A Control Theoretical Approach to Self-optimizing Block Transfer in Web Service Grids

ANASTASIOS GOUNARIS
School of Computer Science
University of Manchester, UK
and
CHRISTOS YFOULIS
Department of Automation
Alexander Technological Educational Institute of Thessaloniki, Greece
and
RIZOS SAKELLARIOU
School of Computer Science
University of Manchester, UK
and
MARIOS D. DIKAIAKOS
Department of Computer Science
University of Cyprus, Cyprus

Nowadays, Web Services (WSs) play an important role in the dissemination and distributed processing of large amounts of data that become available on the Web. In many cases, it is essential to retrieve and process such data in blocks, in order to benefit from pipelined parallelism and reduced communication costs. This paper deals with the problem of minimizing at runtime, in a self-managing way, the total response time of a call to a database exposed to a volatile environment, like the Grid, as a WS. Typically, in this scenario, response time exhibits a concave, non-linear behavior depending on the client-controlled size of the individual requests comprising a fixed size task; in addition, no accurate profiling or internal state information is available, and the optimum point is volatile. This situation is encountered in several systems, such as WS Management Systems (WSMSs) for DBMS-like data management over wide area service-based networks, and the widely spread OGSA-DAI WSs for accessing and integrating traditional DBMSs. The main challenges in this problem, apart from the unavailability of a model, include the presence of noise, which incurs local minima, the volatility of the environment, which results into a moving optimum operating point, and the requirements for fast convergence to the optimal size of the request from the side of the client rather than of the server, and for low overshooting. Two solutions are presented in this work, which fall into the broader areas of runtime optimization and *s*witching extremum control. They incorporate heuristics to avoid local optimal points, and address all the afore-mentioned challenges. The effectiveness of the solutions is verified via both empirical evaluation in real cases and simulations, which show that significant performance benefits can be provided rendering obsolete the need for detailed profiling of the WSs.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems; H.4.0 [**Information Systems Applications**]: General

Additional Key Words and Phrases: autonomic computing, data grids, extremum control, control theory, Web Services, OGSA-DAI

1. INTRODUCTION

The proliferation of Web Service-based Grids and the increasingly growing volume of data that is processed by and shared among such Grids gives rise to the need for the development of more robust techniques for the manipulation of large data volumes in autonomous, unpredictable environments that adopt service-oriented architectures. Thus far, the emphasis has been on architectures for the execution of SQL-like queries that span multiple Web Services (e.g., [Liu et al. 2003a; Alpdemir et al. 2003; Narayanan et al. 2003]), wide area query optimization (e.g., [Srivastava et al. 2006; Gounaris et al. 2005]) and the associated resource scheduling decisions (e.g., [Gounaris et al. 2006]). However, an important factor is the optimization of the cost of calls to WSs, which is largely dependent both on the size of the request (i.e., the size of the data block transferred between services) [Srivastava et al. 2006; Alpdemir et al. 2005], and on the network bandwidth. This is due to the fact that the response time of a remote server serving a series of calls with a fixed total size is characterized by a highly noisy, concave graph with regards to the size of the data chunks or blocks returned by the server. This graph has a different optimal point for different queries and/or different connections, or even different stages of the same query.

Consider for example the case in which a database is globally exposed as a WS through OGSA-DAI wrappers [Antonioletti et al. 2005]. Clients can retrieve data from this database by submitting requests containing SQL queries to the associated WS. When the result set is relatively large (larger than 1-3MB in the current release), then it must be returned in chunks to avoid out-of-memory errors and speed up transmission. Block-based data transmission is also the first choice when the data must be processed at the client side while it is being received in a pipelined fashion. As explained in Section 2, and reported in [Alpdemir et al. 2005], the response time first decreases when the block size is increased and after a point it starts increasing. Similar behavior for other data management WSs is reported also in [Srivastava et al. 2006]. Note that concave graphs can describe other aspects of the behavior of Web Services and Web Servers, in general. An example is the response time of an Apache Web Server with regards to the number of maximum clients allowed to be connected simultaneously to it [Liu et al. 2003b].

In this paper we examine the OGSA-DAI case. The objective is to minimize at runtime the total response time of a query by (continuously) tuning the size of the data blocks that are requested by the client from an OGSA-DAI WS in a self-managing way. We follow a control theoretical approach to this problem. In principle, autonomic computing can benefit a lot from control theory techniques, which are well-established in engineering fields and typically accompanied by theoretical investigations of properties such as stability, accuracy, and settling time [Diao et al. 2005]. Autonomic systems encapsulate components that monitor their environment to collect feedback, analyze the feedback collected, plan responses and enforce such responses by changing the configuration of the self-managing system. Control systems operate in a similar way, despite any differences in the terminology. In control systems, the main part is the controller, which receives system measurements as its input, and outputs a system configuration that impacts on the system performance, termed as control input.

A distinctive feature of our work is that the controller resides on the client rather than on the server. A main challenge in this case-study is that the entity to be configured is exposed as an unknown black box to the controller. A main consequence of this fact is that any solutions developed must operate well in the absence of a parameterized model that describes the behavior of the service. Another consequence is that any scope of using heuristics based on more detailed monitoring and internal state information of the server (e.g., as in [Liu et al. 2003b]) is eliminated. In other words, the only information about the controlled entity is restricted to the measured output, which is the response time of the request of the caller. However, the main benefit is that the measured output is the metric that mostly interests the user, since it includes the transmission cost over the network, and, as such, describes the performance from the user's point of view precisely. This comes at the expense of additional noise and jitter in the measured output, which are inherent in measurements of communication costs across unstable, volatile connections. The noise results in local peaks and non-monotonic behavior on both sides of the optimal point, rendering naive hill climbing techniques non-appealing. Finally, the convergence of any algorithm must be fast with low overshooting (i.e., not reaching extreme values in the transient phase while converging) and devoid of oscillations to the extent possible; otherwise serious performance degradation and out-of-memory errors are not avoided, respectively.

The main contribution of this work is three-fold.

—Firstly, to present fast and robust optimization algorithms that belong to the area of *runtime optimization* and *switching extremum control* [Draper and Y.Li 1954; Blackman 1962] and that are capable of converging to the optimal point quickly despite the presence of local optimum regions, noise, and bad choices for the starting point.

—Secondly, to apply the afore mentioned algorithms to the OGSA-DAI case, and conduct experiments to evaluate them. The evaluation results prove that the algorithms are robust, effective and are characterized by high convergence speed. More specifically, there are significant benefits in the response time in the generic case, where the optimal region of block size is not a priori known; moreover, the algorithms can yield improved performance even in the more limited scenario where this region can be approximated.

—Thirdly, to complement the empirical evaluation with thorough simulation experiments, which, in several cases, make different assumptions with respect to the environmental conditions. Simulations help us to study in depth the behavior of the algorithms presented, eliminating the interference from unidentified factors that are inherent in wide area systems. These simulations shed light on several strengths and weaknesses of the algorithms that empirical evaluation cannot reveal, and thus make it easier for developers to adjust the same solutions to other problems.

The results of this work render the process of calling services self-managing. As such, the need for detailed WS profiling and fine tuning becomes obsolete. The technique presented is applicable to any similar optimization problem with similar characteristics with those elaborated in Section 2; OGSA-DAI services are presented merely as a case study.

Note that complementary efforts to minimize the data transfer cost are described in [Seshasayee et al. 2004] and [Kosar and Livny 2004]. The former suggests improvements to the basic communication mechanism for WSs, whereas the latter investigates solutions based upon runtime selection of the transfer protocol. In addition, in order to avoid poor performance, it is possible to use WSs only for control while the actual data transfer is done via other file transfer mechanisms (e.g., based on SSH).

The remainder of the paper is structured as follows. Section 2 presents the OGSA-DAI approach in brief and measurements that motivated the research described hereby. The solution to the optimization problem is presented in Section 3. Section 4 deals with the evaluation. Related work is discussed in Section 5, and Section 6 concludes the paper[1].

2. THE OGSA-DAI APPROACH

OGSA-DAI services aim at exposing different data resources, such as relational and XML DBMSs, and raw files, in the form of WSs, which are called Data Services (DSs) [Antonioletti et al. 2005]. A single Data Service can provide access to multiple data resources, and this interaction is enabled through the so-called Data Service Resources (DSRs), which implement the core OGSA-DAI functionality. A client or another WS can direct "perform" documents at an OGSA-DAI DS. The protocol used is SOAP over HTTP and the perform document is in XML. Subsequently, a DSR accepts, parses and validates this document, executes the data-related activities specified within it, and constructs the response documents.

The activities described in the perform document define also the data delivery mechanism. Several modes are supported; here we investigate only the pull one. In this mode, the client sends requests to the service, which, as a response, returns all the results either in one big chunk, or in smaller blocks. In OGSA-DAI the block size is in tuples. The advantage of the former case is that the client sends just a single request, whereas, in the latter case, the client sends a series of requests until the complete result set is retrieved. Nevertheless, the advantage of the latter case is that firstly, it can handle large volumes of data that cannot fit entirely into main memory, and secondly, it allows for pipelined post-processing at the client's side. As such, retrieving the result set in a block-based pull mode is more widely applicable.

Suppose a query returning to a local client 100000 tuples of 100 bytes each, and that the client can configure the block size for the whole duration of the query results transmission. Figure 1 illustrates the response times for this query for different block sizes. The values shown are the averages over 5 runs on a machine with 512MB memory and 2.4GHz CPU speed. They are measured at the client side and they correspond to the cost of sending as many requests as required to retrieve the complete result set and getting back the response from the server. The WSRF[2] 2.2 flavor of OGSA-DAI is used; the simple WS flavor could have been used with no difference, since both employ the same asynchronous data transfer mechanism

[1]A short version of this work with more limited empirical evaluation and no simulations has appeared in [Gounaris et al. 2007].
[2]http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

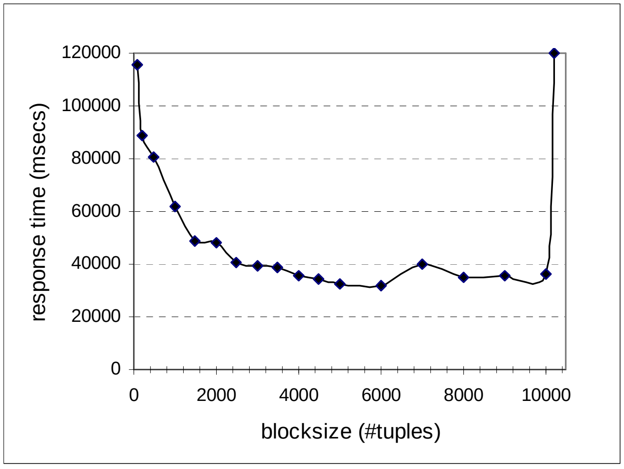

Fig. 1. Response times for a local query returning 100K tuples for different block sizes (tuple-length=100 bytes/tuple).

