Formalization of Autonomic Heuristics-Driven Systems[*]

From a Heuristics to its Autonomic Counterpart

Marco Pasquali
HPC Lab – ISTI/CNR
IMT – Lucca
Italy
marco.pasquali@isti.cnr.it

Patrizio Dazzi
HPC Lab – ISTI/CNR
IMT – Lucca
Italy
patrizio.dazzi@isti.cnr.it

ABSTRACT
In this paper we show the fundamental elements needed to make autonomous a heuristics-based system. It is made by using a formal notation. The proposed method allows heuristics designers to make autonomous a heuristics by exposing its state and the polytope in which the heuristics works properly. The proposed model was evaluated applying it to a real scenario in which a dynamic stream of batch jobs is classified according to job deadline QoS constraint. The conducted tests point out that the proposed autonomic heuristics can be fruitfully exploited.

Categories and Subject Descriptors
F.1.1 [**Computation by abstract devices**]: Models of Computation—*Self-modifying machines, Unbounded-action devices*

General Terms
Autonomic Computing, Adaptive Heuristics

Keywords
autonomicity, higher order functions, heuristics

1. INTRODUCTION
The *Autonomic Computing* is an initiative started by IBM in 2001 (aka ACI [11]). Its ultimate aim is to create self-managing computer systems to overcome their rapidly growing complexity and to enable their further growth. It takes inspiration from the autonomic nervous system of the human body. The nervous system controls important biological functions (e.g. respiration, heart beat, and blood pressure) without any conscious intervention.

The ACI focuses on the definition of foundations for autonomic systems and, in particular, on the definition of elements that are fundamental to make computing system autonomous. In a self-managing Autonomic System, the human operator acquires a new role: She does not control the system directly, instead she defines general policies and rules that are used as an input for the self-management process.

Our work describes, using a formal notation, what are the fundamental elements needed to make autonomous a heuristics-based system. The aim is to provide to the system mechanisms to recognize mismatches between actual and expected system behavior (with respect to its internal state), and to provide to it the capability to adapt itself.

Our approach is not intended to be too abstract nor too formal. We describe with a high-level notation the elements that a heuristics designer needs to care about designing an autonomous heuristics. The challenge is to provide technique that enables software systems to evolve in order to remain useful [12], but to do so in a way that does not incur downtime as traditional maintenance processes do [16].

The paper starts with the formalization of heuristics-driven systems (Section 2). First we describe a not autonomous heuristics, i.e. that does not adapt itself to system changes, then we present the requirements to make it autonomous. Moreover, we discuss how continuous heuristics-state-changes can be made discrete to model more realistic situations. In Section 3, we present a case study in which we transform a classification heuristics in an autonomic one. Such process is led by the proposed formalization. In Section 4 we present the experiments conducted and the related results. In Section 5 related works are described. Finally, in Section 6 we draw our conclusions and the path of future work.

2. AUTONOMIC HEURISTICS-DRIVEN SYSTEMS
In this section we want to formalize two concepts: *heuristics* and *autonomic heuristics*-driven systems.

A *heuristics*-driven system S uses its internal state (s) to transform the inputs (I) in outputs (O), as depicted in Fig-

[*]This work has been supported by SUN Microsystems's grant, and it is in the activity of the European CoreGRID NoE (European Research Network on Foundations, Software Infrastructures and Applications for Large Scale, Distributed, GRID and Peer-to-Peer Technologies, contract no. IST-2002-004265).

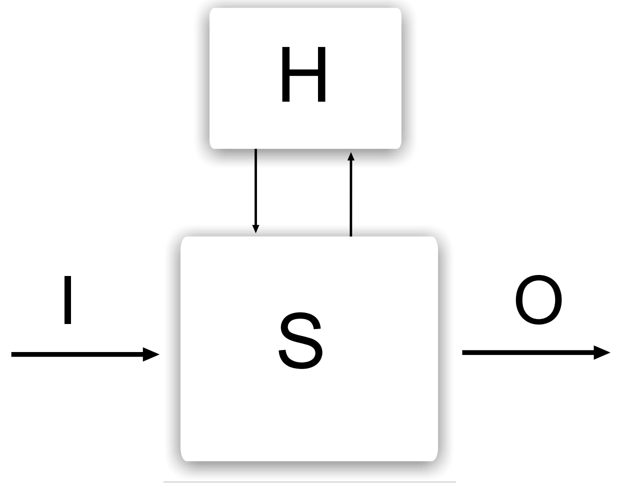

Figure 1: Heuristics-driven systems

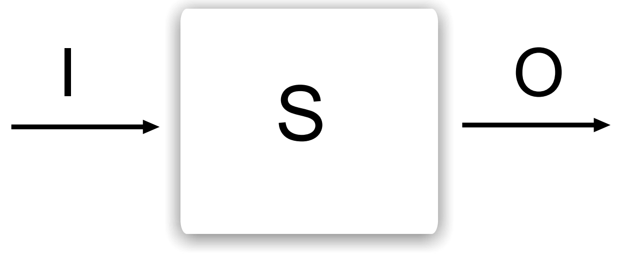

Figure 2: Autonomic heuristics-driven system

ure 1. We can define its behavior as the sequence of execution of two functions:

