Online load balancing in parallel database queries with model predictive control

Christos A. Yfoulis [#1], Anastasios Gounaris [*2]

[#] *Department of Automation, ATEI of Thessaloniki, Greece*
[1] cyfoulis@autom.teithe.gr

[*] *Department of Informatics, Aristotle University of Thessaloniki, Greece*
[2] gounaria@csd.auth.gr

***Abstract*—This work deals with a commonly encountered instantiation of the more generic problem of runtime load balancing. More specifically, we study load balancing in the context of query plans that include partitioned database operators running on remote nodes with time-varying connection speeds and loads. In such query plans, data is processed by similar operator instances placed on different machines in parallel, e.g., using a cloud infrastructure. The goal is to adaptively distribute the data to respond to changes in the environment, in such a way that all machines finish at the same time. Dynamic modifications in the data distribution policy are easily enforced, but they result in non-negligible adaptation overheads due to associated data movements. The emphasis of our approach is on taking into account the inherent cost of adaptations with a view to avoiding over-reacting, which may lead to serious performance degradation. We follow a rigorously founded control theoretical approach to this problem and we present a Model Predictive Control (MPC) controller, which tolerates monitoring inaccuracies and exhibits better performance under periodic and random machine perturbations than other state-of-the-art approaches, as shown by our experiments.**

I. Introduction

Nowadays, more and more tasks are executed remotely making use of cloud infrastructures with a view to attaining scalability and benefitting from the elasticity of such infrastructures. A large portion of such data management tasks are expressed as query plans that are capable of processing both finite and infinite data streams. A practical mechanism to speed-up the plan execution and cope with the vast volume of data is through partitioned parallelism, which scales well with the number of available machines [1]. The execution of a query operator in a plan may benefit from partitioned parallelism when this operator is instantiated several times across different physical nodes with each instance processing a distinct data partition.

Load balancing is a necessary complement to partitioned parallelism. Informally, load balancing in wide area settings is responsible for assigning work to machines in a way that reflects their connection speed and capabilities. This is challenging in most of the real distributed environments, which are usually characterized by at least some of the following characteristics: the machines may be located in arbitrary places and are not dedicated to a single query processing task, which implies that both their connection speed and their load are time-varying because of the concurrent data transfers and tasks, respectively; moreover, the time-varying load is unpredictable. Also, remote infrastructures typically consist of heterogeneous machines, the actual characteristics of which are not necessarily known a-priori. For these reasons, it is not efficient to divide a task consisting of a set of data tuples only statically; instead, dynamic load balancing is required to revise workload allocation decisions at runtime if necessary.

Dividing work into parts that are proportional to current network and machine characteristics (e.g., data transmission and processing rates) is trivial, when such characteristics can be monitored in an accurate and timely manner. Nevertheless, even if this becomes possible, it is not sufficient in common query plans (e.g., those in business intelligence applications) because of operators, such as joins and group-bys that build up internal state (e.g., in the form of a hash table). Such operators are called stateful. A modification in the distribution policy corresponding to stateful operators should be accompanied by movement of internal operator state from one machine to another over the network; otherwise the result is not correct. For example, a redistribution decision regarding a windowed group-by involves the transmission of the partial state of the reallocated groups to their new execution site. Given that the operator state can grow quite large, it is evident that dynamic balancing may incur high overheads that outweigh the benefits of parallelism. Consequently, in wide-area distributed systems, we need to take this overhead into account. To this end, we propose a rigorously-founded self-optimizing solution that adaptively decides on the repartitioning during the execution of pipelined dataflows; note that data partitioning is typically encapsulated by specific operators, such as *exchanges*, that are placed before the operators to be balanced [2], [3].

The approach we follow is founded on applied control theory. In principle, autonomic data management stands to benefit from control theory techniques, which are well-established in engineering fields and are typically accompanied by theoretical investigations of properties such as stability, accuracy, and settling time [4], [5]. A control-theoretical solution to the problem of balancing the load of partitioned operators in queries across multiple heterogeneous machines has appeared in [6], [7], which describes an adaptive multiple-input, multiple-output (MIMO), discrete-time, feedback linear quadratic regulation