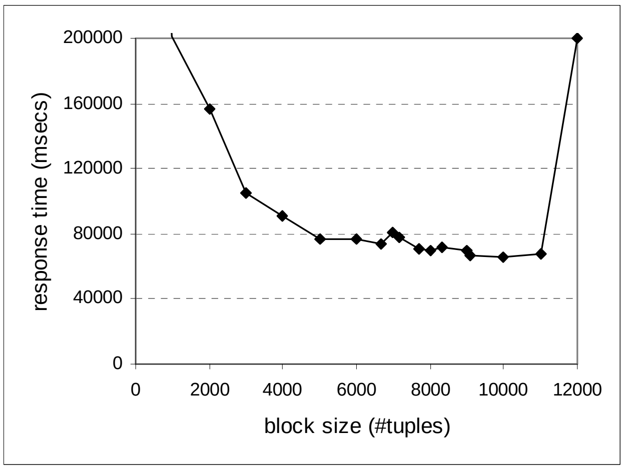

Fig. 2. Response times for a remote query returning 100K tuples for different block sizes (tuple-length=100 bytes/tuple).

using the SOAP over HTTP protocol. In this setting the optimum block size is around 6000 tuples. For this size, the response times are approximately 4 times lower than when the block sizes are a few hundred tuples. The sharp increase in response time with block sizes larger than 10K tuples is due to memory shortage.

Figure 2 shows the response times for the same query in a different setting. The server now is on a machine with 3.2GHz CPU speed and 1GB memory, and the client is remote (the server is in the UK, whereas the client is in Cyprus). No other application are running at the server side. We can observe that the optimum point has moved to around 10K tuples and the optimum size of the previous case now yields approximately 20% worse performance. In another setting, where the client

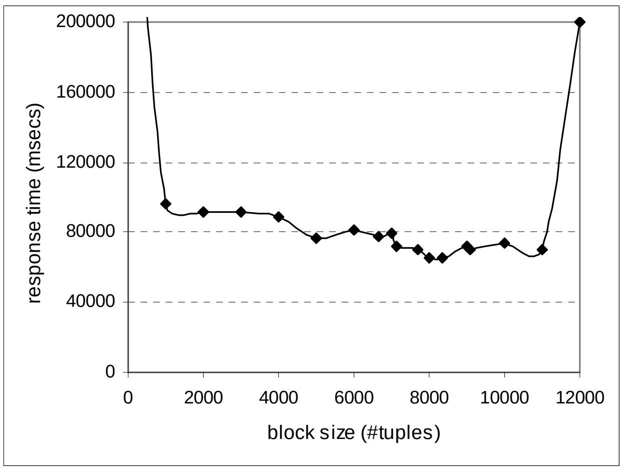

Fig. 3. Response times for a remote query returning 100K tuples for different block sizes (unstable connection, tuple-length=100 bytes/tuple).

block size	including 1st block			without 1st block		
	average	stdev	tuple cost	average	stdev	tuple cost
4001	1330	455	0.33	1242	102	0.31
4500	1491	428	0.33	1404	167	0.31
5001	1314	508	0.26	1203	153	0.24
6000	1560	638	0.26	1400	146	0.23
7000	2135	706	0.3	1944	146	0.28
8000	2426	706	0.3	2220	165	0.28

Table I. Summary of response times for a local query for different block sizes.

and the server are connected through an unstable wireless connection, the optimum point is modified to 8000 tuples approximately, as shown in Figure 3.

All these figures reveal a common pattern: the performance first improves (in a non-monotonic fashion) with increased block sizes, and after a point it starts degrading. It cannot be easily verified which exactly factors are responsible for these; in network applications of this kind the responsibility is diffused. However sending fewer blocks means that the total amount of requests transmitted is reduced and thus can improve performance. On the other hand, larger chunks of data require more resources, such as internal buffers, at the server side, which may start becoming stretched resulting into lower response times. This is why the concave effect exists also in local settings, as shown in Figure 1.

From the above figures, it has become obvious that in different settings in terms of different server-client pairs, the optimum data block size changes. In addition, the noise is high and as a result, on both sides of the optimal point there may exist local optimal points, which must be overcome by the self-optimizing mechanism. This is more evident in Figure 4, which shows 3 out of the 5 runs, the aggregate of which is in Figure 3. Profiling of each pair of nodes cannot be sufficient. This is because

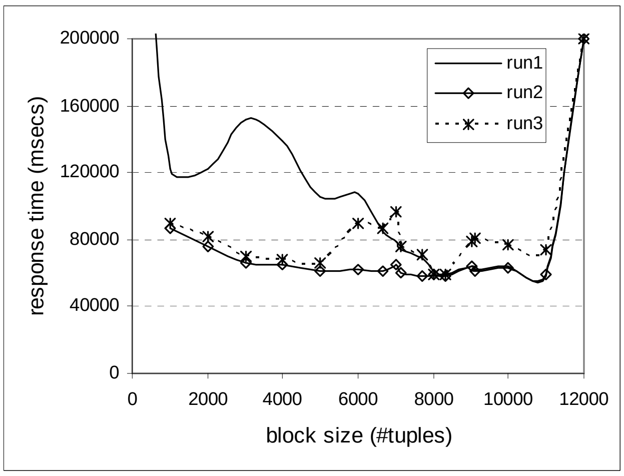

Fig. 4. Response times of individual runs of a remote query returning 100K tuples for different block sizes (unstable connection, tuple-length=100 bytes/tuple).

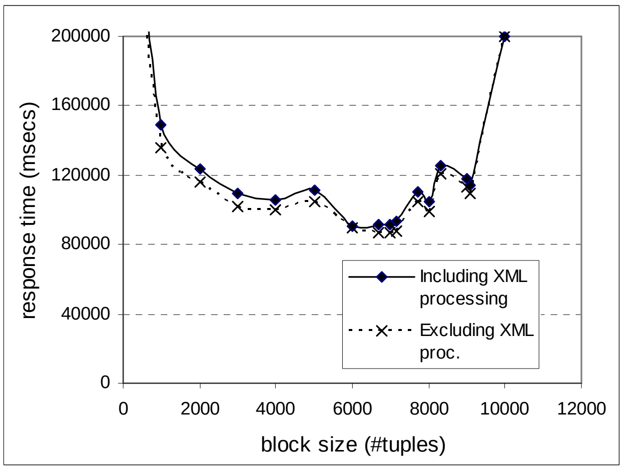

Fig. 5. Response times for a remote query returning 100K tuples for different block sizes (unstable connection, double tuple size: tuple-length=200 bytes/tuple).

of two reasons. Firstly, the resources are non-dedicated in general, which means that the service response time and network bandwidth are subject to frequent, unpredictable changes. Moreover, the optimal point depends also on the length of the tuples in the result set, which is query-dependent. For example, in Figure 5 the response times are presented for the same setting as in Figure 3 with the only difference that the tuple length is doubled. We can observe that the optimum block size has changed in this case as well. In Figure 5, two plots are depicted, one that takes into account the XML processing of the SOAP messages that are used to convey the results, and one that presents the aggregate response time, as in all

block size	including 1st block			without 1st block		
	average	stdev	tuple cost	average	stdev	tuple cost
4001	5198	1014	1.3	5146	996	1.29
4500	5386	856	1.2	5295	706	1.18
5001	6334	1750	1.27	6287	1782	1.26
6000	7337	1327	1.22	7260	1312	1.21
7000	8154	1724	1.16	8026	1696	1.15
8000	9636	1673	1.2	9459	1570	1.18

Table II. Summary of response times for a remote query for different block sizes.

figures thus far. It is shown that the shift of the optimum size, when compared against Figure 3, is not due to a change in the XML processing cost, i.e., if another, non XML-based protocol is employed instead of SOAP, the same phenomenon will appear.

The impact of the volatility of network connections and of noise are summarized is Tables I and II, which correspond to Figures 1 and 5, respectively. In the tables, it can be seen that the standard deviation in the measurements is high enough to mislead a simple optimizer, performing hill-climbing for instance, as to whether increasing the block size is profitable or not. Discarding the cost of the first block which includes the submission of the query on the service side, has little effect in remote cases. As such, applying simple hill-climbing or rule-based techniques (e.g., fuzzy control) is unsuitable in this case. Also, when 100K tuples are transferred and the optimal size is around 6-8K tuples, it means that the query will be finished in less than 20 cycles. As a result, a further requirement is for fast convergence. This leaves little scope for system identification and sampling that would allow for parameterization of an analytical model, based on which the optimum point can be estimated. Overshooting must also be low; otherwise either out-of-memory errors might occur, or the performance degradation due to a few cycles with a block size near the point where the system runs out of memory cannot be outweighed by future optimized decisions since the number of overall cycles is small.

3. ONLINE ADJUSTMENT

If y is the performance metric to be optimized, such as response time or equivalently, the per tuple cost in time units, and x the size of the data block, we assume that there exists a function f for which

$$y = f(x) + e$$

where e represents the noise. We further assume that e is responsible for the local peaks on both sides of the optimal block size.

To explain the concave shape, it can be also assumed that, at least in the neighborhood of the optimal point, there is a quadratic function

$$f(x) = a(x - x^*)^2 + b,$$

where a, b and x^* are unknown constants.

The optimal point is the value of x for which the derivative of $f(x)$ is zero, i.e., $\bigtriangledown f(x) = 0$. Obviously, this value is x^*.

