 1–34. Springer-Verlag.

Ferrari, D. and Zhou, S. (1988). An empirical investigation of load indices for load balancing applications. In *International Symposium on Computer Performance Modelling, Measurement and Evaluation*, pages 515–528.

Fraser, A. M. and Swinney, H. L. (1986). Independent coordinates for strange attractors from mutual information. *Phys. Rev. A*, 33(2):1134–1140.

Gibbons, R. (1997). A historical application profiler for use by parallel schedulers. In *Job Scheduling Strategies for Parallel Processing*, volume 1291 of *LNCS*, pages 58–77. Springer-Verlag.

González-Miranda, J. M. (2004). *Syncronization and Control of Chaos: An Introduction for Scientists and Engineers*. Imperial College Press.

Kennedy, J., Eberhart, R. C., and Shi, Y. (2001). *Swarm Intelligence*. Morgan Kaufmann Publishers.

Kennel, M. B., Brown, R., and Abarbanel, H. D. I. (1992). Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Phys. Rev. A*, 45(6):3403–3411.

Krishnaswamy, S., Loke, S. W., and Zaslavsky, A. (2004). Estimating computation times of data-intensive applications. *IEEE Distributed Systems Online*, 5(4):1–12.

Lee, B.-D. and Schopf, J. M. (2003). Run-time prediction of parallel applications on shared environments. In *IEEE International Conference on Cluster Computing*, pages 487–491.

Mello, R. F. and Yang, L. (2009). Prediction of dynamical, nonlinear, and unstable process behavior. *The Journal of Supercomputing*, 49(1):22–41.

Schopf, J. M. and Berman, F. (1999). Stochastic scheduling. In *ACM/IEEE Conference on Supercomputing*.

Sevcik, K. C. (1989). Characterizations of parallelism in applications and their use in scheduling. *Performance Evaluation Review*, 17(1):171–180.

Smith, W., Foster, I., and Taylor, V. (2004). Predicting application run times with historical information. *Journal of Parallel and Distributed Computing*, 64(9):1007–1016.

Takens, F. (1980). Detecting strange attractors in turbulence. In *Dynamical Systems and Turbulence*, pages 366–381. Springer.

Whitney, H. (1936). Differentiable manifolds. *The Annals of Mathematics*, 37(3):645–680.