(LQR) controller. In general, LQR controllers can encapsulate the cost to enforce a workload redistribution decision (i.e., the cost to move state from one machine to another) along with the cost of deviations from the ideal state in a unified cost function. However, there are certain limitations associated with an LQR methodology. The most important ones are the loss of controllability when the controller manages all the nodes explicitly and the incapability to handle the load balancing inequality and equality constraints that denote that all data must be processed, no negative workload proportion values are allowed and no duplicates should be produced. To deal with these limitations, one physical node had to be excluded from the design and the satisfaction of the constraints relied on heuristics and careful selection of the Q, R weighting matrices in the LQR cost function. These limitations can be overcome by resorting to the MPC (model predictive control) [8] strategy in this paper, which allows direct incorporation of the constraints without the controllability assumption. We propose a light MPC scheme and show that it is suitable for the problem in question; our proposal may deemed as a simplified version of the controller we designed in [9]. Moreover, we believe that our solution is of broader interest, since it can be applied to other load balancing scenarios with similar formulation apart from partitioned database queries.

The contributions of this work are summarized as follows: (i) it introduces a novel MPC-based approach to load balancing in (stateful) parallel database queries and presents a detailed methodology as to how such a controller mechanism is designed; and (ii) it presents its behavior through examples and evaluates its performance compared to the solutions in [7] with the help of a detailed simulator. The evaluation results are conducted in exactly the same setting as in [7] to allow fair comparison.

Structure. In the next section, we briefly discuss related work. Section III formally describes the load balancing problem. The presentation of our new MPC controller is in Section IV. In Section V, we conduct detailed simulated experiments and Section VI concludes the paper.

II. Related Work

Typically, load balancing in queries is either addressed just before execution or, for specific operators, at a single point during execution (e.g., [10], [11], [12]). More adaptive solutions fall into the area of adaptive query processing [13], which deals with self-optimizing execution of queries. However, most of the proposals in this area so far (i) do not consider parallel or distributed execution, thus overlooking runtime load balancing problems; and (ii) do not adhere to any formal model that is amenable to theoretical analysis. The proposal that is closest to our work is Flux [3]. Flux is initially proposed for centralized settings, but can be applied to wide area settings as well. It aims to equalize machine utilization, but does not take into account the adaptation cost during decisions. However, Flux tries to guarantee that the time spent enforcing adaptivity decisions (i.e., moving state from one machine to another as a result of a workload reallocation) does not exceed the time of query processing. In [14], some extensions to the Flux approach are described. Several other dynamic flavors of the load balancing problem in database queries have been examined as well. For example, in [15], each node acts selfishly and autonomously based on bilateral contracts, whereas in [16], load balancing is achieved through the redistribution of operators rather than redistribution of data.

III. Problem Formulation

We assume the execution of a dataflow pipeline, the most expensive operators of which are partitioned. We address load balancing for each partitioned operator individually. The load balancing problem can be formalized as follows. Let P be the degree of intra-operator parallelism of a stateful operator o, e.g., a windowed group-by that processes streams of data. $M = \{m_1, m_2, \cdots, m_P\}$ is the set of P nodes participating in the execution of o. We assume that M does not change during execution and that there exists a monitoring mechanism, which provides updates of the computation and communication characteristics at periodic intervals (steps). The workload proportion that each of these nodes receives at the kth adaptivity step is $u_1(k), u_2(k), \cdots, u_P(k)$, with the constraints $\sum_{i=1}^{P} u_i(k) = 1, \forall k$ and $u_i(k) \geq 0, \forall k$. Each node possesses a certain amount of state. $c_i(k)$ denotes the cost (overhead) to reach the new state required, as a result of a change in $u_i(k)$.

$y_1(k), y_2(k), \ldots, y_P(k)$ define the expected values for the completion time of each of the participating nodes given the workload allocation of the kth adaptivity step. In communication bounded-queries, this cost is dominated by the data transmission cost. In general, it depends on both communication and computation cost. If the query is continuous (e.g., over data streams) the expected completion time refers to the time to complete the evaluation of a fixed-sized workload. The role of the load balancer is, at each step, to minimize the following

$$max(y_i(k+1) + c_i(k+1)), \quad i = 1 \ldots P \tag{1}$$

and estimate $u_i(k+1)$ values accordingly. In a balanced execution, all nodes are expected to finish at the same time, i.e., $y_1(k) = y_2(k) = \cdots = y_P(k)$.