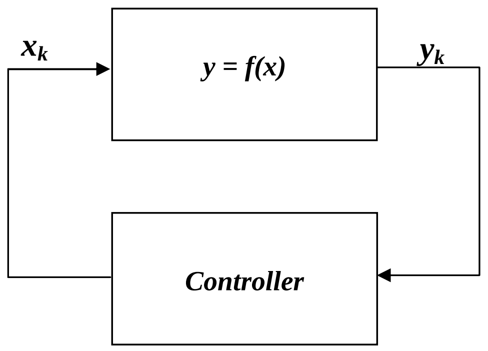

Fig. 6. A self-tuning system based on extremum control.

In this paper, two main approaches are investigated, none of which relies on the precise knowledge of a, b and x^*. The first is a typical numerical optimization method, while the second comes from the field of extremum control. In both approaches, in one adaptivity cycle or step, the time needed to transmit a data block (of a known size) is collected as feedback and a new value for the block size is calculated by the controller (see Figure 6).

3.1 Runtime optimization

The runtime optimization method is inspired by Newton's technique [Persinni 1988], which estimates, at each step, the next value of the block size, based on its previous values and the derivatives of x and y. More specifically, it defines that the value of x at the kth step, x_k is given by the following formula

$$x_k = x_{k-1} - \frac{\bigtriangledown f(x_{k-1})}{\bigtriangledown^2 f(x_{k-1})} \tag{1}$$

The second partial derivative ($\bigtriangledown^2$) in the denominator of the fraction allows for quick convergence. A nice characteristic of this method is that, if noise is eliminated and $y = f(x)$ indeed, then the algorithm converges in one step, since

$$\bigtriangledown f(x_{k-1}) = 2a(x_{k-1} - x_o)$$

and

$$\bigtriangledown^2 f(x_{k-1}) = 2a$$

However, the main drawback is that Newton's method is known to be very sensitive to noise, whereas, in the case examined in this report, the noise is not only existent but also non-negligible; moreover, the behavior of the system may have some quadratic characteristics, but this does not mean that a quadratic function, the parameters of which are unknown anyway, can describe it accurately. The unavailability of a model leads to an approximate estimate of the partial derivatives using backward difference operators $\Delta u = u_k - u_{k-1}$, i.e

$$\bigtriangledown f(k) \simeq \frac{\Delta y}{\Delta x} = \frac{y_k - y_{k-1}}{x_k - x_{k-1}}$$

It was mentioned previously that the main challenges in the problem investigated in this paper include, apart from the unavailability of a model, the presence of noise and the volatility of the optimum block size, even at an intra-run level. To mitigate the impact of the noise in the graphs, the measured output and the control input are firstly averaged over a sequence of n measurements. This may reduce the speed of response to changes. Hence, a proper choice of the averaging horizon must be made to trade-off speed of response with noise removal. This is further discussed in the following sections. To facilitate the controller to be capable of continuously probing the block size space, since the optimum point may move during query execution, a dither signal $d(k) = d_f \cdot w(k)$ is added, where d_f is a constant factor and w a pseudo-random variable that follows a Gaussian distribution with mean 0 and standard deviation 1. As such, instead of using Eq. (1) as it is, the value of the block size at each step is calculated by the controller as follows:

$$x_k = \overline{x}_{k-1} - \frac{\Delta \overline{y}_{k-1}}{\Delta^2 \overline{y}_{k-1}} + d(k) \tag{2}$$

where the average measurements $\bar{x}_k$ and $\bar{y}_k$ are given by

$$\bar{x}_k = \frac{1}{n} \sum_{i=k}^{k-n+1} x_i \text{ and } \bar{y}_k = \frac{1}{n} \sum_{i=k}^{k-n+1} y_i, \text{ respectively.}$$

3.2 Extremum control

The second approach investigated hereby is inspired by *extremum control*, which can yield results and track a varying optimum operating point even in the absence of a detailed analytic model. The role of an extremum controller is to manipulate the input x to the performance function $f(x)$, as a function of this output, as shown in Figure 6. Extremum control is based upon numerical optimization but goes beyond that since it can be blended with well known control approaches, including variable setpoint (optimum tracking) controllers, feedforward controllers, perturbation analysis, self tuning and adaptive techniques, so that noise, model uncertainties and time variations can be dealt with. Filtering and averaging are also typically included in the aforementioned techniques. There is a rich literature and many different methodologies and applications [Ariyur and Krstic 2003; Wellstead and M.B.Zarrop 1995; Larsson 2001].

In this paper, due to the difficulties mentioned in the previous sections, we decided to experiment initially with a simple and straightforward scheme, called *switching extremum control*.

Two flavors are examined; both can be described by

$$x_k = \overline{x}_{k-1} - g \cdot sign(\Delta \overline{y}_{k-1} \Delta \overline{x}_{k-1}) + d(k) \tag{3}$$

where g is the gain, and the function $sign(.)$ returns 1 for positive arguments and -1 for negative ones.

The formula above can detect the side of the optimum point where the current block size resides on. The rationale is that the next block size must be greater than the previous one, if, in the last step, an increase has led to performance improvement, or a decrease has led to performance degradation. Otherwise, the block size must become smaller. In the first flavor, $g = b_1$ is a constant (positive)

ID	#tuples retrieved	avg raw tuple size
Q1	150000	27 bytes
Q2	150000	65 bytes
Q3	200000	57 bytes
Q4	450000	4 bytes
Q5	1000000	2 bytes

Table III. The characteristics of the example queries.

tuning parameter. Without applying a dither signal, the step size is always the same, and since the absolute value of x, $\|\Delta x\|$, is equal to b_1, b_1 defines the rate at which x is modified.

In the second flavor

$$g = b_2 \| \frac{\Delta \overline{y}_{k-1}}{\overline{y}_{k-1}} \Delta \overline{x}_{k-1} \|, \quad b_2 > 0 \tag{4}$$

where b_2 is constant. In this case, the step size (gain) is adaptive and is proportional to the product of the performance change and the change in the block size. As in the first approach, it is important to limit the effect of noise around the optimum point, since very small changes in the block size in Equation (4) may induce a high noise to signal ratio. To this end, averaging is applied in this case as well. On the other hand, the convergence or the tracking ability of our iterative algorithm should not be harmed. Moreover, high overshooting and sustained oscillations are highly undesirable. These issues are investigated and discussed in the sequel. In both approaches, maximum and minimum limits can be imposed to avoid overshooting with detrimental effects.

4. EVALUATION

4.1 Prototype Implementation and Results

To test the actual performance of the techniques described, a thin client is built that can submit SQL queries to an OGSA-DAI DS. The client requests results to be delivered in blocks using SOAP/HTTP. The block size is determined by the client and can change during the delivery of the same result set. The data come from the TPC-H database[3] (scale 1) stored in a MySQL DBMS. Five queries are used throughout as shown in Table III. Note that the actual tuple length communicated across the network is significantly increased by the XML tags, as reported in [Dobrzelecki et al. 2006]. All queries are simple scan queries without joins, so that the computational load in the server is minimal and, as a result, the time to produce a block is negligible and does not affect the measurements.

The experimental setup is as follows, unless explicitly mentioned. The server is in Manchester, UK and the client is in Greece. The client is connected to the Internet through a wireless connection. The server's CPU speed is 3.2GHz and the memory 1GB. Each query configuration ran 10 times, and the different configurations were executed in a round robin fashion, i.e., there is no concurrency. The complete set of experiments presented here lasted for 10 days approximately around the

[3]http://www.tpc.org/tpch/

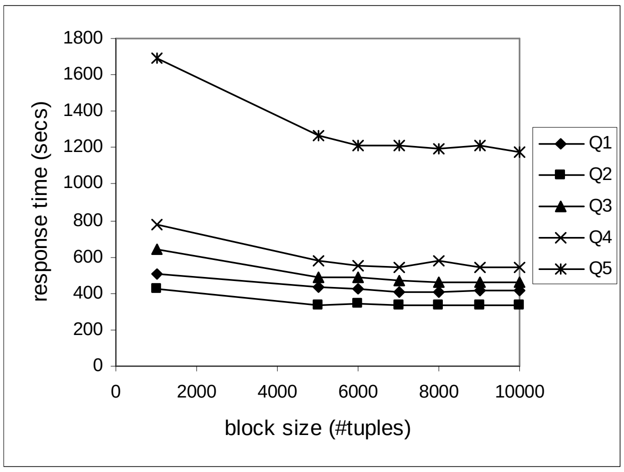

Fig. 7. Response times for queries Q1-Q5 for fixed block sizes.

clock, and as such, it reflects the condition of the network during significantly different workloads. Consequently, the measurements presented are characterized by a relatively high standard deviation due to the volatility of the environment. To smooth the standard deviation the lowest and the highest value of each set of the 10 runs is removed, and the average of the rest is presented.

The response times of the 5 example queries are presented in Figure 7. The fixed block sizes used to produce this profiling figure are 1K, 5K, 6K, 7K, 8K, 9K and 10K tuples. When the initial decision on the block size is clearly suboptimal, i.e., around 1000 tuples, an adaptive method can yield significant performance benefits; to the contrary, at first sight, it seems that there is little scope for optimization otherwise, since the near-optimum region is relatively wide. However, as will be discussed next, even in these cases where the initial decision is not clearly suboptimal, adaptive policies can yield more robust and consistent performance improvements that, in some cases, are around 10%.

On average, the optimum point for Q1 on the grounds of the profiling information is at 7000 tuples, with a 8000 tuples block size being very close to it, as shown in the figures. In reality, the actual optimum point is hard to be detected as it is moving between the 10 runs: the optimum is 5000 tuples once, 7000 four times, 8000 three times and 9000 twice. In the remainder of the text, the average optimum based on the profiling will be referred to as the optimum point. Note that Q1 seems to be costlier than Q2 although it transfers lower volumes of data across the network; this is explained by the different utilization of the server and the bandwidth of the connection during the runs of the two queries.

