

Self-optimizing Block Transfer in Web Service Grids^a

^aTo appear in the Proceedings of WIDM 2007.

Anastasios Gounaris, Marios D. Dikaiakos
{gounaris, mdd}@cs.ucy.ac.cy
Department of Computer Science - University of Cyprus, Cyprus

Christos Yfoulis
{cyfoulis}@teithe.gr
Department of Automation - ATEI of Thessaloniki, Greece

Rizos Sakellariou
{rizos}@cs.man.ac.uk
School of Computer Science - University of Manchester, UK



CoreGRID Technical Report
Number TR-0113
October 4, 2007

Institute on Knowledge and Data Management
Institute on System Architecture

CoreGRID - Network of Excellence
URL: <http://www.coregrid.net>

Self-optimizing Block Transfer in Web Service Grids

Anastasios Gounaris, Marios D. Dikaiakos

{gounaris, mdd}@cs.ucy.ac.cy

Department of Computer Science - University of Cyprus, Cyprus

Christos Yfoulis

{cyfoulis}@teithe.gr

Department of Automation - ATEI of Thessaloniki, Greece

Rizos Sakellariou

{rizos}@cs.man.ac.uk

School of Computer Science - University of Manchester, UK

CoreGRID TR-0113

October 4, 2007

Abstract

Nowadays, Web Services (WSs) play an increasingly important role in Web data management solutions, since they offer a practical solution for accessing and manipulating data sources spanning administrative domains. Nevertheless, they are notoriously slow and transferring large data volumes across WSs becomes the main bottleneck in such WS-based applications. This paper deals with the problem of minimizing at runtime, in a self-managing way, the data transfer cost of a WS encapsulating a data source. To reduce the transfer cost, the data volume is typically divided into blocks. In this case, response time exhibits a quadratic-like, non-linear behavior with regards to the block size; as such, minimizing the transfer cost entails finding the optimum block size. 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 WSs for accessing and integrating traditional DBMSs. The main challenges in this problem include (i) the unavailability of an analytical model; (ii) the presence of noise, which incurs local minima; (iii) the volatility of the environment, which results into a moving optimum operating point; and (iv) 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. This paper presents two novel solutions for detecting the optimum block size during data transmission, thus yielding lower response times. The solutions are inspired by the broader areas of runtime optimization and switching extremum control. They incorporate heuristics to avoid local optimal points, and address all the afore-mentioned challenges. The effectiveness and efficiency of the solutions is verified through empirical evaluation in real cases.

1 Introduction

The proliferation of Web Service-based Grids and the increasingly growing volume of data that is processed by and shared among such web data management applications 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., [16, 3, 19]), wide area query optimization (e.g., [22, 14]) and the associated resource scheduling decisions (e.g., [13]). However, an important factor is the optimization of the data transfer cost for WSs that encapsulate data sources. Widely adopted standards for WS communication, such as the XML-based SOAP,

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

facilitate remote communication but are rather inefficient for transferring large volumes of data, and as such, web data management solutions based on WSs suffer from high data transmission costs, which lead to poor performance. To reduce this type of cost, a typical approach is to split large data volume into several blocks. The block-based data transmission cost is largely dependent both on the size of the request (i.e., the size of the data block transferred between services) [22, 4], and on the network bandwidth. More specifically, the performance of a remote server serving a series of calls with a fixed total size, in terms of the task response time, is characterized by a noisy (due to the volatile network conditions), concave graph with regards to the size of the data chunks or blocks. Such a graph has a different optimal point for different queries and/or different connections, or even different stages of the same query. The problem that this paper deals with is to detect at runtime this optimal point in a self-optimizing way.

Consider for example the case in which a database is globally exposed as a WS through OGSA-DAI wrappers [5]. 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 of OGSA-DAI), then it must be returned in chunks that can be specified by the recipient to avoid out-of-memory errors and speed up transmission. As reported also in [4], the response time first decreases when the block size is increased and after a point it starts increasing. This point is different for each query-connection combination. Similar behavior for other data management WSs is reported also in [22]. 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 [17].