$$\begin{aligned} f^o &: I \times s \rightarrow O \\ f^\star &: I \times O \times s \rightarrow s'. \end{aligned}$$

f^o defines the behavior of the system, transforming inputs in outputs using the internal state as one of its parameters. $f^\star$ is the function acting on the system: Using the inputs, the computed outputs and the current system internal state, it generates a new system state (s').

An *autonomic heuristics*-driven system S is characterized by an external heuristics H (Figure 2). Both S and H have different internal states s_s and s_h. As the heuristics-driven system, its behavior can be defined as the sequence of execution of two functions:

$$\begin{aligned} f^o &: I \times s_s \rightarrow O \\ f^\star &: I \times O \times s_s \times s_h \rightarrow s'_s \times s'_h. \end{aligned}$$

In this second case, f^o behaves exactly as the one defined for the heuristics-driven systems, instead $f^\star$ behaves in a different way: Using the inputs, the computed outputs, the current internal state of the system and the state of the heuristics, it generates a new state (s'_s) for the system and a new state (s'_h) for the heuristics.

This division enable the application designer to disjoin the system and the heuristics that drives the system itself. In this way the system can be *adapted* to new scenarios unplugging the current heuristics and plugging-in another one.

2.1 Continue and Discrete Heuristics-Driven Systems

The two systems, we previously defined can be categorized as *continuous*: For each input the system evaluates a state change.

More realistic systems are characterized by value ranges in which the internal state of the system is free to move without changing the whole system behavior. We refer to these systems as *discrete*.

To formalize the concept of discrete heuristics-driven system, we need to consider the following abstraction: The state of the system is a point in a geometric space and the value range in which it is free to move is a polytope in the same geometric space.

This implies that the internal state (s_s) of the system using autonomic heuristics must be transformed in the pair (s_s, $\mathcal{P}$), where s_s is the new representation of the state as a point in the space and $\mathcal{P}$ is the polytope that represents the bounds in which s_s is free to move without changing the system behavior.

We can define the polytope $\mathcal{P}$ of heuristics h for the system s as the union of system states in which h has a acceptable behavior. Formally:

$$\mathcal{P} = \{s_s \mid Acceptance_H(s_s)\}$$

where $Acceptance_H$ is a boolean function that returns true if the heuristics's behavior is acceptable.

The $f^\star$ function must be changed to use the new system state representation:

$$f^\star : I \times O \times (s_s,\ \mathcal{P}) \times s_h \rightarrow (s'_s,\ \mathcal{P}') \times s'_h.$$

As depicted in figure 3.a the state is represented as a point s inside a polytope $\mathcal{P}$. When $f^\star$ is computed, the state change can be represented as a vector v in space (Figure 3.b) and the state is modified to a new value s'. If the new state is outside the polytope $\mathcal{P}$, the behavior of the heuristics, and consequently the behavior of the system, must be changed according to the new state. Then a new polytope is computed to bound the new state (Figure 3.c).