4.1.1 *Comparison of Adaptive Techniques.* Table IV presents the adaptive policies evaluated. In the Newton (NTN) and the switching extremum control (SEC) with adaptive gain, b_1 is used in the first runs when no adequate information has been gathered to estimate the derivative. The starting point in all configurations is 5000 tuples, and the averaging window n of both $\bar{x}$ and $\bar{y}$ is set to 3. Minimum and

name	policy	b_1	b_2	d_f
NTN-noD	NTN	400	-	0
NTN-D	NTN	400	-	100
SEC-const-D	SEC	400	-	100
SEC-5-noD	SEC	400	5	0
SEC-10-noD	SEC	400	10	0
SEC-15-noD	SEC	400	15	0
SEC-20-noD	SEC	400	20	0
SEC-25-noD	SEC	400	25	0
SEC-5-D	SEC	400	5	100
SEC-10-D	SEC	400	10	100
SEC-15-D	SEC	400	15	100
SEC-20-D	SEC	400	20	100
SEC-25-D	SEC	400	25	100

Table IV. The adaptive policies evaluated.

name	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**avg**
NTN-noD	**1**	1.1029	1.0269	1.0429	1.0912	**1.0528**
NTN-D	1.0047	1.0567	1.0053	1.0654	1.1127	**1.0489**
SEC-const-D	1.0215	1.1289	1.2465	1.0463	1.1785	**1.1243**
SEC-5-noD	1.0147	1.0613	1.036	1.0142	1.1549	**1.0562**
SEC-10-noD	1.03	1.0452	1.03	1.0322	1.1852	**1.0645**
SEC-15-noD	1.0366	**1**	1.0405	1.064	1.129	**1.05**
SEC-20-noD	1.0059	1.0724	1.0015	1.0182	1.0645	**1.0325**
SEC-25-noD	1.012	1.0486	**1**	1.0395	**1**	**1.02**
SEC-5-D	1.0249	1.1047	1.1277	1.0766	1.2189	**1.1106**
SEC-10-D	1.0214	1.0147	1.0893	1.0674	1.1614	**1.0708**
SEC-15-D	1.0102	1.0239	1.0987	1.0405	1.1528	**1.0652**
SEC-20-D	1.0103	1.0616	1.1413	1.1018	1.1288	**1.0888**
SEC-25-D	1.0042	1.0659	1.1748	**1**	1.1177	**1.0725**

Table V. Comparison of adaptive policies.

maximum value constraints on the block size are imposed, set to 1000 and 10000, respectively.

The comparison of the techniques is shown in Table V, which presents the normalized response times for each of the adaptivity configurations of Table IV. The lowest response time for each query is given the value 1 in the table. As such, the cell values correspond to the performance degradation when compared against the most effective of the policies investigated. Several useful observations can be drawn from this table.

—Firstly, taking into account the small differences between the response times for different block sizes shown in Figure 7, the differences between the performance of the different adaptivity policies are not negligible.

—Secondly, there is no policy that outperforms the others consistently.

—Thirdly, as expected, the Newton-based techniques cannot perform as well as the best switching extremum control policies; the former are known to be more sensitive to noise.

block size	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**avg**
1000	1.25	1.281	1.4	1.45	1.43	**1.361**
5000	1.0718	1.0052	1.0752	1.0729	1.075	**1.06**
6000	1.0422	1.0286	1.0626	1.0198	1.024	**1.0354**
7000	**1**	1.018	1.0205	1.0104	1.0235	**1.0145**
8000	1.0013	**1**	1.0077	1.0698	1.0143	**1.0186**
9000	1.0134	1.005	**1**	**1**	1.0302	**1.0097**
10000	1.0241	1.0151	1.0084	1.0084	**1**	**1.0112**
dynamic	**0.989**	**0.9945**	**0.9436**	**0.9764**	**0.8922**	**0.9591**

Table VI. Comparison of dynamic adjustment of block size against fixed size policies.

—An additional observation is that, somewhat counter-intuitively, the effects of the dithering signal and adaptive gain based on the performance change seem to annul each other, and consequently, the best approaches to SEC with adaptive gain seem to be those that have zero dithering factor. On average, SEC-25-noD yields 2% worse response times than the best policy (which is not known a priori), whereas the best SEC policy with both adaptive gain and dither signal yields 6.52% worse performance. An explanation could be that the moving optimum point and the volatility of the environment are adequate for continuously searching the block size space and thus to overcome local optimum points, whereas dithering results in increased instability. Also, in most cases, the dither signal does not change the mean value of block size but causes a fluctuation on both sides of it with an amplitude which depends on d_f. When the dynamic adjustment operates near the starting point area, negative dither signals cause more significant performance degradation than the performance improvement in the case of positive signals because of the shape of the response time graphs. As such, applying a dither signal seems more appropriate for cases in which the slope on both sides of the global and local optimum points is steeper.

—Finally, the first flavor of the SEC policy with constant step, SEC-constant-D, is not efficient. Perhaps, the performance would improve with different values for b_1 but this would shift the problem from fine-tuning the block size to fine-tuning the adaptivity parameters.

4.1.2 *Performance Improvements.* Thus far we have discussed how the adaptivity techniques compare to each other. In the following paragraphs the comparison with the fixed block size cases will be discussed, with a summary provided by Table VI. In this table, the values are normalized with the optimum point of Figure 7 set to 1. The last row depicts the relative performance of the most effective policy for each query, as shown in Table V. We can observe that:

—For this experiment set, the improvement may exceed 40% (if the fixed sized blocks were 1000 tuples). Similar or much larger improvements may be noticed in other settings (e.g., Section 4.1.4), in the generic case where the near-optimum area of block sizes is unknown from before. In the following, the more limited case where this area can be approximated is discussed.

—Dynamically adjusting the block size outperforms fixed size configurations by more than 4% on average, even if these are known, e.g., through profiling, and set to their optimum before execution. Also dynamic techniques can track the optimal point; in fixed configurations, the optimum from a finite set that has been profiled is chosen; however it might be the case that the global optimal is not in this set.

—On average, the best size for fixed size configuration in our experiments is 9000 tuples. This yields more than 5% performance degradation when compared to dynamic adjustment, which may be translated into several minutes in real time units (given that all queries and especially Q5 are rather expensive and long running as shown in Figure 7).

—The starting block size of the adaptivity policies is 5000 tuples. If this size was used for fixed size configurations instead of 9000, the performance improvement would be more than 10%.

—In the table, the performance of the best adaptivity policy is taken into account, for each query. If instead, SEC-25-noD is used for all queries, the average performance improvement is around 3%, 8% and 38% compared to fixed blocks of 9000, 5000 and 1000 tuples, respectively, which is still significant. NTN-D yields 0.5%, 5.5% and 36% lower response times, respectively. NTN-noD, which requires not a single configuration parameter, behaves the same as a 9000 tuples fixed block size.

The performance benefits may be further increased by modifying the averaging window. We reran Q1 for two adaptivity policies, namely NTN-noD and SEC-25-D, and changed the averaging window from 3 to 5. In both cases, a further decrease of response time by 2.3% was observed. Also, as discussed earlier, SEC policies with adaptive step and dither signal do not perform as efficiently as SEC policies with adaptive step but without dither signal. When rerunning Q1 for SEC-25-D and dithering factor 200 and 400 instead of 100, we noticed negligible performance improvement in the first case, and performance degradation in the latter. Consequently, it can be inferred with high confidence that the value of d_f does not play a big role in approaches with dithering signal.

4.1.3 *Speed of Convergence and Stability.* The convergence speed of the second flavor of the switching extremum control (SEC) techniques is fast, and this is depicted in Figure 8. On average, the adjustment converges to its final region at 5 adaptivity cycles, i.e., five block transmissions. When there is no dither signal, the block size remains stable thereafter. The drawback is that if either Δy_{k-1} or Δx_{k-1} remains unchanged for two consecutive averaging windows, then a chain effect takes place where all future block sizes cannot be modified. This is not desirable when the optimum point changes significantly during query runtime. It is avoided with dithering, where there is a continuous search of the space, which sometimes has negative effects as discussed earlier, but in some cases enables higher accuracy as in Q4 (see Figure 8). The fast convergence property does not hold for the first flavor of SEC, as shown in Figure 9. SEC-const-D seems to require more cycles to converge than the complete length of the query execution.

The fluctuation effect of dither signal is more evident in approaches based on Newton's technique and it may lead to instability. Figures 10 and 11 refer to the average decisions of NTN techniques during query execution for Q1 and Q5.

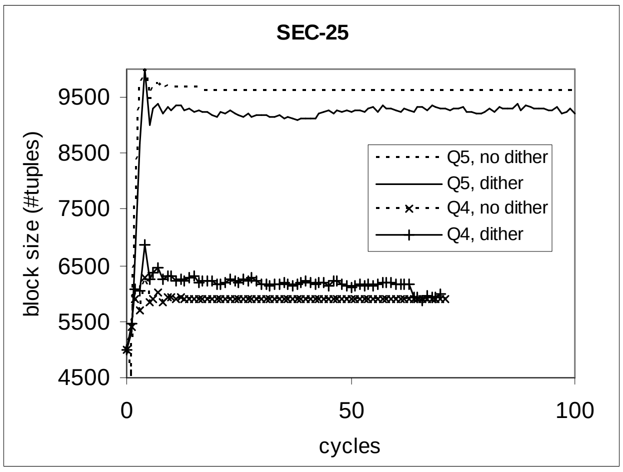

Fig. 8. The block sizes at different adjustment cycles for Q4, Q5 when SEC is employed with $b_2 = 25$.

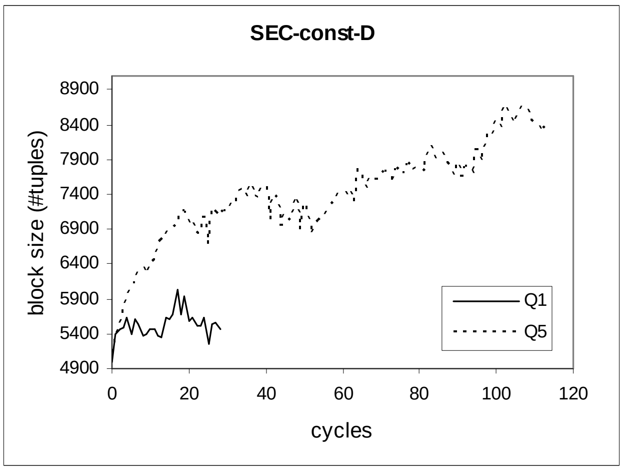

Fig. 9. The block sizes at different adjustment cycles for Q1, Q5 when SEC-const-D is employed.