In this paper we choose OGSA-DAI as our case study. 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. We follow a control theoretical approach and so, the self-optimizing entity, which resides at the client, is referred to as the controller and the WS as the controlled entity. 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 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 [17]) 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 (i.e., the block size for which the time for the complete data transmission is minimized), rendering naive hill climbing techniques non-appealing. Finally, the convergence of any algorithm must be fast; otherwise serious performance degradation is not avoided.

The contribution of this paper is to present solutions to the afore-mentioned problem that meet the requirements mentioned above. More specifically, the main contribution of this work is two-fold.

- Firstly, to present fast and robust optimization algorithms that belong to the area of *runtime optimization* and *switching extremum control* and that are capable of converging to the optimal block size quickly despite the presence of local optimum regions, noise, and bad choices for the starting point.
- Secondly, to apply these 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.

The remainder of this paper is structured as follows. The next section presents the OGSA-DAI approach and some measurements that motivated the research described hereby. The solution to the optimization problem is presented in Section 3. Section 4 deals with the evaluation. Section 5 discusses related work, and Section 6 concludes the paper.

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) [5]. A single Data Service can provide access to multiple data

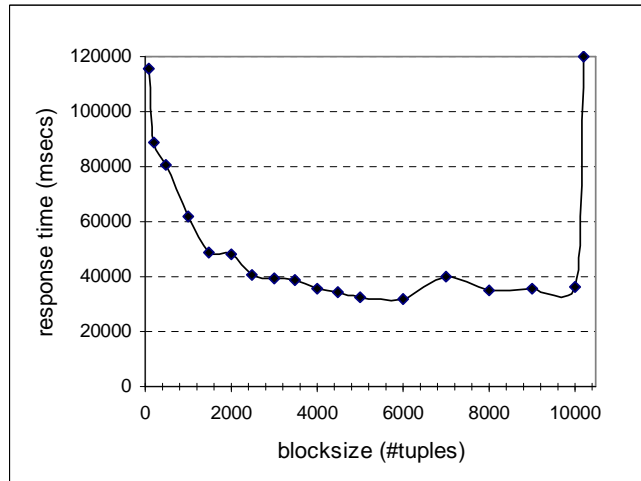


Figure 1: Response times for a local query for different block sizes.

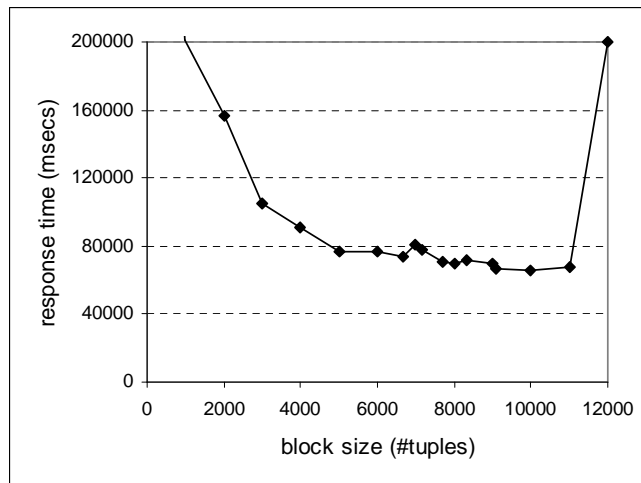


Figure 2: Response times for a remote query for different block sizes.

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.

Figure 1 illustrates the response times for a query returning to a local client 100000 tuples of 100 bytes each, when the data block sizes are fixed for the whole duration of the query. 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

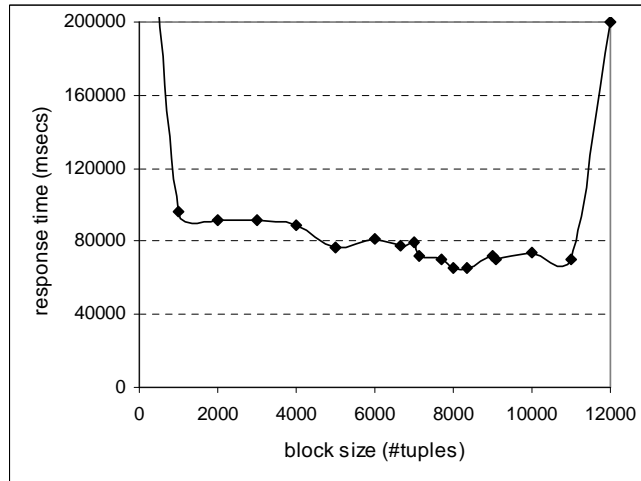


Figure 3: Response times for a remote query for different block sizes (unstable connection).