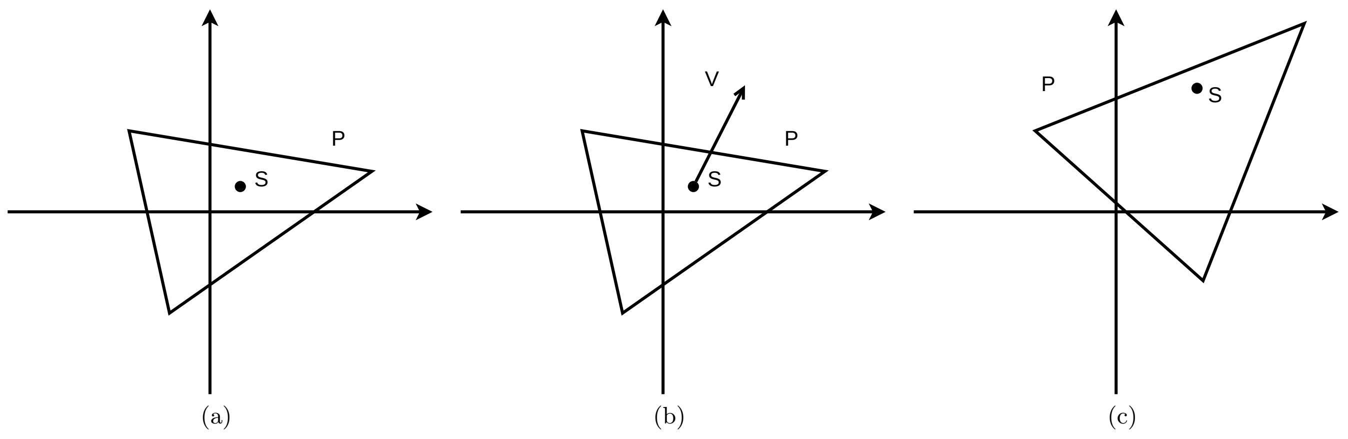

Figure 3: Discrete autonomic heuristics evolution. In (a) the autonomic heuristics is identified by polytope $\mathcal{P}$ and status s. In (b) a new input generates the transition vector v that moves the status outside the polytope. In (c), after the application of function $f^{\star}$, the (autonomic-)heuristics changes both state and polytope to s' and $\mathcal{P}'$.

The number of dimensions in which the polytope and the state are defined depends on the number of heuristics configuration freedom degrees.

3. CASE STUDY

In this section we describe how to apply the proposed formalization model to a real scenario concerning the scheduling of batch jobs on computing farms according to jobs QoS constraints.

In general, the objective of a job scheduler is twofold: to optimize both the system throughput and the applications performance. This means that the scheduler tries to maximize the overall resource utilization guaranteeing the required level of QoS to applications. Several job parameters describing the hw/sw job requirements, the job execution estimation time, the job deadline (the time at which the job must be completed), and other information such as the kind of user requiring the job execution, are used by the job schedulers to implement their scheduling policies.

To this end we propose an autonomic classification heuristics that we developed in the context of "Scheduling under the Sun" [1] research project. Our heuristics can be used in online [14] job classifiers receiving jobs from users, classifying the jobs and eventually sending them to a scheduling system. In our system, as depicted in Figure 4, the classifier is the front-end and the scheduler is the back-end: The first one is interfaced with users whereas the latter deal with physical resources.

The goal of our job classifier is to assign a priority value to each submitted job. The priority value defines a *job total-order of execution.* It is computed when a job is submitted to the system. The job priority is a function of only job's parameters and it does not consider system information, such as: The number and the type of machines, the software licenses availability, and the machines workload. The goal of our system is to exploit job attributes/characteristics and metrics to enable a job classification in an independent way with respect to the features of the computing platform used.

[1]The project "Scheduling under the SUN" have as objective the design and the evaluation of scalable approaches to dynamically schedule a stream of batch jobs in a large-scale Grid for utility computing

To validate our approach we consider a simple job classifier only based on the job deadline parameter.

To this end, we introduce a heuristics, called *DeadLine*, which is used to implement a classifier, which exploits the job deadline parameter, according to the description given in section 3.1. This choice was driven by our will of implement a simple case of study in which operates only one heuristics. In this way, it is simple to recognize bad heuristics behaviors, and, it also makes easier to understand how the autonomic model can change the heuristics state.

3.1 DeadLine Heuristics

The *DeadLine* heuristics characterizes a job with respect to its deadline. It considers a time value called *margin*, defined as the difference between the time at which the job execution must start in order to respect its deadline, and the job submission time (Figure 5).

To compute the priority value of a job i, i.e. $\Delta_{p,i}$, *DeadLine* computes the average margin value considering the margin values of all the jobs analyzed before the submission of job i. We consider a job to be a "proximity" one if its margin value is smaller than the average margin value, vice versa it is considered to be a "faraway" one.

To compute $\Delta_{p,i}$, the heuristics considers the double value of average margin. It permits to over estimate the width of the interval in which a job is a proximity one.

DeadLine is a configurable heuristics. Indeed, the administrator of the heuristic-driven system can choose the job classification distribution policy. The policy can be used to express the heuristics required behavior [3]. Such policies can be used to specify the expected behavior of the autonomic heuristics (Section 3.2). We chose a exponential dis-