Without dithering, such approaches may not be as accurate as SEC ones, but they are characterized by the same convergence speed. However, their estimates may fluctuate, as in the last cycles of Figure 11. The amplitude of the fluctuations, which are due to both the presence of a dither signal and the sudden changes in NTN, can be mitigated by increasing the averaging window.

4.1.4 *Dynamic adjustment with clearly suboptimal starting point.* In the experiments presented above, the initial starting point for the adaptive techniques has been relatively close to the optimum. To further prove the robustness and efficiency of the adaptivity approaches, the five queries are executed again (in a LAN setting this time) and the initial starting point is set to 1000 tuples. In this setting, the per-

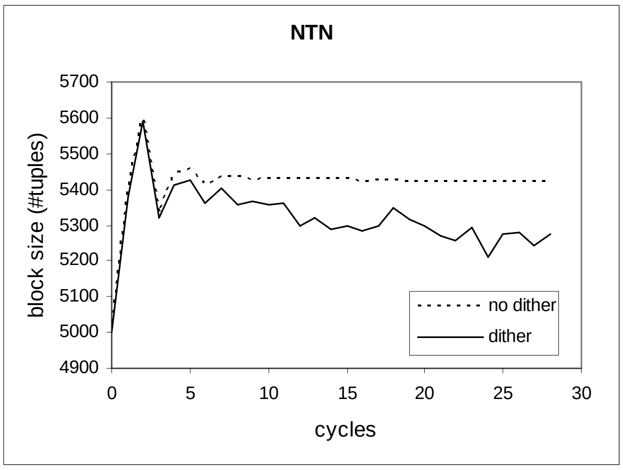

Fig. 10. The block sizes at different adjustment cycles for Q1 when NTN is employed.

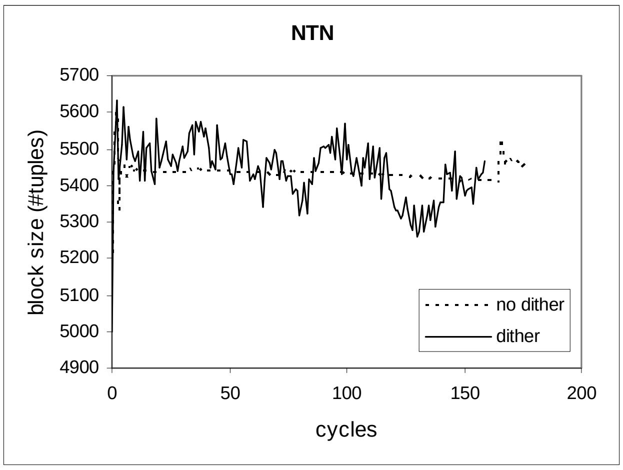

Fig. 11. The block sizes at different adjustment cycles for Q5 when NTN is employed.

query	dynamic		fixed at 1000 tuples	
	avg	stdev	avg	stdev
Q1	1.152	3.77%	2.581	0.42%
Q2	1.191	3.58%	2.557	0.18%
Q3	1.192	3.19%	2.404	1.08%
Q4	1.154	6.52%	2.178	0.41%
Q5	1.02	5.15%	2.06	0.77%

Table VII. Comparison of the performance of dynamic adjustment of block size when the initial block size is clearly suboptimal

formance degradation of such a suboptimal decision is more severe than in a WAN

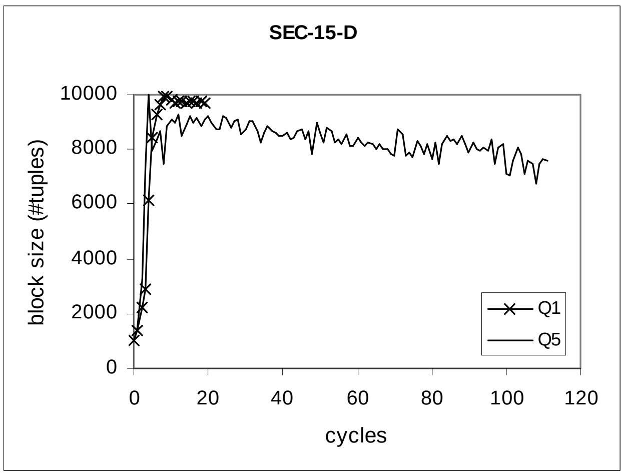

Fig. 12. The block sizes at different adjustment cycles when SEC-15-D is employed and the starting point is 1000 tuples.

environment (more than 100%). A single configuration of the SEC techniques is picked, namely SEC-15-D, which is outperformed by SEC approaches with adaptive gain and no dithering but, according to the previous experiments, is the most effective from the SEC approaches that apply dithering. Table VII summarizes the results in normalized time units. The value of 1 corresponds to the lower response time after performing profiling with several fixed sized configurations. When there is no adjustment, the response times are more than 200% the optimum, whereas SEC-15-D yields on average only 14% worse response times. The intra-query behavior for Q1 and Q5 is shown in Figure 12; the rest of the queries exhibit similar behavior. An observation is that the adaptivity techniques very quickly move out of the clearly suboptimal region of 5000 tuples or less. Overshooting is avoided due to the maximum hard constraint imposed.

4.2 Simulation Results

The simulation setup is intended to provide a complementary study and experimentation with the algorithms proposed so that a more representative picture of the results can be obtained and more reliable conclusions can be drawn. Empirical evaluation in real cases cannot always reveal some aspects of the algorithms's characteristics, since extensive experimentation with many different parameter settings and algorithms requires a significant amount of time. Using simulation further remarks can be made and other issues can be more clearly investigated.

On the basis of the profiles obtained by real evaluation experiments, as presented in Figure 7 and Table VI, we developed a simulation engine based on $MATLAB^{TM}$. We implemented all adaptive policies proposed in Table IV and also experimented with many different parameter settings. Our experimentation revealed a number of additional issues, which allow a more detailed and fairer comparison of the algorithms, and further reveal their features, advantages and disadvantages.

An important aspect of a faithful simulation in our case study is the ability to emulate the unique characteristics present. Although the response times exhibit an approximately concave shape on average, we have seen that the impact of a number of unknown and unpredictable factors – variable network conditions, server utilization level, transients after block size changes – induce jitter and hence noisy measurements and also frequent movements of the optimal point. These factors give rise to local peaks and non-monotonic behavior. These issues have to be somehow emulated.

To this end, we incorporated into the simulation engine extra features for modifying the profiles. Jitter, transients and movements of the optimum point can be injected into the initial profiles in order to test how they affect the performance and whether they can be dealt with the proposed policies. Jitter and transients may be emulated in the form of additional random noise uniformly distributed around the static profile averaged values over all runs. Small movements of the optimal point can easily occur due to the random noise added in the profiles, which is assumed to be more or less representative of small deviations in the measured values attributed to the volatility of the environment, and are expected even under normal conditions.

We also wanted to cover the case of sudden changes of a more critical nature that can be attributed to sudden non-trivial congestion occurrences, server failure or significant server performance degradation. Such critical events can result in significant changes in the profile shape and level of values, e.g., an increase of the measured values for all block sizes, or a movement of the optimal value to significantly lower or higher block size values. The simulation engine developed is in the position to emulate these events so that the performance of the adaptive policies and their robustness can be studied. Deviations corresponding to 10-15%, on average, of the static profile values for Q1 and Q3 are injected, and 3-5% for Q2,Q4,Q5, which are proportional to their standard deviations (not shown in Figure 7).

Nevertheless, even when the aforementioned issues are taken into account, it should be noted that a simulation environment is certainly very different from its real world counterpart, and it is natural to expect deviations in some aspects of the results. It is impossible to simulate precisely the exact nature and dynamics of the real systems. Moreover, in the simulation we are able to replicate runs so that some characteristics are kept the same and fair comparison of different policies can be made. This is not true in general in the real world, where different runs may possess different (and perhaps not observable) characteristics. Nonetheless, further useful observations can be made and important conclusions can be drawn.

4.2.1 *Simulation 1 - Tuning and comparison results.*

Prior to the evaluation of the adaptive policies using simulation, the influence of the parameters involved must be studied and an appropriate tuning of their values must be conducted. This is typical in the implementation of any algorithm, and involves a gain calibration procedure, i.e., selection of a “good” or “optimal” value so that the performance objectives are met. This procedure presupposes good knowledge of the algorithm’s environment. In our case we assume that there is profiling information available, on the basis of real data acquired by prior experimentation. However, the volatility of the environment requires the usage of algorithms which are robust in that sense, i.e.,

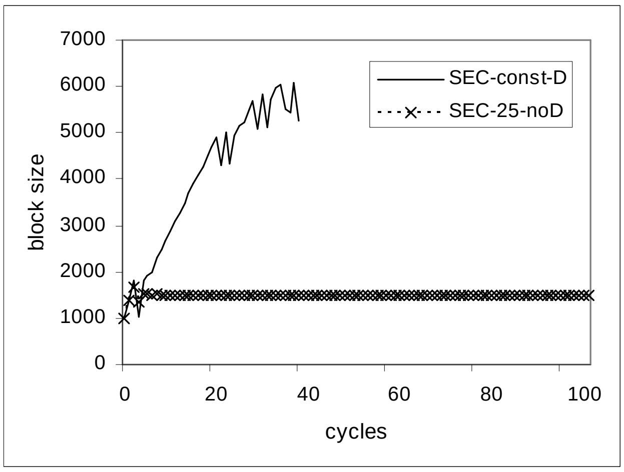

Fig. 13. Comparison of the performance with the initial parameter values and a starting point of 1000 tuples.

they are not based on proper or continuous tuning of their parameters to perform well.

While the *Newnton-based* algorithms do not require any tuning, the *constant gain* and *adaptive gain* policies are based on choosing the b_1, b_2 parameters. Moreover, if a dithering signal is employed, a value for d_f must be chosen as well.