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 1: Summary of response times for a local query for different block sizes.

the server. The WSRF¹ 2.2 flavor of OGSA-DAI is used. 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). 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 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 decreases (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 happens for 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

¹http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

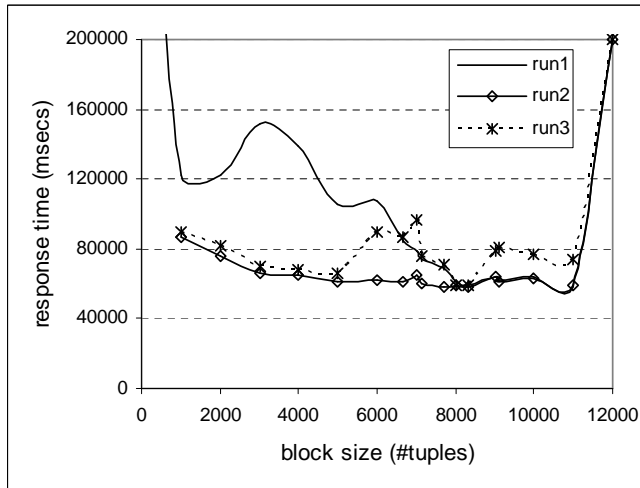


Figure 4: Response times of individual runs of a remote query for different block sizes (unstable connection).

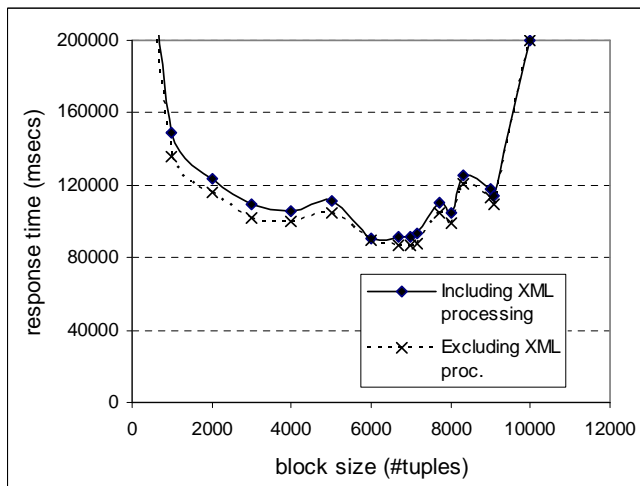


Figure 5: Response times for a remote query for different block sizes (unstable connection, double tuple size).

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 2: Summary of response times for a remote query for different block sizes.

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 been modified 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 figures up to this point. 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 in Tables 1 and 2, 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 due to the small number of overall cycles.

3 Online Adjustment

The high level architecture assumed is that the controller at the client analyzes the response time of the WS for previous block sizes, and based on this information, continuously (i.e., at each step) adjusts the size for the next data block to be requested. If y is the performance metric, such as response time or equivalently, the per tuple cost in time units, and x the size of the data block, it is usually assumed that, at least in the neighborhood of the optimal point, there is a quadratic function $f(x) = a(x - x_o)^2 + b$, where a, b are unknown constants, for which $y = f(x) + e$. e represents the noise, and we further assume that e is responsible for the local peaks on both sides of the optimal block size. The optimal point is the value of x for which the first partial derivative of $f(x)$ is 0 (i.e., $\nabla f(x) = 0$), which is obviously x_o .