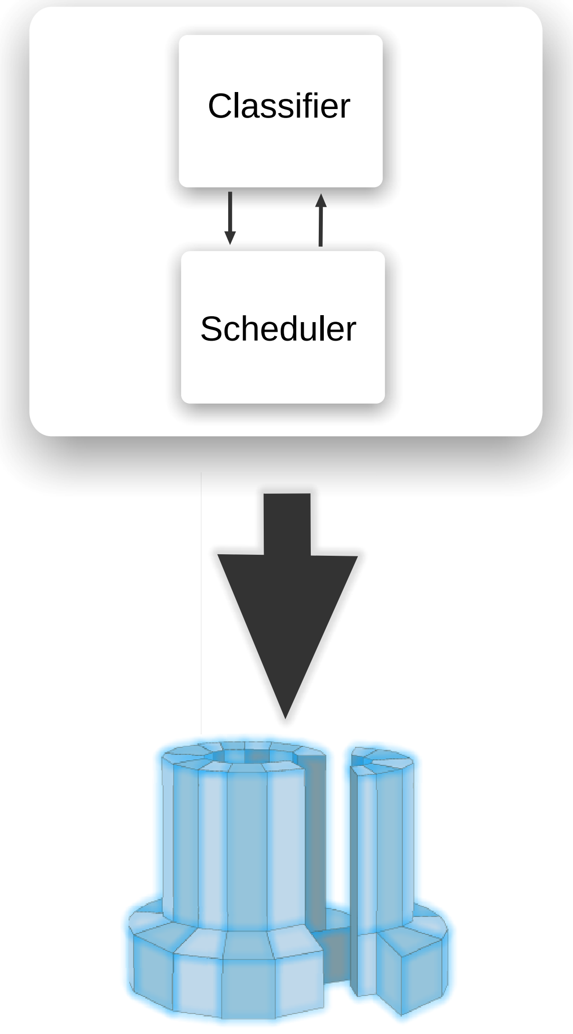

Figure 4: System architecture

tribution policy, i.e. a distribution in which the number of jobs with priority p is exponentially greater than the number of jobs with priority $p-1$. This allows the system to strongly limit the number of jobs characterized by the highest priority, permitting to better satisfy the requirements of the jobs.

The following relation formalize the priority distribution we used to configure the *DeadLine* heuristics: let i and j be two integer representing two different priority values with $i < j$, let $\#job_p$ the number of job with priority equals to p, then our scheduler behaves at best when:

$$\#job_j = \frac{\#job_i}{2^{j-i}}. \tag{1}$$

In order to assign the priority values to the jobs using the deadline time following the described distribution, the interval $[0, 2 * E[margin]]$ is divided in subintervals as depicted in Figure 6.

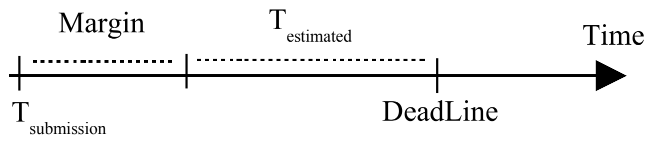

Figure 5: Graphical representation of the job submission, estimation, margin and deadline times.

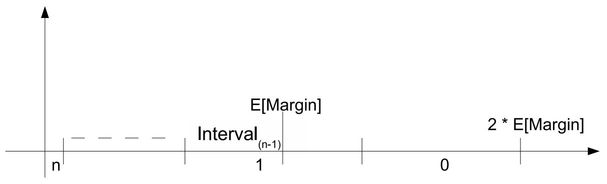

Figure 6: Graphical representation of the margin value.

The subintervals are computed as:

$$interval_{(max-k)} = [S_k, S_{k+1}] \text{ with } k = 0, ..., max$$

where max is the highest priority value that a job can assume, and where:

$$\begin{cases} S_0 = 0 \\ S_k = S_{k-1} + MinUnity \cdot 2^k \end{cases} \tag{2}$$

where $MinUnity$ is

$$MinUnity = \frac{2 * E[margin]}{\sum_{k=1}^{max} 2^k} \tag{3}$$

3.2 Autonomic DeadLine Heuristics

We tested the *DeadLine* heuristics performing experiments and it showed to have good performances in a lot of different situation (see the tests performed in Section 4). We also tested it in a critical situation, namely when the job-stream is logically divided into distinct sub-streams ($substream_1, ..., substream_n$) containing jobs with a very similar margin, but consecutive sub-streams have very different average margin values. Figure 7 depicts this situation: On the x-axis are represented job sub-streams and on the y-axis are represented the job average margin time.

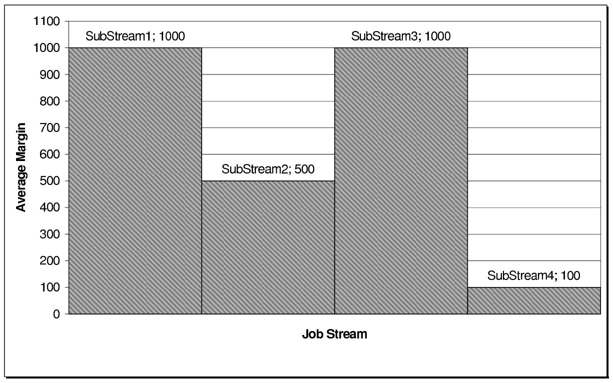

Figure 7: Margin value for each sub-stream

In the boundary regions between two consecutive job sub-streams, *DeadLine* produces a classification in which jobs priorities are amassed in extremely low (or high) values. Nevertheless, the *DeadLine* heuristics continues to perform properly if each job sub-stream length is sizable. This is because after a transient state the average margin used by heuristics (margin values of all the jobs analyzed before) changes approaching to the average margin of the current job sub-stream. It permits to the *DeadLine* heuristics to work properly i.e. emit job priorities with a distribution faithful to the exponential one.