Essentially, the load balancing objective defined in Eq. (1) includes a trade-off between (a) reaching the optimal workload allocation, in which the expected completion times are equalized across all participating nodes, and (b) the cost for reaching such an allocation, which is due to the state movements. To meet such an objective, we employ a state space model, on top of which we implement a state feedback controller, which is designed with the help of an MPC scheme.

IV. Design of the MPC controller

In control theory, system dynamics can be modeled either through transfer functions or through state-space models [4]. In our setting, we have chosen to employ a state-space model, which is scalable with regard to the number of inputs and outputs. We assume the existence of a centralized controller that receives feedback from each machine and controls the

workload distribution policy of the nodes that send data to partitioned operators. The controller's output vector $\mathbf{y}(k)$ is a $P \times 1$ vector with the values of the expected completion time for each node. The input vector $\mathbf{u}(k)$ is a $P \times 1$ vector of our manipulated variables, which are the workload allocations at the kth step.

According to the load balancing requirement, all outputs $y_j\,,\, j = 1, \ldots, P$ should be equalized to their optimal value, which is their average $\overline{y}(k) = \frac{1}{P}\sum_{i=1}^{P} y_i(k)$. Hence we have to design a tracking controller so that the outputs follow a time-varying reference trajectory, defined by the current value of $\overline{y}(k)$.

The limitations of LQR and the special characteristics of the load balancing problem, suggest a natural way forward toward the use of MPC ideas. Such schemes are very popular nowadays for control problems [8], and possess a number of desirable features, such as (i) direct incorporation of constraints mentioned in Sec. III into the design, which is not possible in LQR (where constraints are considered indirectly [7]); and (ii) the fact that the MPC cost function penalizes changes in the input rather than exact input values (as in LQR); this better reflects the transfer cost in our problem.

The Model Predictive control (MPC) approach is capable of dealing with MIMO control problems in the presence of several constraints in an optimal control context. It is a sophisticated algorithm combining performance prediction, optimization, constraint satisfaction, and feedback control. For more details, the reader is referred to [8]. The basic idea of an MPC controller is to optimize an appropriate cost function defined over a time interval in the future, termed as *prediction horizon* H_p. The cost function considers both the deviation from the optimal value, as defined in the reference trajectory (i.e., the *error*), and the cost to modify the input vector. The controller employs a model of the system to predict its future behavior over H_p sampling periods. The main objective of the controller is to generate an input trajectory over H_u sampling periods that does not aim to converge immediately to the reference trajectory but instead to minimize the cost function while satisfying all available constraints; H_u is the *control horizon*. This is exactly the spirit of *optimal control*. The complete design, which leverages the design in [9] is presented below.

We define an appropriate reference trajectory and its associated cost function, and an appropriate approximate model under the load balancing constraints. Our cost function is in the form:

$$\mathcal{J} = \sum_{i=1}^{H_p} \parallel \mathbf{y}(k+i|k) - \mathbf{r}(k+i|k) \parallel^2_{\mathbf{Q}(i)} + \sum_{i=1}^{H_u-1} \parallel \Delta\mathbf{u}(k+i|k) \parallel^2_{\mathbf{R}(i)} \quad (2)$$

The variables $\mathbf{y}(k+i|k),\ \mathbf{r}(k+i|k), \mathbf{u}(k+i|k)$ are the $P \times 1$ output, reference input and input vectors at discrete time $k+i$, respectively, under the assumption that their values at time k are known. The quadratic norm used above is defined as $\parallel \mathbf{x} \parallel^2_M \,=\, \mathbf{x}^T \cdot M \cdot \mathbf{x}$. As mentioned above, the reference trajectory $\mathbf{r}(k)$ in our case is

$$\mathbf{r}(k+i|k) \;=\; \frac{1}{P}\sum_{j=1}^{P} y_j(k+i|k)\ ,\ i = 1, \ldots, P \quad (3)$$

We observe that the MPC cost function penalizes changes in the input rather than input values, which reflects the transfer cost in our load balancing problem more accurately. The optimization consists of specifying the changes that should be applied to our inputs, $\Delta\mathbf{u}(k+i|k)$, along the control horizon, such that $\mathcal{J}$ is minimized while the inequality and equality constraints of the load balancing problem are satisfied.