In this paper, two main approaches are investigated. The first is a typical numerical optimization method, while the second comes from the field of extremum control. The former is inspired by the Newton's method [20], which defines that the value of x at the k th step, x_k is given by the following formula

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

The first partial derivative allows the next block size to move towards the optimum point, whereas the second partial derivative is for faster convergence. However, the main drawback is that the Newton's method is known to be very sensitive to noise, and in the case examined, the noise is 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 approximate estimate of the partial derivatives using backward difference operators $\Delta u = u_k - u_{k-1}$, i.e. $\nabla f(k) \simeq \frac{\Delta y}{\Delta x} = \frac{y_k - y_{k-1}}{x_k - x_{k-1}}$.

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. 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, the value of the block size at each step is estimated as follows:

$$x_k = \bar{x}_{k-1} - \frac{\Delta \bar{y}_{k-1}}{\Delta^2 \bar{y}_{k-1}} + d(k), \quad \{\bar{x}_k, \bar{y}_k\} = \frac{1}{n} \sum_{i=k}^{k-n+1} \{x_i, y_i\} \quad (2)$$

The second approach investigated is inspired by extremum control [7], 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. 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 [6, 23].

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

Table 3: The characteristics of the example queries.

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

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

The formula above can detect the side of the optimum point where the current block size resides on. The $\text{sign}(x)$ function returns 1 if x is positive, and -1 otherwise. The rationale of the formula 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) tuning parameter. Without applying a dither signal, the step size is always the same, and since $\|\Delta x\| = b_1$, b_1 defines the rate at which x is modified. In the second flavor

$$g = b_2 \left\| \frac{\Delta \bar{y}_{k-1}}{\bar{y}_{k-1}} \Delta \bar{x}_{k-1} \right\|, \quad b_2 > 0 \quad (4)$$

where b_2 is constant. In this case, the step (gain) is adaptive and is proportional to the product of the performance change and the change in the block size. In both approaches, maximum and minimum limits can be imposed to avoid overshooting with detrimental effects.

4 Prototype Implementation and Evaluation

To test the actual performance of the techniques described, a thin client is built that can submit SQL queries to an OGSA-DAI Data Service and request results to be delivered in blocks, the size each of which is determined by a controller encapsulated in the client. The data comes from the TPC-H database (scale 1) stored in a MySQL DBMS. Five queries are used throughout as shown in Table 3. Note that the actual tuple length communicated across the network is significantly increased by the XML tags, as reported in [10]. Since we are interested in the data transmission cost and not in the cost to evaluate the queries within the WS, 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 there is a single client in Greece. 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 and the WS was not serving third party requests during the period of experiments. The complete set of experiments presented here lasted for 10 days approximately around the 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 that is not due to a bad approach to experimentation but to the volatility of the environment; running more experiments using non-dedicated resources at arbitrary time periods, as in these experiments, would not tackle this phenomenon. 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 6. The fixed block sizes used to produce this profiling figure are 1K, 5K, 6K, 7K, 8K, 9K and 10K tuples. At first sight, when the initial decision on the block size is clearly suboptimal, i.e. around 1000 tuples, an adaptive method can yield significant performance benefits²; to the

²The main adopter of OGSA-DAI, OGSA-DQP [3] suffers from such a suboptimal decision in its current release.

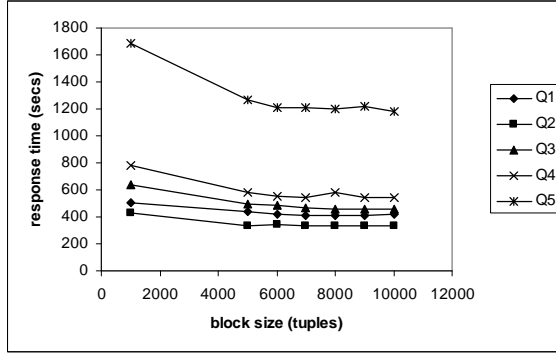


Figure 6: Response times for queries Q1-Q5 for fixed block sizes.

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 4: 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 5: Comparison of adaptive policies.

contrary 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%.

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
dyna-mic	0.989	0.9945	0.9436	0.9764	0.8922	0.9591

Table 6: Comparison of dynamic adjustment of block size against fixed size policies.