To make the *DeadLine* heuristics autonomic and to enable it to compute a profitable priority distribution, even during the transition between one job sub-stream to the next one, we need to point out the causes of unprofitable distributions: the large differences in the average margin value between a job sub-stream and the consecutive one. This large difference causes an unprofitable priority values assignment that is the generation of too many low priority values or too many high priority values. To avoid this situation, we should be able to change the way in which we compute priorities, and to maintain a proper priority distribution when the average margin changes. It can be done *(i)* by changing the heuristics used to compute job priorities or *(ii)* by adapting the old one. We chose the second solution because, using a proper set of information, we can map this behavior on an autonomic heuristics as shown in Section 2.

Now, we provide a mapping between our autonomic heuristics for job classification and the concepts previously exposed. The functions are mapped in the following way:

$$\begin{aligned} f^o &: \; Job \times (Avg \times Curr.JD) \rightarrow QoS \\ f^\star &: \; Job \times QoS \times (Avg \times Curr.JD, IdealJD) \times base \\ &\quad \rightarrow (Avg \times Curr.JD', IdealJD') \times base'. \end{aligned}$$

The system state is composed by the following values: $(Avg \times Curr.JD)$ representing the average margin value of jobs analyzed until now and the fixed-size queue containing the priorities assigned to latest analyzed jobs (which number depends on a parameter we changed during the test session reported in Section 4). The polytope is represented by the $IdealJD$ value indicating the ideal job distribution among the priorities.

The heuristics state, represented by the $base$ value, is used by the *DeadLine* heuristics to compute the size of the QoS buckets (or priority intervals).

For each job submitted into the system, the *DeadLine* heuristics computes a priority value, updates the average job margin value (Avg) and updates the priority queue ($Curr.JD$) by adding the latest assigned priority and removing the oldest one.

If the distribution of the jobs within $Curr.JD$ is outside the polytope defined by the exponential job distribution used to configure the heuristics (Relation 1), the $base$ value is increased (or decreased) to $base'$ value, in order to adapt the heuristics behavior to the margin of new jobs.

With such a mapping the polytope can be represented as a point p representing the distribution used for configure the *DeadLine* heuristics, in this case each job distribution that differs from it will trigger a heuristic-state change. To avoid it we decided to expand the polytope admitting an allowance i.e. an acceptability area with a neighborhood instead of a point. A neighborhood centered in p with ray r. With such representation a job distribution lie inside the polytope if for each priority, the difference between the number of jobs within $Curr.JD$ having such priority and their expected values (exp) is less than $\frac{r}{exp}$.

With the mapping we implemented the polytope never change, in fact if current distribution does not lie inside the polytope the autonomic heuristics does not change it but modify its state, i.e. the $base$ value.

Nevertheless, another mapping, with a shifting polytope, can be provided. For instance considering a lower-bounded deadline-margin interval, i.e. an interval that does not start from zero but from the minimum value among the latest analyzed job. In this case when the current distribution lie outside the polytope, the reconfiguration process does not limit its intervention only to the heuristics state but the lower bound of job-margin interval is changed too.

4. EXPERIMENTS

To evaluate the goodness of the *Autonomic DeadLine Heuristics* (ADH) solution we conducted simulations applying the classification algorithm to a stream of jobs characterized by a high variability of the margin parameter.

To conduct evaluation we developed an ad-hoc event-driven simulator. For each simulation, we randomly generated a job-stream whose margin parameter was generated according to different distributions and described in each test. A simulation step includes: (1)selection and classification of new jobs, (2) update of the system and heuristics state, (3) check for correct behavior of the system, and eventually, perform the system adaptation. The time of job submission is driven by the *wall clock*. When the *wall clock* reaches the job submission time, the job enters in the simulation.

The aim of the experimentation phase was to carry out a priority distribution among jobs according to an administrative policy constituting the input of the proposed heuristics: System administrators can define a relation among the number and the kind of jobs in the system.

In our tests, the relation on the number of jobs for each priority is characterized as relation 1. We have five priority classes in our simulations, afterwards the possible priority values that a job can assume are in $[0, 4]$.

In our experiments we compare the ADH and the *DeadLine* heuristics. The performance metric used is the faithful of the classification given by these two heuristics, with the classification given by the exponential distribution described in 1, which we consider the optimal one in our case study.

Figure 8 shows the behavior of the *DeadLine* heuristics and of its autonomic version (ADH), compared to the *optimal* solution, when the margin of each job is uniformly generated. In this case ADH is never invoked because the *DeadLine* heuristics is good enough to model this situation and there is not need to operate to modify its behavior.