Our initial runs with the values $b_1 = 400$, $b_2 = 5, 10, 15, 20, 25$, $d_f = 100$, as they appear in Table IV, revealed some similar results to the ones described previously in Section 4.1, as well as some different observations. The performance of most policies is heavily influenced by the choice of the starting block size. When this is selected as $B_S = 5000$ tuples, in a position lying in the wide near-optimal region of our profiles, similar results to Section 4.1 are obtained. On the contrary, when a clearly non-optimal initial value $B_S = 1000$ tuples is used, significantly different results are obtained. More specifically, the Newton-based and the adaptive gain policies show the same transient behavior but a poor steady-state performance, and they are stuck far from the near-optimal region. On the contrary, the constant gain policies are robust in reaching the optimum, but with a retarded and more oscillatory transient (and steady-state) response. These aspects are depicted in Figure 13.

We continued with further experiments for appropriate tuning of the parameters involved. The tuning procedure and the results found are described below. In the following runs, the Query 1 profiled data are selected.

—The performance of the *constant gain* policies when tuning the gain b_1 to take the values $400, 800, 1200$ is depicted in Figure 14. The robustness to jitter and local peaks is good; the performance and the speed of convergence improve as the gain is increased, at the expense of larger amplitude of oscillations at steady state. The gain value can be selected so that a tradeoff between speed of convergence and overshooting is achieved. However, this technique is robust enough to yield good performance even for low and not properly tuned constant gain values.

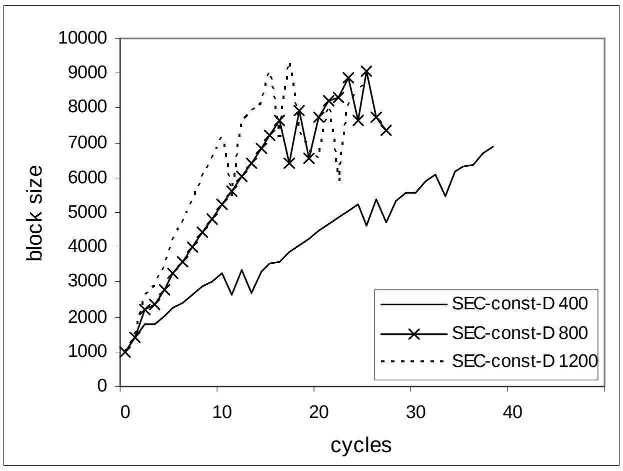

Fig. 14. The performance of constant gain policies with $b_1 = 400, 800, 1200$ for a starting point of 1000 tuples.

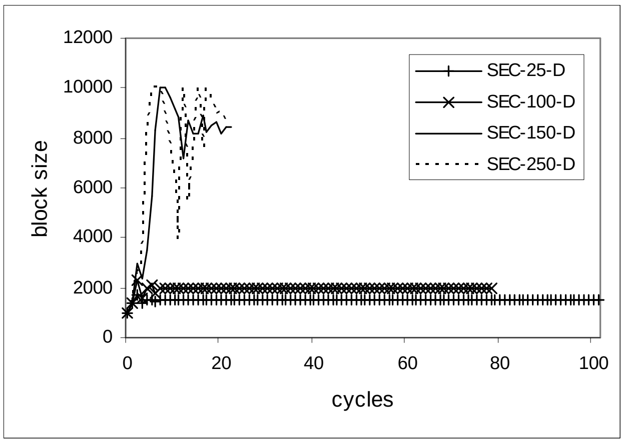

Fig. 15. The performance of adaptive gain policies with $b_2 = 25, 100, 150, 250$ for a starting point of 1000 tuples.

—The influence of the gain b_2 to the performance of the *adaptive gain* policies is shown in Figure 15. We observe significant changes in its response when the gain is increased from 25 to 250. However, the results are not consistent and robust, they are sensitive to jitter and local peaks (see Figure 16 where the results of two different runs are shown), and when convergence close to the real optimal is achieved, the high gain may induce high overshooting. We experimented with a simple tuning procedure. First, the gain b_2 is increased up to a value in which tracking of the optimum in steady-state is possible in most cases. Second, the oscillations and hard-limit impacts are suppressed by imposing a rate limiter, i.e., an upper bound value $\overline{B}$ s.t. $\Delta y = y_k - y_{k-1} \leq \overline{B}$. With $b_2 = 250$ and $\overline{B} = 5000, 2000, 1000$ we obtain the improved responses shown in Figure 17,

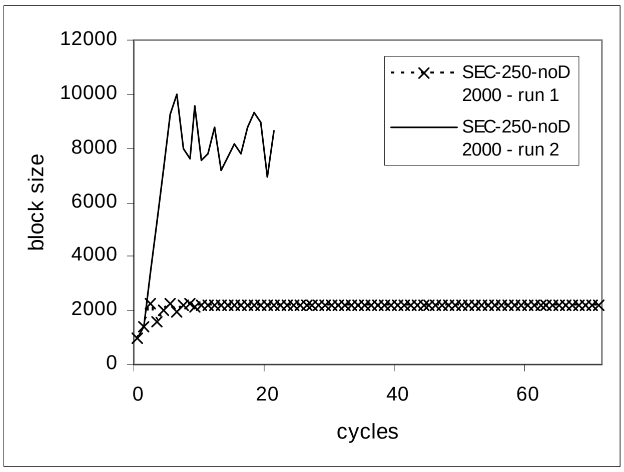

Fig. 16. The performance of the SEC-250-noD policy with $\overline{B} = 2000$ and a starting point of 1000 tuples - two different runs are shown.

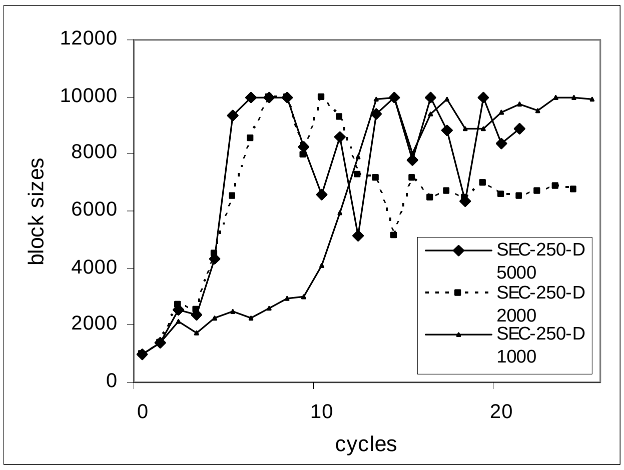

Fig. 17. The performance of the SEC-250-D policy with $\overline{B} = 5000, 2000, 1000$, a starting point of 1000 tuples and dither $d_f = 10$.

where the influence of the rate limiter bound may also be seen.

—The *Newton-based* policies cannot be influenced by the choice of a gain and further experimentation suggested that they are inappropriate in dealing with our profiles and their discontinuous and noisy characteristics. Hence, they are not studied in the sequel.

—The role of *dither* and its magnitude has also been studied. For the constant gain policies the effect of dither and its magnitude is not significant. Those policies do not require the use of dither to remain excited (in order to continuously search the block size space thus avoiding stagnancy). However, the dither $d_f = 100$ used so far usually helps. For the adaptive gain policies it has been observed that in the absence of dither the combined effect of averaging and convergence leads to stagnancy. A large amount of dither is misleading and induces oscillations, while a small amount of dither can be useful. We decided to use a much smaller amount of dither compared to the initial experiments to ensure excitation and tracking ability. In fact, the results in Figures 14,15,17 are obtained with $d_f = 10$.

—Changing the *averaging horizon* from $n = 3$ to $n = 5$ seems to result in a small improvement for the constant gain policies. We observed oscillation suppression without robustness reduction. Unfortunately, larger averaging horizons result in performance degradation for adaptive gain policies, mainly because of their fragile robustness to the volatility of the environment.

—Constant and adaptive gain policies exhibit similar performance for different values of the tuning parameters. Nevertheless, adaptive gain policies are worse than constant gain ones in terms of consistency and robustness, which can be quantified as 4-5 times larger standard deviations. As shown in Figure 16, the large deviations of the adaptive gain policies are translated in the frequent occurrence of runs that fail to overcome the profile obstacles and remain far from the near-optimal region, thus yielding fairly degraded performance.

4.2.2 *Simulation 2 - Tracking ability.* To test the tracking ability of our adaptive policies to changes in the optimal point we decided to inject synthetic changes into the simulation. We use Query 1 with an increased number of tuples (from 150000 to 450000). The changes introduced are as follows :

—At step (cycle) k=100 the near-optimal region of the profile (excluding the non-optimal initial region of 1000-5000 tuples which remains constant) for Query 1 is shifted to the right by 1000 tuples, i.e., the optimal point moves from 7000-8000 tuples to 8000-9000 tuples.

—At step k=150 the profile is scaled by 1.25, i.e., the optimal point remains at 8000-9000 tuples, and all response times are multiplied by 1.25.

—At step k=200 the profile is shifted to the right by 2000 tuples, i.e., the optimal point moves to 10000 tuples.

—At step k=250 the profile is scaled by 0.75, i.e., all response times are multiplied by 0.75.

—At step k=300 the profile is shifted (cyclically) to the right by 2000 tuples, i.e., the optimal point moves to 6000-7000 tuples.

—Finally, at step k=400 the profile is shifted to the right by 2000 tuples, i.e., the optimal point moves to 8000-9000 tuples.

Figure 18 reveals the good and bad characteristics of the adaptive gain schemes. Low overshooting and fast response are on the positive side. On the negative side, some runs may be characterized by poor and slow tracking and poor robustness properties. As before, they cannot track changes consistently and they often fail to detect changes made. Dither can only partially help, but it does not resolve the problem.

Figure 19 shows that the constant gain schemes are capable of tracking well the changes made. Larger values for the gain b_1 allow quicker response but also oscillations of larger amplitude.