4.1 Comparison of Adaptive Techniques

Table 4 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 is set to 3. Minimum and maximum value constraints are imposed, set to 1000 and 10000, respectively.

The comparison of the techniques is shown in Table 5, which presents the normalized response times for each of the adaptivity configurations of Table 4. The lowest response time 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 6, 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.
- An additional observation is that, somewhat counter-intuitively, the effects of 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 1.02 times worse response times than the best policy (which is unknown from before), whereas the best SEC policy with both adaptive gain and dither signal yields 1.0652 times 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 to 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 then we would result in moving the problem from fine-tuning the block size to fine tuning the adaptivity parameters.

4.2 Performance Improvements

So 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 6. In this table, the values are normalized with the optimum point of Figure 6 set to 1. The last row depicts the relative performance of the most effective policy for each query, as shown in Table 5. The overhead is included in this time and, from the table, it can be inferred that it is negligible. We can observe that:

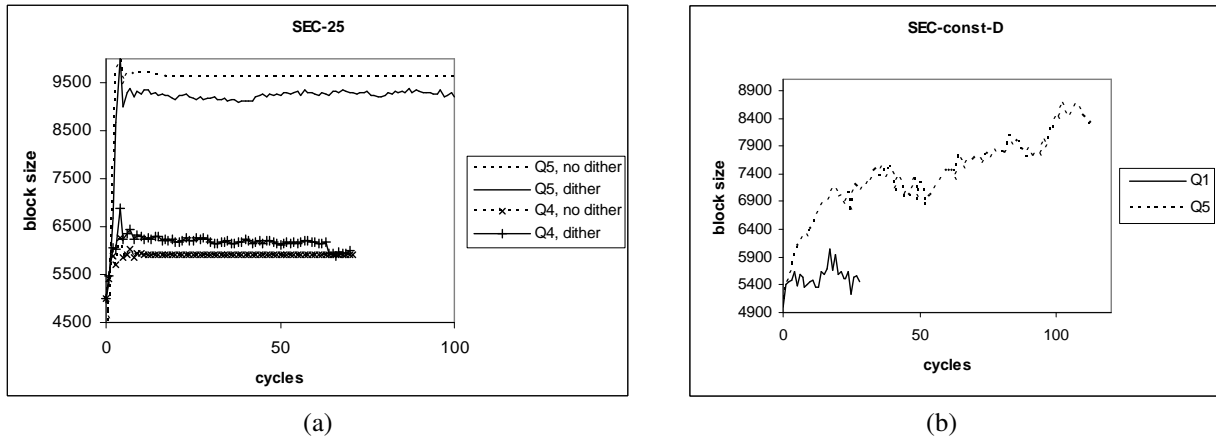


Figure 7: The block sizes at different adjustment cycles (a) for Q4, Q5 when SEC is employed with $b_2 = 25$ and (b) for Q1, Q5 when SEC-const-D is employed.

- 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 4.4), 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 is translated in several minutes in real time units (given that all queries and especially Q5 are rather expensive and long running as shown in Figure 6).
- The starting point 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.

4.3 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 7(a). 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 7(a)). The fast convergence property does not hold for the first flavor of SEC, as shown in Figure 7(b). SEC-const-D seem to require more cycles to converge than the complete length of the query execution.

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 7: Comparison of the performance of dynamic adjustment of block size when the initial block size is clearly suboptimal.

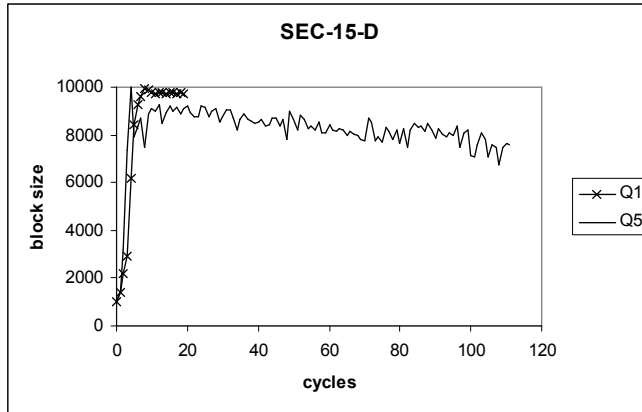


Figure 8: The block sizes at different adjustment cycles when SEC-15-D is employed and the starting point is 1000 tuples.