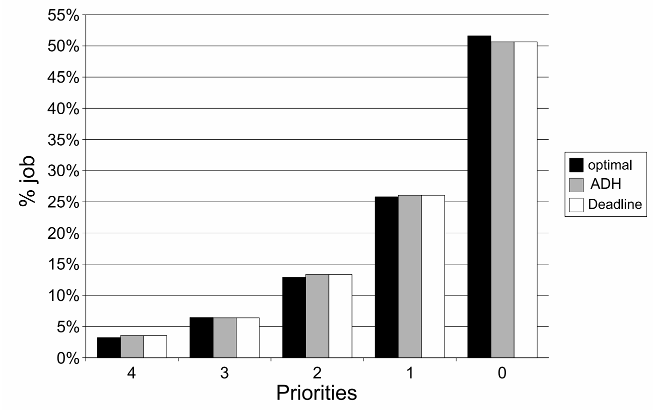

Figure 8: Deadline and ADH evaluation in the case of a uniform distribution of the margin parameter

Table 1 shows the range values used to generate the margin in the uniform case. The first column of the table shows sub-streams of jobs that have the same range of margin, the second column shows the range for the specific job sub-stream. Obviously, in the uniform case all the jobs belong to the same range.

#Jobs	Margin Range
2000	0-200

Table 1: Range values for the margin parameter in the case of a uniform distribution

Figure 9 shows the trend of the margin parameter in a non-uniformly distributed jobs generation, compared with the behavior of the *base* value (b), that is the base to compute the division of the range in QoS buckets. We can note that the *Average Margin* quickly changes and in a not predictable way. Moreover, *base* of the ADH frequently changes in order to control the generated jobs priority distribution. Table 2 shows the four job sub-streams in which we divided the job-stream, and the range of margin values used in these tests.

#Jobs	Margin Range
400	2000-4000
550	0-200
350	2000-4000
700	1000-200

Table 2: Range values for the margin parameter in the case of a not-uniform distribution.

Figure 10 depicts the priority assignments of ADH and the *DeadLine* heuristics compared with the optimal solution when the job deadline is generated according to the Table 2. The results point out that the autonomic heuristics lacks of accuracy for the low priority jobs, but it is close to the optimal solution for the jobs with high priority. Furthermore, ADH trend respects the optimal solution. The *DeadLine* heuristics, by itself, is not able to handle changes in the margin distribution: this because the margin parameter falls down too fast with respect to its average value.

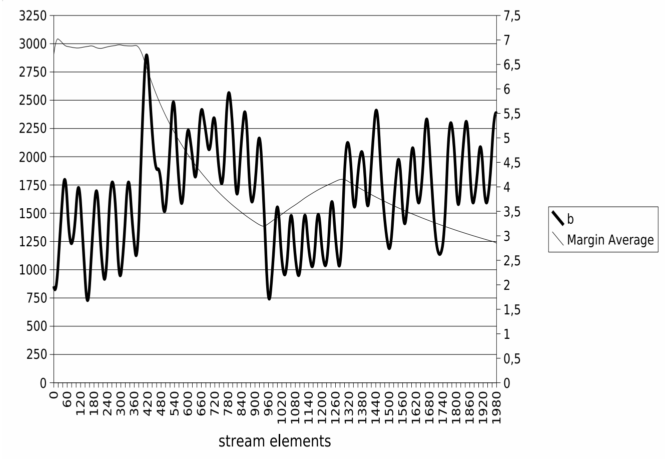

Figure 9: Deadline and ADH evaluation in the case of a not-uniform distribution of the margin parameter

The two last figures show the behavior of the system when the margin parameter is stable for some elements of the job-stream, namely the intervals [1000-1100] and [1300-1500]. Figure 11 shows that *base* does not change when the slope of the margin average curve is not steep. This implies that the *DeadLine* heuristics assignments are profitable. In the other cases, when the curve sheers or falls down, the ADH intervenes to adapt heuristics behavior.

Table 3 shows the job-stream subdivision and the range of margin values defined for each job sub-stream.

Finally, Figure 12 shows the behavior of the system according to a job deadline generation described in Table 3. The

#Jobs	Margin Range
500	3000-4000
500	500-200
500	500-700
500	500-200

Table 3: Range values for the margin parameter in the case of a non-uniform distribution.

DeadLine heuristics is not able to recognize hot spots in the QoS buckets. In fact, when a lot of jobs with the same margin are submitted to the system, they receive the same priority. Instead, ADH is able to change the *base* value to compute a better division of the interval and it is able to satisfy the administrator policy.

5. RELATED WORK

Adaptive performance tuning has only recently become conceivable, so only few papers address it directly.

Diao et al. [7] analyze how to choose certain parameters of the Apache web server in order to keep CPU and memory usage near a pre-set parameter. The authors make the assumption that there is an optimal setting for those parameters, and make no claim that the parameters impact the performance of the web server in a known way.