In order to develop the state-space model, we assume that the outputs coincide with the states, i.e. $\mathbf{y}(k) = \mathbf{x}(k)$. In general, the expected completion times depend on the workload allocated and can be described by a first order difference equation:

$$\mathbf{x}(k+1) = \mathbf{A}(k)\,\mathbf{x}(k) \,+ \mathbf{B}(k)\,\mathbf{u}(k) \quad ,\quad \mathbf{y}(k) = \mathbf{x}(k) \quad (4)$$

In this work, we consider a simplified approximate adaptive model, in which $\mathbf{A}$ is ignored to denote the fact that the expected completion time for each machine depends only on the current capacity captured by $\mathbf{B}$, and the current amount of allocated workload captured by $\mathbf{u}$ (i.e., no transient dynamics are considered). Since the expected completion time of one machine does not depend on the processing speed of another, we can safely assume that $\mathbf{B}$ is diagonal. $\mathbf{B}(k)$ is updated at every step when new feedback measurements become available. Its diagonal elements correspond to the inverse of the throughput of the machines[1]. In this way, $\mathbf{B}(k)$ can capture both changes in the communication and computational capacity, e.g., due to load change. To comply with Eq. (2), through simple manipulations, we transform the approximate model in an incremental form, where $\Delta\mathbf{u}(k)$ is used:

$$\mathbf{x}(k+1) \;=\; \mathbf{x}(k) \;+\; \mathbf{B}(k)\,\Delta\mathbf{u}(k) \quad (5)$$

To deal with the uncertainties and disturbances in our environment, care is required to ensure the presence of integral action and unbiased predictions. Note that, in our setting, disturbance rejection corresponds to tolerating monitoring inaccuracies. The simplest approach is to use a disturbance observer $\mathbf{d}(k)$ on the basis of the constant output disturbance assumption (DMC scheme) [8], i.e. in the form

$$\mathbf{x}(k+1) \;=\; \mathbf{x}(k) \;+\; \mathbf{B}(k)\,\Delta\mathbf{u}(k) \;+\; \mathbf{d}(k) \quad (6)$$

Assuming similarly a constant matrix $\mathbf{B}(k)$ along the prediction horizon, this simple model lends itself to a natural MPC implementation with the following prediction equations for the future states, denoted by $\hat{\mathbf{x}}$:

$$\hat{\mathbf{x}}(k+i|k) \;=\; \mathbf{x}(k) \;+\; \mathbf{B}(k) \sum_{i=0}^{H_p-1} \Delta\mathbf{u}(k+i|k) \;+\; \hat{\mathbf{d}}(k+i|k) \quad (7)$$

[1]In this work, throughput is defined as the inverse of the aggregate time needed to send and remotely process data.

where $\hat{\mathbf{d}}(k+i|k) = \hat{\mathbf{d}}(k|k) = y(k) - \hat{\mathbf{x}}(k|k-1)$, i.e. it is estimated by comparing the measured output with the predicted one. To compute the tracking error term in Eq. (2) we first need to express the predictions in the form

$$Y(k) = \Psi\,\mathbf{x}(k) + \Theta\,\Delta U(k) + \Gamma\,\hat{\mathbf{d}}(k) \tag{8}$$

where $Y(k), \Delta U(k)$ are vectors collecting the variables along the whole horizon, i.e.

$$Y(k) = \begin{bmatrix} \hat{\mathbf{x}}(k+1|k) \\ \vdots \\ \hat{\mathbf{x}}(k+H_p|k) \end{bmatrix}, \tag{9}$$

$$\Delta U(k) = \begin{bmatrix} \Delta\mathbf{u}(k+1|k) \\ \vdots \\ \Delta\mathbf{u}(k+H_u-1|k) \end{bmatrix} \tag{10}$$

with the matrices Ψ, Θ, Γ defined as follows:

$$\Psi = \underbrace{\begin{bmatrix} \mathbf{I}_{PXP} \\ \vdots \\ \mathbf{I}_{PXP} \end{bmatrix}}_{PH_p \times H_p}, \; \Gamma = \underbrace{\begin{bmatrix} \mathbf{I}_{PXP} \\ \vdots \\ \mathbf{I}_{PXP} \end{bmatrix}}_{PH_p \times H_p}, \tag{11}$$

$$\Theta = \underbrace{\begin{bmatrix} \mathbf{B} & \dots & \dots & \mathbf{0} \\ \mathbf{B} & \mathbf{B} & \dots & \mathbf{0} \\ \vdots & \ddots & \ddots & \vdots \\ \mathbf{B} & \mathbf{B} & \dots & \mathbf{B} \\ \vdots & \ddots & \ddots & \vdots \\ \mathbf{B} & \mathbf{B} & \dots & \mathbf{B} \end{bmatrix}}_{PH_p \times PH_u} \tag{12}$$