4.4 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 performance degradation of such a suboptimal decision is more severe than in a WAN 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 7 summarizes the results in normalized time units. 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 8; 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.

5 Related Work

There is a recent booming in applying control theory in computing systems, software engineering and software services [15], [2]. This is due to the trend of going beyond ad hoc and heuristic techniques towards an autonomic computing paradigm [9]. 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 [1, 12, 18]. Furthermore, to meet 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 [17], 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 [8] to guarantee good performance. For application servers, optimal configurations have also been sought in [21] using off-line experimentation and statistical analysis. From the control theory point of view, extremum control has been employed from the early stages of control theory development. Although applied in many engineering systems, to the authors' knowledge this work is the first application to the Web Service Management Systems (WSMS) problems described in our context. For non-engineering problems, in a totally different context, an extremum control approach has only recently appeared in [11] for the problem of error correction in packet-switched networks.

6 Conclusions

This paper presents algorithms for the online adjustment of block size for enhanced transmission of large data volumes in Web Service Grids following a control theoretical approach. These algorithms yield lower response times for retrieving large data volumes from WSs, which is the main bottleneck in service-oriented web data management applications. More specifically, they can produce significant benefits in the response time in the generic case, where the optimal region of block size is not a priori known; in this case the response time can be reduced to the half or less. Moreover, the algorithms can yield improved performance even in the more limited scenario where this region can be approximated. The results of this work render the process of calling services self-optimizing, and detailed WS profiling and fine tuning obsolete. The algorithms analyze the behavior of past values for the block size in order to determine the future configurations. Detailed experiments demonstrate the efficiency and the effectiveness of the presented solutions and prove that, apart from reducing significantly the response time, they are characterized by all four main desired properties for self-managing systems, i.e., stability, accuracy, speed of convergence, and overshoot avoidance [9]. The techniques presented are applicable to any similar optimization problem where the entity to be configured remotely exhibits a quadratic behavior; OGSA-DAI services are presented merely as a case study.

Apparently, other techniques inspired from different fields of numerical optimization and extremum control may be applicable. At first sight, the aforementioned problematic issues that are present in our case study leave little scope for the use of model-based techniques, especially for short-lived transfer tasks and highly variable conditions. However, under some further assumptions and restrictions, these methods might be able to provide improved results. Hybrid schemes, combining ideas from different methods, especially suited to our needs, will also be investigated. It is a subject of future work to experiment with such new schemes in more settings, including concurrent WS requests. Initial results when there are concurrent requests to a WS show that the near optimal region is narrower and the performance benefits even more significant.

7 Acknowledgments

This work has been supported by the EU-funded CoreGrid Network of Excellence project through grant FP6-004265. C. Yfoulis was supported by the ATEI grant titled "Adaptive QoS control and optimization of computing systems." A.Gounaris is also with the University of Manchester as a part-time researcher.

References

- [1] T. F. Abdelzaher, K. G. Shin, and N. T. Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Trans. Parallel Distrib. Systems*, 13(1):80–96, 2002.
- [2] T. F. Abdelzaher, A. Stankovic, C. Lu, R. Zhang, and Y. Lu. Feedback performance control in software services. *IEEE Control Systems Magazine*, 23(3), 2003.
- [3] M. N. Alpdemir, A. Mukherjee, N. W. Paton, P. Watson, A. A. A. Fernandes, A. Gounaris, and J. Smith. Service-based distributed querying on the grid. In *Proc. of 1st Int. Conf. on Service Oriented Computing - ICSOC*, pages 467–482. Springer, 2003.
- [4] N. Alpdemir, A. Gounaris, A. Mukherjee, D. Fitzgerald, N. W. Paton, P. Watson, R. Sakellariou, A. A. Fernandes, and J. Smith. Experience on Performance Evaluation with OGSA-DQP. In *Proceedings of the UK e-Science All Hands Meeting*, 2005.