Raphael M. Bahati et al. define a policy as a notation to express required or desired behavior of systems and applications. In [3], they describe how policies are exploited and how they are realized as actions driving autonomic management in the context of managing the performance of an Apache web server. Their aim is to address the problem to express policies appropriate for an autonomic computing system and then map them to executable elements of the autonomic system.

In [16], Warren e al. describe mechanisms used to realize dynamic reconfiguration that must respect a number of fundamental issues when making run-time changes to a system. They suggest that such mechanisms have to behave according to: (1) the dynamic reconfiguration capability should not compromise applications *integrity/correctness*, (2) the runtime *overhead* introduced by a reconfiguration management facility should be acceptable, (3) the dynamic reconfiguration should be *transparent* to application developers. Their work are particularly concerned with preserving an applications integrity during periods of runtime change. They have extended OpenRec (a framework for managing reconfiguration of component-based applications [10]) with functionality which automatically verifies the structure of an application during periods of dynamic reconfiguration.

Other approaches, like ours, optimize performance maintaining a fixed level of service. For example Abdelzaher et al. [1] outline a system that maintains multiple complete content trees, each with a different quality setting. As workload increases, quality can be decreased in order to satisfy the maximum number of users. Additionally, Cohen et al. [6] use Tree-Augmented Naive Bayesian Networks to correlate

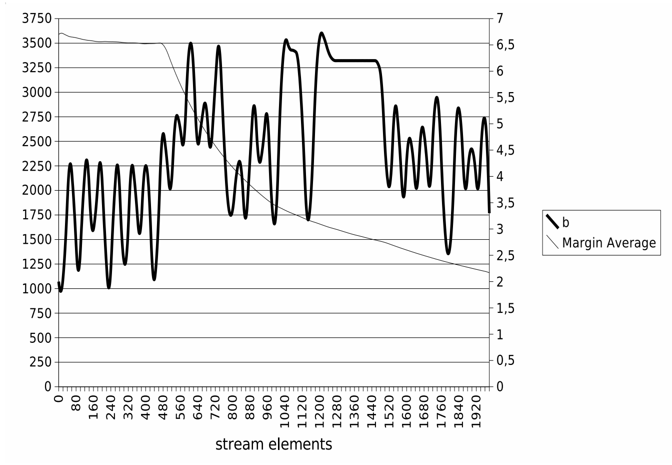

Figure 11: Margin average and *base* trend

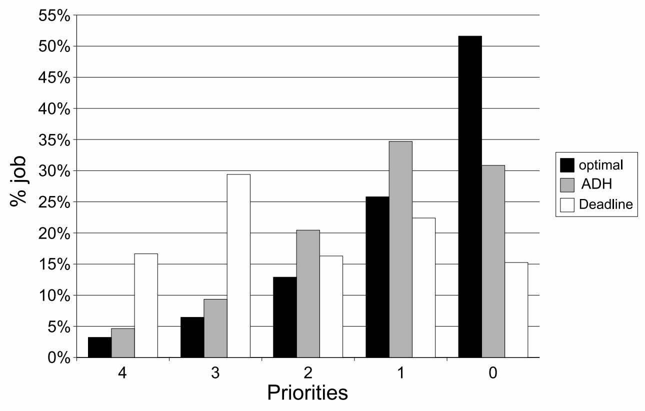

Figure 12: Deadline and ADH evaluation in the case of a not-uniform distribution of the margin parameter

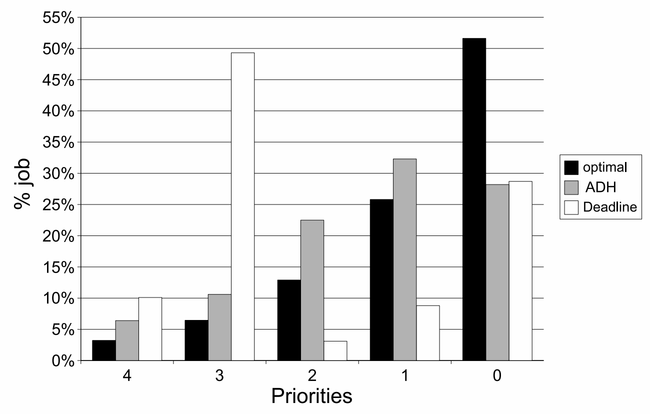

Figure 10: Deadline and ADH evaluation in the case of a not-uniform distribution of the margin parameter

system statistics to a high-level performance metric (compliance or non-compliance with required service levels). Unlike our work, this work relies on a specialized instrumentation layer.

Hellerstien et al. [8] analyzed the performance of a system over large spans of time with statistical models, which could then determine online when unexpected changes occurred. Our approach combines long-time observation with a short-time analysis used to adapt the system to changes.

Other work within the field of autonomic computing focuses on failure diagnosis [17, 5], file system organization [13], adaptive branch prediction [9], autonomous network creation [4], installation and configuration analysis [2] and utility function optimization [15].