We define a $P \times 1$ vector $\mathbf{t}(k+i|k) = (\sum_{i=1}^{P} y_j(k+i|k)) \cdot [1 \dots 1]^T$ to form

$$T(k) = \frac{1}{P} \begin{bmatrix} \mathbf{t}(k+1|k) \\ \vdots \\ \mathbf{t}(k+H_p|k) \end{bmatrix} \tag{13}$$

Now we are in the position to express the error term in our cost function as

$$E(k) = Y(k) - T(k) = W \cdot Y(k)\,,\; W = \frac{1}{P} \cdot diag\{S_p\} \tag{14}$$

where W is a block diagonal $PH_p \times PH_p$ matrix consisting of blocks $S_p : P \times P$ with

$$S_p(i,j) = -1,\; i \neq j\;,\; S_p(i,i) = 1\;,\; \forall i,j = 1,\dots,P$$

Combining (8) and (14) yields

$$E(k) = \tilde{\Psi}\,\mathbf{x}(k) + \tilde{\Theta}\,\Delta U(k) + \tilde{\Gamma}\,\hat{\mathbf{d}}(k) \tag{15}$$

where $\tilde{\Psi} = W \cdot \Psi\;,\; \tilde{\Theta} = W \cdot \Theta\;,\; \tilde{\Gamma} = W \cdot \Gamma$.

Next, the constraints of the load balancing problem have to be added, which are in the form of linear inequalities (allocation bounds) and linear equalities (load balancing requirements)

$$0 \leq u_j(k+i|k) \leq 1\;\;,\;\; \sum_{j=1}^{P} \Delta u_j(k+i|k) = 0 \tag{16}$$

with $j = 1,\dots,P\;\;,\;\; i = 1,\dots,H_u$. After some manipulations, they can be cast into the MPC optimization as follows:

$$\Omega_1 \cdot \Delta U(k) \leq \omega_1 - \omega_2 \cdot \mathbf{u}(k-1)\;\;,\;\; \Omega_2 \cdot \Delta U(k) = \mathbf{0} \tag{17}$$

$$\Omega_1 = \underbrace{\begin{bmatrix} 1 & 0 & \dots & 0 \\ 1 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \\ -1 & 0 & \dots & 0 \\ -1 & -1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \dots & -1 \end{bmatrix}}_{2PH_u \times PH_u}, \; \omega_1 = \underbrace{\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{2PH_u \times 1}, \; \omega_2 = \underbrace{\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ -1 \\ \vdots \\ -1 \end{bmatrix}}_{2PH_u \times 1} \tag{18}$$

$$\Omega_2 = \underbrace{\begin{bmatrix} 1\dots1 & 0\dots0 & 0\dots0 \\ 0\dots0 & 1\dots1 & 0\dots0 \\ \vdots & \ddots & \vdots \\ 0\dots0 & \dots & 1\dots1 \end{bmatrix}}_{H_u \times PH_u} \tag{19}$$

Finally, it is straightforward to transform our MPC formulation to a standard constrained least squares or quadratic programming formulation (details are omitted due to lack of space). In our simulator, the controller is implemented using either *lsqlin* or *quadprog* solver in Matlab. Their computational complexity is polynomial to the product of the number of machines, and the control and the prediction horizons, hence we expect a negligible overhead at least for medium-sized problems.

We present an example to illustrate the behavior of the MPC controller. Fig. 1 shows simulation results for 3 machines that are all subject to periodic Poisson loads. In this experiment we set $H_u = 5,\; H_p = 10$ and $\mathbf{Q}(i) = \mathbf{I}_p\;,\; \mathbf{R}(i) = 10 \cdot \mathbf{I}_p$, where $\mathbf{I}_p$ is the $P \times P$ unity matrix. The periodic (cyclic) Poisson loads are generated by using periodic (sinus) load profiles corrupted by random job arrivals according to the Poisson distribution. The results show good performance in the sense that the controller manages to keep the expected completion time of both perturbed and non-perturbed machines roughly equal (see right figure); i.e., the controller appears to be capable of dealing with the load balancing problem in the presence of variable and noisy loads, uncertainties and disturbances.