- [5] M. Antonioletti et al. The design and implementation of grid database services in OGSA-DAI. *Concurrency - Practice and Experience*, 17(2-4):357–376, 2005.
- [6] K. Ariyur and M. Krstic. *Real-Time Optimization by Extremum-Seeking Control*. John Wiley & Sons, 2003.
- [7] K. J. Åström and B. Wittenmark. *Adaptive Control*. Addison-Wesley, Reading, MA, USA, 1995.
- [8] Y. Diao, F. Eskesen, S. Forehlich, J. Hellerstein, L. Spainhower, and M. Surendra. Generic online optimization of multiple configuration parameters with application to a database server. *DSOM*, pages 3–15, 2003.
- [9] Y. Diao, J. L. Hellerstein, S. S. Parekh, R. Griffith, G. E. Kaiser, and D. B. Phung. Self-managing systems: A control theory foundation. In *Proc of IEEE International Conference and Workshop on the Engineering of Computer Based Systems ECBS 2005*, pages 441–448, 2005.
- [10] B. Dobrzelecki, M. Antonioletti, J. Schopf, A. Hume, M. Atkinson, N. C. Hong, M. Jackson, K. Karasavvas, A. Krause, M. Parsons, T. Sugden, and E. Theocharopoulos. Profiling OGSA-DAI Performance for Common Use Patterns. In *Proceedings of the UK e-Science All Hands Meeting*, 2006.
- [11] O. Flardh, K. Johansson and M. Johansson. A new feedback control mechanism for error correction in packet-switched networks. pages 488–493, 2005. IEEE Conference on Decision and Control.
- [12] N. Gandhi, J. Hellerstein, D. Tilbury, and T. Jayram. Using control theory to achieve service level objectives in performance management. *Real-Time Systems*, 23:127–141, 2002.
- [13] A. Gounaris, R. Sakellariou, N. W. Paton, and A. A. A. Fernandes. A novel approach to resource scheduling for parallel query processing on computational grids. *Distributed and Parallel Databases*, 19(2-3):87–106, 2006.
- [14] A. Gounaris, J. Smith, N. W. Paton, R. Sakellariou, A. A. A. Fernandes, and P. Watson. Adapting to changing resource performance in grid query processing. In *Data Management in Grids, First VLDB Workshop, DMG 2005*, pages 30–44, 2005.
- [15] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury. Control engineering for computing systems. *IEEE Control Systems Magazine*, 25(6):56–68, 2005.
- [16] D. T. Liu, M. J. Franklin, and D. Parekh. Griddb: A database interface to the grid. In A. Y. Halevy, Z. G. Ives, and A. Doan, editors, *Proceedings of ACM SIGMOD*, page 660. ACM, 2003.
- [17] X. Liu, L. Sha, Y. Diao, S. Froehlich, J. L. Hellerstein, and S. S. Parekh. Online response time optimization of apache web server. In *IWQoS*, pages 461–478, 2003.
- [18] C. Lu, X. Wang, and X. D. Koutsoukos. Feedback utilization control in distributed real-time systems with end-to-end tasks. *IEEE Trans. Parallel Distrib. Systems*, 16(6):550–561, 2005.
- [19] S. Narayanan, U. V. Catalyrek, T. M. Kurc, X. Zhang, and J. H. Saltz. Applying database support for large scale data driven science in distributed environemnts. In *Proc. of the 4th Workshop on Grid Computing, GRID'03*, 2003.
- [20] A. Persinni. *The Mathematics of Nonlinear Programming*. Springer-Verlag, 1988.
- [21] Y. Raghavachari, D. Reimer, and R. Johnson. The deployer's problem: Configuring application servers for performance and reliability. pages 3–15, 2003. ICSE.
- [22] U. Srivastava, K. Munagala, J. Widom, and R. Motwani. Query optimization over web services. In *VLDB*, pages 355–366, 2006.
- [23] P. Wellstead and M.B.Zarrop. *Self tuning systems: control and signal processing*. John Wiley & Sons, 1995.