6. CONCLUSIONS AND FUTURE WORK

Nowadays, The complex nature of current systems and applications implies the need to automate their management in order to meet operational or behavioral requirements. Our approach is not intended to be too abstract nor too formal. We describe with a high-level notation the elements that a heuristics designer needs to care about designing an autonomous heuristics.

The purpose of this paper is to present fundamental elements that are needed to make autonomous a heuristics-based system. We describe with a high level notation the elements that a heuristics system designer needs to care about in the design process of a new heuristics and to make it autonomous. We show an effective way for providing autonomic features to an heuristics-based system by the division of the heuristics state transition function from the system state transition function. This division enable the application designer to disjoin the system and the heuristics that drives the system itself. In this way the system can be *adapted* to new scenarios unplugging the current heuristics and plugging-in another one.

We present an interesting case study in which we show how our model can be applied to a real scenario. We point out results that exploit an autonomic behavior of the system, and an improvement of the respect of administrator policies.

We plan to apply our model to complex scheduling framework in which more than the deadline aspect is taking care. We would like to implement an autonomic scheduling heuristics with a shiftable polytope, more adherent to our formal model, like the one sketched in Section 3.2.

Finally, formalization could be more precise and could be extended to specify more in detail the system behavior.

7. ACKNOWLEDGEMENTS

We wish to thank Ariel David Techiouba which participated to the development and validation of autonomic deadline heuristics and Ranieri Baraglia for his precious advices for drafting this paper.

8. REFERENCES

[1] Tarek F. Abdelzaher, Kang G. Shin, and Nina Bhatti. Performance guarantees for Web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, 2002.

[2] Gagan Aggarwal. On identifying stable ways to configure systems. In *ICAC '04: Proceedings of the First International Conference on Autonomic Computing (ICAC'04)*, pages 148–153, Washington, DC, USA, 2004. IEEE Computer Society.

[3] Raphael M. Bahati, Michael A. Bauer, and Elvis M. Vieira. Mapping policies into autonomic management actions. *icas*, 0:38, 2006.

[4] Yu-Han Chang, Tracey Ho, and Leslie Pack Kaelbling. Mobilized ad-hoc networks: A reinforcement learning approach. *icac*, 00:240–247, 2004.

[5] Mike Y. Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, and Eric Brewer. Pinpoint: Problem determination in large, dynamic internet services. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 595–604, Washington, DC, USA, 2002. IEEE Computer Society.

[6] Ira Cohen, Jeffrey S. Chase, Moisés Goldszmidt, Terence Kelly, and Julie Symons. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *OSDI*, pages 231–244, 2004.

[7] Y. Diao, J. L. Hellerstein, S. Parekh, and J. P. Bigus. Managing web server performance with autotune agents. *IBM Syst. J.*, 42(1):136–149, 2003.

[8] Joseph Hellerstein Fan. Characterizing normal operation of a web server: Application to workload forecasting and problem detection.

[9] Alan Fern, Robert Givan, Babak Falsafi, and T. N. Vijaykumar. Dynamic feature selection for hardware prediction. *J. Syst. Archit.*, 52(4):213–234, 2006.

[10] J. Hillman and I. Warren. An open framework for dynamic reconfiguration. *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, page 594603, 2004.

[11] IBM. Autonomic Computing Initiative. www.ibm.com/autonomic.

[12] M. M. Lehman and L. A. Belady, editors. *Program evolution: processes of software change*. Academic Press Professional, Inc., San Diego, CA, USA, 1985.

[13] Michael Mesnier, Eno Thereska, Gregory R. Ganger, and Daniel Ellard. File classification in self-* storage systems. In *ICAC '04: Proceedings of the First International Conference on Autonomic Computing (ICAC'04)*, pages 44–51, Washington, DC, USA, 2004. IEEE Computer Society.

[14] J. Sgall. *Online Algorithms*, chapter On-line scheduling, pages 196–231. Book Series Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Monday, April 10, 2006.

[15] J. O. Kephart W. E. Walsh, G. Tesauro and R. Das. Utility functions in autonomic systems. In *In Proceedings of the 1st International Conference on Autonomic Computing*, May 2004.

[16] Ian Warren, Jing Sun, Sanjev Krishnamohan, and Thiranjith Weerasinghe. An automated formal approach to managing dynamic reconfiguration. In

ASE '06: Proceedings of the 21st IEEE International Conference on Automated Software Engineering (ASE'06), pages 37–46, Washington, DC, USA, 2006. IEEE Computer Society.

[17] Alice X. Zheng, Jim Lloyd, and Eric Brewer. Failure diagnosis using decision trees. In *ICAC '04: Proceedings of the First International Conference on Autonomic Computing (ICAC'04)*, pages 36–43, Washington, DC, USA, 2004. IEEE Computer Society.