Compared to the controller design in [9], the design in this paper follows a simplified approach and does not consider transient dynamics, does not perform online identification and does not employ an augmented state-model, which enhances disturbance rejection; we leave the full implementation and evaluation of the controller in [9] in the context of parallel queries for future work.

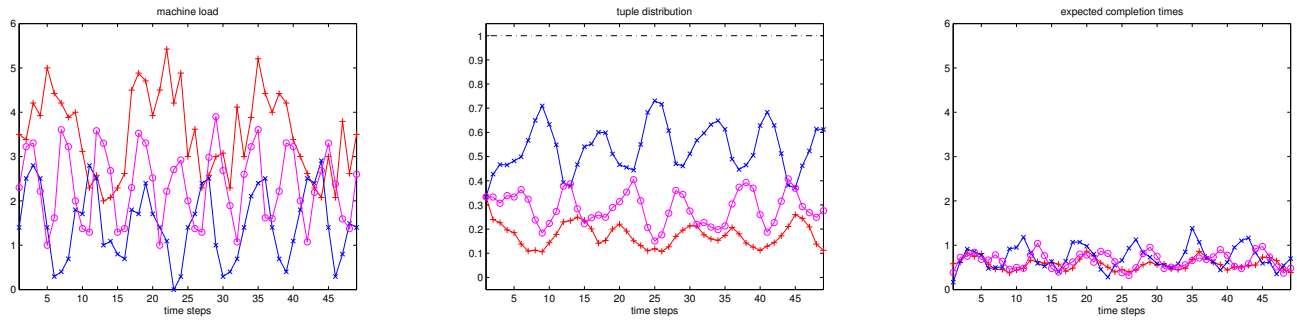

Fig. 1. Example with three machines under periodic Poisson load. Left: the machine load; middle: the tuple allocation; right: the expected completion times.

V. Evaluation

This section compares the performance of the MPC controller described in Section IV with the LQR controller in [6], [7], where the LQR controller has been thoroughly evaluated and showed improved performance when compared against state-of-the-art load balancers, such as those inspired by Flux [3]. In this paper, due to space limitations, we present only a few representative experiments, in exactly the same experiment setting as in [6], [7]. The simulation model is explained in detail in [14]. The simulator monitors the output rate of operators, which is determined by how rapidly the operator itself can process data and the rate at which its children can supply data. Based on the machine contention, which depends on both the external jobs and query execution, $\mathbf{B}$ is produced at each step (or iteration).

To allow direct comparison with the results in [7], the example operator to be balanced throughout the experiments is a parallel hash join, although windowed group-by's seem more appropriate for our case [3]. For the same reason, the query to be examined is a key/foreign-key join over the *Order* (1.5M tuples) and *LineItem* (6M tuples) tables of TPC-H database after having projected out 25% of the columns; this query will be referred to as *Q1*. During the build phase, the hash join is created. Adaptations occur during the pipelined probe phase. When workload allocation changes, then buckets from the hash table move from one machine to another to ensure result correctness. However, the load balancer presented in this work is independent of any particular operator implementation, and we believe that the results apply to any setting in which movements of operator state must follow such on-the-fly changes in partitioned parallelism.

To artificially create random imbalances, we consider two types of random external job arrival: *Poisson*, where the rate of job arrival to (some of) the machines evaluating the join in each iteration follows a Poisson distribution, and *Poisson cyclic*, where the average arrival rate of external jobs follows a Poisson distribution multiplied by a sinusoidal function with a period of 5 secs. As such, the latter type can capture more realistically situations where the external workload is lighter on average but fluctuates periodically. In both cases, the duration of external jobs is 1sec. The randomness of the external load, and the inner complexities of query processing (which are also simulated) result in more realistic, albeit more complex non-smooth load profiles even for the non-perturbed machines, thus posing a significant challenge to load balancing controllers (corresponding figures are presented in [7]).

The performance metric is the overall query response time, which captures the impact of imbalanced execution that is experienced by the user. Each time step, i.e. each controller cycle, is equal to 0.1 secs. In the experiment, the techniques were checked under 10 different random imbalances, and the mean performance values were obtained.