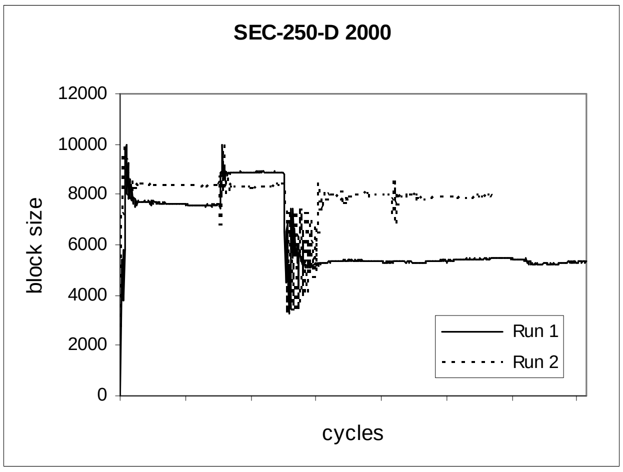

Fig. 18. The performance of the SEC-250-D policy with $\overline{B} = 2000, d_f = 10$ - tracking ability with two runs.

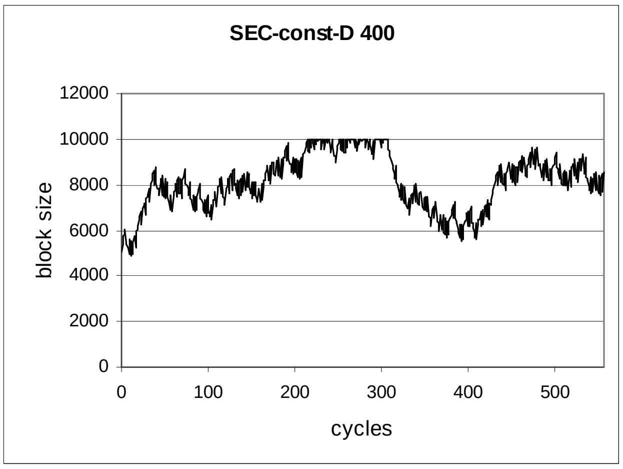

Fig. 19. The performance of the SEC-const-D policy with $b_1 = 400, d_f = 10$ - tracking ability.

4.2.3 *Simulation 3 - Modified profiles.* To get a better understanding of the influence of several factors to the performance of our algorithms we continued our experimentation with modified profiles. In the previous simulations we used static profiles based upon the average values obtained over all runs for each query, corrupted by additional noise. In this simulation we employ the following modifications:

(1) First, instead of using average value profiles, we experiment with profiles corresponding to individual runs, with or without additional noise.

(2) Second, smooth versions of the initial non-smooth profiles are produced, by fitting a quadratic model to the data.

It is noticeable that we did not observe any difference by replacing the average values profiles with the individual runs profiles, with or without additional noise. As a matter of fact, it is not difficult to see that, besides their different numerical values and optimal point location, they share the same characteristics, i.e., discontinuities, local peaks (non-concavities), volatility, etc. All observations regarding the advantages and disadvantages of each of the two sides (constant and adaptive gain policies) made previously continue to hold and this provides further justification for the analysis and the remarks made in the previous sections.

On the contrary, when experimenting with smooth versions of any of the discontinuous profiles (either corresponding to individual runs or average values) we observed different characteristics. In this case, the adaptive gain policies outperform the constant gain policies, since the oscillations in steady state quickly die out and the transient performance is faster. Repeated simulations have also assured us that the adaptive gain policies have good robustness properties. This is of course attributed to the absence of discontinuities in the profiles.

4.2.4 *Discussion.* Parameter tuning in an attempt to yield improved results for specific applications is common in algorithmic development. When the environment is predictable, time invariant and can be modelled precisely it is a fair thing to do. For our case study and the aforementioned challenges, we would rather prefer a general purpose adaptive technique characterized mainly by robustness and general applicability, while being able to yield acceptable results without requiring fine and cumbersome tuning.

The simulation results presented before suggest that constant gain policies possess to a high degree these desirable characteristics regardless of the initial block size; they can perform well even without proper tuning (when required, tuning can be done with a single parameter in a straightforward manner), although their transient behavior and steady state stability deviate from the optimal. Their robustness to non-smooth profiles is good and they are characterized by good tracking ability to significant profile changes.

On the other hand, when the initial block size is out of the near-optimal region, adaptive gain policies have nice transient and stability properties but they are quite sensitive to noise and non-smooth profile shapes. They require finer and more time-consuming tuning of more than one parameters to perform acceptably, and still they cannot reach good robustness properties or reliable tracking ability. This is attributed to the non-smoothness of the profiles which, in the absence of a model capturing clear shape trends, or even the absence of such definite trends, results in poor approximations of the real derivatives by finite differences leading to stagnancy and failure of the corresponding algorithms. However, further simulations with smooth profiles showed that in this case they are characterized by superior performance, good robustness properties and tracking ability.

Although the simulation results in Section 4.2 have been demonstrated using data from Query 1, all relevant observations and remarks have been validated for the rest of the queries, as well. Before concluding this section, a further note deserves special mention, i.e., how the simulations results compare with the results of the empirical evaluation. It is true that some of the problematic issues identified in the simulation runs have not been observed in real experiments, and conversely policies and heuristics identified as less appropriate in empirical evaluation have been found robust and attractive in the simulations. Our remarks are summarized as follows:

—Empirical evaluation identifies the adaptive gain policies as more efficient than constant gain policies, which have been found more reliable in terms of robustness (in the sense that they are not based on proper or continuous tuning of their parameters to perform well) and tracking ability by simulation experimentation. When the initial block size is within the near optimal region, or close to it, the empirical and simulation results are in agreement, essentially. If this is not the case, although empirical evaluation shows that adaptive gain policies are robust as well (Section 4.1.4), this observation cannot be verified by simulations. Possible explanations are given below.

—The simulations have been performed with volatile and discontinuous averaged data profiles corrupted by noise, whereas real profiles are expected to be slightly smoother. This proved to be the main reason for the incompetence of schemes relying on derivative calculations, i.e., Newton policies or the poor robustness and tracking properties of the adaptive gain policies in some cases, as well. Careful tuning of the adaptive gains cannot resolve this problem.

—In real cases, as those dealt with in Section 4.1, the degraded performance of the constant gain schemes should be attributed to the constantly changing environment they create. As implied also by the data in Tables I and II, frequent and significant block size changes give rise to transients –perhaps by exciting unknown and unobservable system dynamics, governed by internal server behavior– which are difficult to simulate and cannot be easily handled by an optimization algorithm, unless we ensure that they have died out, e.g., by slow sampling. This is not the case in our implementation, and we expect all policies to suffer from this phenomenon. During the transient phase of their response all policies generate large steps, and this phenomenon does not prevent them from reaching the near-optimal region quickly. Unfortunately, the phenomenon's influence plays a crucial role during the steady state phase, i.e., around the optimal point. The adaptive gain policies are influenced only to a small extent since they converge quickly and maintain an almost constant value since their gains become very small, whereas the constant gain policies can be heavily influenced because they still attain the same large gains and therefore sustained oscillations of a large amplitude. Similarly, the use of a large dither signal can further excite these dynamics and transients.

—In either experimentation efforts we have experimented with different parameter values, and responses with similar characteristics have been specified for most policies. The fact that for a reasonably good performance (that is, convergence ability to the optimum point), gains of different order of magnitude have been found should not be considered as problematic, since by simple scaling the results can be reasonably matched.

It becomes obvious that the fruitful combination of empirical evaluation and simulation results is particularly enlightening, as regards the applicability and efficiency of the proposed techniques to the problem under question in this paper. Furthermore, it paves the way for potential future improvements; the key issues that need to be tackled have been identified.

5. DISCUSSION OF RELATED WORK AND POINTERS TO FUTURE RESEARCH

There is a recent booming in applying control theory in computing systems, software engineering and software services [Hellerstein et al. 2005], [Abdelzaher et al. 2003]. This is due to the trend of going beyond ad-hoc and heuristic techniques towards an autonomic computing paradigm [Diao et al. 2005]. Exploitation of the rich arsenal of techniques, methods, ideas and foundations of control theory, developed for many decades since the second world war, has already led to improved designs in many areas and problems [Abdelzaher et al. 2002; Gandhi et al. 2002; Lu et al. 2005].

Furthermore, for preserving QoS, optimization approaches have been also developed in many works, where dynamic tuning of several configuration parameters that are related to the performance of computing systems is suggested. More specifically, online minimization of the response time of an Apache web server by dynamic tuning of the number of maximum clients allowed to be connected simultaneously is described in [Liu et al. 2003b], where hill climbing and fuzzy control techniques are employed. For a database server, online adjustment of multiple configuration parameters using online random and direct search techniques is proposed in [Diao et al. 2003] to guarantee good performance. For application servers, optimal configurations have also been sought in [Raghavachari et al. 2003] using off-line experimentation and statistical analysis.

Other recent works where optimization problems are dealt with are the efforts described in [Stanojevic et al. 2006; Stanojevic and Shorten 2007]. The problem there is the optimal choice of the buffer size in the Internet routers, so that minimum queueing delays and maximum utilization (or any desired trade-off) is achieved. Adaptive online tuning of the buffer size is suggested and iterative MIMD (multiplicative increase-multiplicative decrease) algorithms are proposed. Such schemes, including the AIAD (additive increase-additive decrease) and AIMD choices are used in networking and congestion control problems, and are inherently linear. They are not suitable to our case study, since they do not converge but rather search continuously for the optimum. They are usually slowly responsive due to their small step sizes. Larger steps can speed up the transient behavior at the expense of undesirable large oscillations during the steady-state. This behavior is problematic in our case study, as our experimentation with constant gain switching extremum control schemes has shown.

From the control theory point of view, extremum control has been employed from the early stages of control theory development [Draper and Y.Li 1954; Blackman 1962], but is still a subject under development [Ariyur and Krstic 2003; Killingsworth and Krstic 2006; Choi et al. 2002; Krstic and Wang 2000]. A review of related techniques can be found in [Larsson 2001]. There are two basic families, gradient and parametric extremum control methods. Gradient methods usually appear in two flavors, switching and perturbation extremum control techniques. Gradient methods are based upon model-free gradient approximate computations whereas parametric methods rely on parametric models and online parameter identification. The former are lighter and more related to hill climbing and gradient descent optimization schemes, while the latter employ self-tuning and adaptive control ideas and are more complicated and computationally demanding.