Adaptations of workload distribution are enforced only if the allocation is modified by 5% or more at least on one machine with a view to avoiding overreacting. Due to the rapidly changing imbalances, it might be the case that the load change on a machine between two consecutive steps is higher than a threshold (set to 5%), which implies that the load has not temporarily converged; in that case, no effort to adapt is made to avoid performance degradation. Also, the controllers start enforcing adaptations only after an initial settling period, to avoid oscillations in the starting phase. It is an issue for future work to investigate the exact impact of these heuristics on the performance and the stability of the dynamic load balancing techniques.

Experiments. In the first experiment, the degree of intra-operator join parallelism is 3, and we compare the improvements of the LQR and MPC-based approaches over the non-adaptive case for varying average numbers of external jobs arriving at only one of the machines per second (from 1 to 6). In this scenario, only a single machine is perturbed. The tuning parameters of LQR are those that lead to the best performance in [6]. For the MPC controller, we set $H_u = 5,\ H_p = 10$ and $\mathbf{Q}(i) = \mathbf{I}_p,\ \mathbf{R}(i) = 5\mathbf{I}_p$, where $\mathbf{I}_p$ is the $P \times P$ unity matrix.

The average response times of ten random imbalances for the non-adaptive, LQR and MPC cases are shown in Fig. 2(left-middle). From the figure, it can be shown that both LQR and MPC perform much better compared to the case where, although there is load imbalance, no adaptations are performed. MPC performs slightly better for both types of load, but the improvements are more evident in the case of Poisson cyclic imbalances. For Poisson loads, the average improvement of MPC over LQR is 5%, whereas it is 10% for Poisson cyclic loads, and reaches 26% when 6 external jobs arrive at the perturbed machine per second. Another interesting observation is that the MPC controller is significantly less

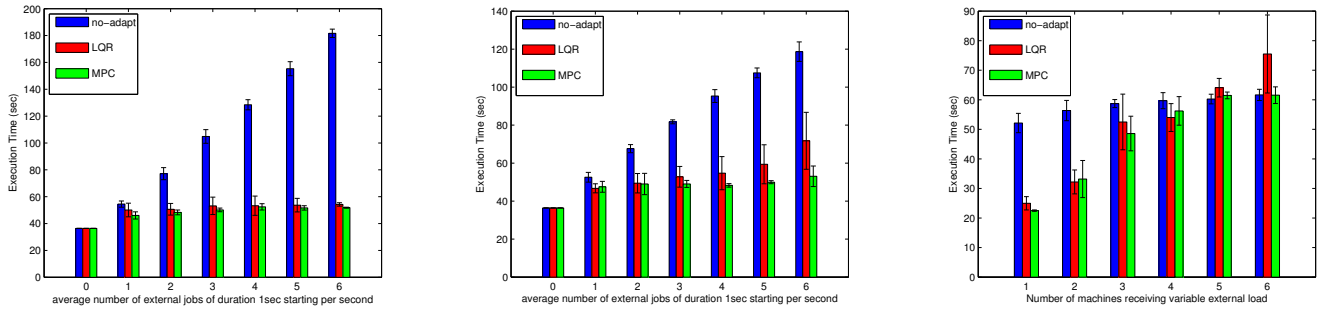

Fig. 2. The average response times for a random Poisson (left) and Poisson cyclic (middle) imbalance for a partitioned execution on three machines, of which one is perturbed. Right: the response times for a random Poisson imbalance when six machines are used and all of them may be perturbed.

sensitive to its parameters. We have experimented with several horizons ($H_u = \{5, 10, 20, 50\}$, $H_p = \{10, 20, 50\}$), with no significant impact on the behavior, apart of course, from the increased optimization cost. The application of a moving average filter and changes in the entries of $\mathbf{R}$ seemed not to have a significant impact either. For all these, no detailed figures and results are presented.

In the second experiment, the degree of join parallelism is set to 6, the average number of external jobs arriving at the perturbed machines is set to 3, and the number of perturbed machines varies from 1 to 6. When more machines are perturbed, the benefits of dynamic load balancing decrease significantly. As shown in Fig. 2(right), LQR may even fail to perform better than when no adaptations are performed. The MPC controller does not suffer from this limitation. Also, for this experiment, the configuration of LQR proved quite difficult (see [7] for details); however, for MPC, the standard configurations mentioned above were sufficiently efficient and there was no need for fine-tuning.

VI. Conclusions

This work presents a novel approach to balancing parallel query execution over machines with unpredictably time-varying behavior and its comparison with current state-of-the-art approaches. The main challenge in this problem is to perform workload adaptations judiciously, because adaptations incur non negligible costs that, if not considered, may lead to performance degradation. Our solution has rigorous theoretical foundation and employs an MPC controller. It improves on a recently proposed LQR-based technique, which was also control theoretical, but suffered from significant limitations in the way the load balancing problem was modeled. Our dynamic balancing technique is characterized by improved performance and effectiveness and less sensitivity to configuration parameters. Our technique is lightweight and can be easily inserted in complete systems, on the grounds that it affects only the distribution policy across partitioned operators. The monitoring information required is the throughput (or the output rate) of partitioned instances, which can be obtained in a straightforward manner (to a certain level of accuracy and freshness). Our ongoing work includes the development of a real prototype in order to evaluate the proposed solution in a real environment considering also additional stateful operators, continuous queries and larger datasets.

Acknowledgements: We would like to thank N. W. Paton from the University of Manchester, for providing the detailed query simulator that we used in our experiments. C. Yfoulis has been supported by the ATEI grant titled "Advanced control of computing systems".

References

[1] D. J. DeWitt and J. Gray, "Parallel database systems: The future of high performance database systems." *Commun. ACM*, vol. 35, no. 6, pp. 85–98, 1992.

[2] G. Graefe, "Encapsulation of Parallelism in the Volcano Query Processing System." in *SIGMOD Conference*, 1990, pp. 102–111.

[3] M. A. Shah, J. M. Hellerstein, S. Chandrasekaran, and M. J. Franklin, "Flux: An adaptive partitioning operator for continuous query systems." in *Proc. of ICDE*, 2003, pp. 25–36.

[4] J. Hellerstein, D. Tilbury, Y. Diao, and S. Parekh, *Feedback Control of Computing Systems*. Wiley, 2004.

[5] X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, P. Padala, and K. Shin, "What does control theory bring to systems research?" *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 1, pp. 62–69, 2009.

[6] A. Gounaris, C. A. Yfoulis, and N. W. Paton, "An efficient load balancing LQR controller in parallel databases queries under random perturbations," in *3rd IEEE Multi-conference on Systems and Control (MSC 2009)*, 2009.

[7] A. Gounaris, C. A. Yfoulis, R. Sakellariou, and N. W. Paton, "Efficient load balancing in partitioned queries under random perturbations," *ACM Transactions on Autonomous and Adaptive Systems, to appear*.

[8] J. M. Maciejowski, *Predictive Control with Constraints*. Harlow, England: Prentice Hall, 2002.

[9] C. Yfoulis, A. Gounaris, and D. Tzolas, "Minimization of the response time in parallel database queries: An adaptive cost-aware mpc-based solution," in *19th Mediterranean Conference on Control Automation (MED)*, 2011, pp. 813 –818.

[10] E. Rahm and R. Marek, "Dynamic multi-resource load balancing in parallel database systems." in *Proc. of 21th Int. Conf. on Very Large Data Bases VLDB.*, 1995, pp. 395–406.

[11] M. Mehta and D. J. DeWitt, "Managing intra-operator parallelism in parallel database systems." in *Proc. of 21th Int. Conf. on Very Large Data Bases VLDB.*, 1995, pp. 382–394.

[12] H. Lu and K.-L. Tan, "Dynamic and load-balanced task-oriented database query processing in parallel systems." in *3rd Int. Conf. on Extending Database Technology EDBT*. Springer, 1992, pp. 357–372.

[13] A. Deshpande, Z. G. Ives, and V. Raman, "Adaptive query processing." *Foundations and Trends in Databases*, vol. 1, no. 1, pp. 1–140, 2007.

[14] N. W. Paton, J. B. Chávez, M. Chen, V. Raman, G. Swart, I. Narang, D. M. Yellin, and A. A. A. Fernandes, "Autonomic query parallelization using non-dedicated computers: an evaluation of adaptivity options," *VLDB J.*, vol. 18, no. 1, pp. 119–140, 2009.

[15] M. Balazinska, H. Balakrishnan, and M. Stonebraker, "Contract-based load management in federated distributed systems," in *NSDI*, 2004, pp. 197–210.

[16] Y. Xing, S. B. Zdonik, and J.-H. Hwang, "Dynamic load distribution in the borealis stream processor," in *ICDE*, 2005, pp. 791–802.