Although applied in many engineering systems, to the author's knowledge this work is the first application to the WSMS problems described in our context. For non-engineering problems, in a totally different context, an extremum control approach has only recently appeared in [O. Flardh and Johansson 2005] for the problem of error correction in packet-switched networks. Feedback and feedforward control techniques based upon prediction are employed. In the absence of any additional information, switching extremum control is suggested, facilitated with filtering and averaging to deal with the noisy and frequently changing measurement data.

It is obvious that many other techniques from the fields of numerical optimization and extremum control could be used. More specifically, as a subject for future work, we should perhaps focus on our profiles's shapes and observe that, despite the volatility, local peaks, jitter etc. there is overall a very clear picture, which can be represented by a smooth quadratic (or sometimes monotonically decreasing) concave curve. The experiments in Simulation 3 suggest one possibility for further improvements. If we manage to fit our data to develop smooth profiles (e.g., by constructing models with parameter identification) without sacrificing good robustness and generalization properties, the adaptive gain policies could reach improved performance. This has to be done in a computationally tractable manner, while allowing early detection and fast response to changes. Furthermore, hybrid schemes combining adaptive with constant gain policies could also lead to improved results. In this case, simple mechanisms for accurate and reliable detection of the different response phases have to be developed. It is a subject of future work to experiment with new schemes and test their applicability in special cases, including concurrent requests.

6. CONCLUSIONS

This paper describes algorithms for the online adjustment of block size for enhanced transmission of large data volumes in Web Service Grids following a control theoretical approach. The algorithms analyze the behavior of past values for the block size in order to determine the future configurations. The algorithms fall into two broad areas: runtime optimization inspired by hill-climbing techniques, and switching extremum control. As expected the former category is outperformed by the latter, which is more suitable for systems exhibiting noisy, non-monotonical behavior. For the latter category, we distinguish between techniques employing constant gain and those employing adaptive gains. The trade-offs between these two types can be summarized as follows. Adaptive gain policies seem to be the most suitable choice when the near optimal region can be approximated. However, in this case the performance benefits may not exceed a 10% decrease in response times. Larger improvements, over 100% decrease in performance degradation caused by suboptimal choice of block sizes, can be provided when this region is a priori unknown. In this case, adaptive gain policies have nice transient and stability properties but they are still quite sensitive to noise and non-smooth profile shapes. On the other hand, constant gain policies can perform well even without fine tuning, but their transient behavior and steady state stability may deviate from the optimum point.

In summary, the results of this work render the process of calling services self-optimizing, and detailed WS profiling and fine tuning obsolete. Detailed experiments in a real environment, complemented by simulations that can investigate additional configurations, demonstrate the efficiency and the effectiveness of the presented solutions and prove that these are capable of reducing significantly the response time (especially in the case where the optimum region of block sizes can not be a priori approximated) and can be characterized by all four main desired properties for self-managing systems, i.e., stability, accuracy, speed of convergence, and overshoot avoidance [Diao et al. 2005]. The techniques presented are applicable to any similar optimization problem where the entity to be configured remotely exhibits a non-monotonic, concave behavior; OGSA-DAI services are presented merely as a case study.

ACKNOWLEDGMENTS

This work has been supported by the EU-funded CoreGrid Network of Excellence project through grant FP6-004265. Dr. Yfoulis has been supported by the ATEI grant 6/24/4-7-2007 titled "Adaptive QoS control and optimization of computing systems". This work has been conducted while A. Gounaris was holding a visiting lecturer position with the University of Cyprus.

REFERENCES

Abdelzaher, T. F., Shin, K. G., and Bhatti, N. T. 2002. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Trans. Parallel Distrib. Systems 13,* 1, 80–96.

Abdelzaher, T. F., Stankovic, A., Lu, C., Zhang, R., and Lu, Y. 2003. Feedback performance control in software services. *IEEE Control Systems Magazine 23,* 3.

Alpdemir, M. N., Mukherjee, A., Paton, N. W., Watson, P., Fernandes, A. A. A., Gounaris, A., and Smith, J. 2003. Service-based distributed querying on the grid. In *Proc. of 1st International Conference on Service Oriented Computing - ICSOC.* Springer, 467–482.

Alpdemir, N., Gounaris, A., Mukherjee, A., Fitzgerald, D., Paton, N. W., Watson, P., Sakellariou, R., Fernandes, A. A., and Smith, J. 2005. Experience on Performance Evaluation with OGSA-DQP. In *Proceedings of the UK e-Science All Hands Meeting.*

Antonioletti, M. et al. 2005. The design and implementation of grid database services in OGSA-DAI. *Concurrency - Practice and Experience 17,* 2-4, 357–376.

Ariyur, K. and Krstic, M. 2003. *Real-Time Optimization by Extremum-Seeking Control.* John Wiley & Sons.

Blackman, P. 1962. *Extremum-seeking regulators : an exposition of adaptive control.* Pergamon Press.

Choi, J., Krstic, M., Ariyur, K., and Lee, J. 2002. Extremum seeking control for discrete-time systems. *IEEE Transactions on Automatic Control 47,* 2, 318–323.

Diao, Y., Eskesen, F., Forehlich, S., Hellerstein, J., Spainhower, L., and Surendra, M. 2003. Generic online optimization of multiple configuration parameters with application to a database server. *DSOM*, 3–15. LNCS 2867.

Diao, Y., Hellerstein, J. L., Parekh, S. S., Griffith, R., Kaiser, G. E., and Phung, D. B. 2005. Self-managing systems: A control theory foundation. In *Proc of IEEE International Conference and Workshop on the Engineering of Computer Based Systems ECBS 2005.* 441–448.

Dobrzelecki, B., Antonioletti, M., Schopf, J., Hume, A., Atkinson, M., Hong, N. C., Jackson, M., Karasavvas, K., Krause, A., Parsons, M., Sugden, T., and Theocharopoulos, E. 2006. Profiling OGSA-DAI Performance for Common Use Patterns. In *Proceedings of the UK e-Science All Hands Meeting.*

DRAPER, C. AND Y.LI. 1954. *Principles of Optimizing Control Systems.* ASME Publications.

GANDHI, N., HELLERSTEIN, J., TILBURY, D., AND JAYRAM, T. 2002. Using control theory to achieve service level objectives in performance management. *Real-Time Systems 23*, 127–141.

GOUNARIS, A., SAKELLARIOU, R., PATON, N. W., AND FERNANDES, A. A. A. 2006. A novel approach to resource scheduling for parallel query processing on computational grids. *Distributed and Parallel Databases 19,* 2-3, 87–106.

GOUNARIS, A., SMITH, J., PATON, N. W., SAKELLARIOU, R., FERNANDES, A. A. A., AND WATSON, P. 2005. Adapting to changing resource performance in grid query processing. In *Data Management in Grids, First VLDB Workshop, DMG 2005.* 30–44.

GOUNARIS, A., YFOULIS, C., SAKELLARIOU, R., AND DIKAIAKOS, M. D. 2007. Self-optimizing block transfer in web service grids. In *WIDM '07: Proceedings of the 9th annual ACM international workshop on Web information and data management.* ACM, 49–56.

HELLERSTEIN, J., DIAO, Y., PAREKH, S., AND TILBURY, D. 2005. Control engineering for computing systems. *IEEE Control Systems Magazine 25,* 6, 56–68.

KILLINGSWORTH, N. AND KRSTIC, M. 2006. PID tuning using extremum seeking. *IEEE Control Systems Magazine*, 70–79. February 2006.

KOSAR, T. AND LIVNY, M. 2004. Stork: Making data placement a first class citizen in the grid. In *24th International Conference on Distributed Computing Systems (ICDCS 2004), 24-26 March 2004, Hachioji, Tokyo, Japan.* IEEE Computer Society, 342–349.

KRSTIC, M. AND WANG, H. 2000. Stability of extremum seeking feedback for general nonlinear dynamic systems. *Automatica 36*, 595–601.

LARSSON, S. 2001. Literature study on extremum control. Tech. rep., Chalmers University of Technology.

LIU, D. T., FRANKLIN, M. J., AND PAREKH, D. 2003a. Griddb: A database interface to the grid. In *Proceedings of ACM SIGMOD*, A. Y. Halevy, Z. G. Ives, and A. Doan, Eds. ACM, 660.

LIU, X., SHA, L., DIAO, Y., FROEHLICH, S., HELLERSTEIN, J. L., AND PAREKH, S. S. 2003b. Online response time optimization of apache web server. In *IWQoS*. 461–478.

LU, C., WANG, X., AND KOUTSOUKOS, X. D. 2005. Feedback utilization control in distributed real-time systems with end-to-end tasks. *IEEE Trans. Parallel Distrib. Systems 16,* 6, 550–561.

NARAYANAN, S., CATALYREK, U. V., KURC, T. M., ZHANG, X., AND SALTZ, J. H. 2003. Applying database support for large scale data driven science in distributed environemnts. In *Proc. of the 4th Workshop on Grid Computing, GRID'03.*

O. FLARDH, K. J. AND JOHANSSON, M. 2005. A new feedback control mechanism for error correction in packet-switched networks. 488–493. IEEE Conference on Decision and Control.

PERSINNI, A. 1988. *The Mathematics of Nonlinear Programming.* Springer-Verlag.

RAGHAVACHARI, Y., REIMER, D., AND JOHNSON, R. 2003. The deployer's problem: Configuring application servers for performance and reliability. 3–15. ICSE.

SESHASAYEE, B., SCHWAN, K., AND WIDENER, P. 2004. Soap-binq: High-performance soap with continuous quality management. In *ICDCS*. 158–165.

SRIVASTAVA, U., MUNAGALA, K., WIDOM, J., AND MOTWANI, R. 2006. Query optimization over web services. In *VLDB*. 355–366.

STANOJEVIC, R., KELLET, C., AND R.N.SHORTEN. 2006. Adaptive tuning of drop-tail buffers for reducing queueing delays. *IEEE Communications Letters 10,* 7.

STANOJEVIC, R. AND SHORTEN, R. 2007. How expensive is link utilization. Technical Report, available at http://www.hamilton.ie./person/rade/QP.pdf.

WELLSTEAD, P. AND M.B.ZARROP. 1995. *Self tuning systems: control and signal processing.* John Wiley & Sons.

Received June 2007; accepted February 2008.